Institut Puig Castellar
Santa Coloma de Gramenet

Desarrollo de un videojuego con Godot/GDScript

“Syoberu”

CFGS Administració de Sistemes Informàtics i Xarxes
CFGS Desenvolupament d'Aplicacions Multiplataforma

Christian A. Lara Canaza
Joan Párraga Castillo
Hamza Jamil

CFGS - DAM 2A

20 Marzo 2018

Licencia de GODOT

Resum del projecte (màxim 250 paraules):

Este proyecto trata sobre el desarrollo de un juego utilizando el motor de juego llamado Godot Engine. Con este motor se pueden hacer juegos en 2D y 3D.

Este motor soporta lenguaje C# aunque tiene sus propios lenguajes de programación nativos que son los siguientes :

- **GD Script**
- **Visual Script**

La idea principal es desarrollar un videojuego en 2D de acción - plataformas donde el jugador deberá avanzar hasta el final del nivel, derrotando a los enemigos y recogiendo los objetos que irá encontrando en el nivel para aumentar su puntuación. A parte de la partida, el jugador podrá acceder a una tienda donde podrá comprar objetos con los puntos que haya conseguido durante las partidas.

Abstract (in English, 250 words or less):

This project is about the development of a game using the game engine called Godot Engine. With this engine you can make games in 2D and 3D.
This engine supports C # language although it has its own native programming languages which are the fzollowing:

- GD Script
- Visual Script

The main idea is to develop a 2D action video game - platforms where the player must advance to the end of the level, defeating the enemies and collecting the objects they are finding in the level to increase their score. A part of the game, the player can access a store where you can buy items with the points you have obtained during the games.

Paraules clau (entre 4 y 8):

Android, Godot, Gd Script, Multiplataforma, Interfaz de Usuario, Juego

Índice

1. Introducción

1.1 Contexto y justificación del Trabajo

La elección que hemos propuesto es un juego multiplataforma para dispositivos móviles y PC, que se desarrollará con un entorno gráfico, utilizando la reciente actualización de Godot Engine 3.0.

La creación de un juego es una tema relevante para todos del grupo. Desarrollar una aplicación de la que tienes una idea clara junto con la actitud positiva, provocará un desarrollo satisfactorio para todos. Para realizar este proyecto hemos acordado aprender nuevos lenguajes como GDScript que será utilizado más adelante en el entorno gráfico (Godot).

Al completar el trabajo realizado queremos una aplicación entretenida y confortable para cualquier tipo de usuario, y al mismo tiempo poder presenciar nuestra aplicación finalizada con una satisfacción alta y contemplar el recorrido que se hace en un proyecto de desarrollo en grupo.

El Juego está catalogado de tipo plataforma, donde se intentará completar niveles para finalizar el juego en su total finalidad. El personaje irá variando según pasa el juego aumentando su poder y experiencia, a la vez que pasa los niveles variarà los niveles y enemigos. Principalmente está orientado para un usuario casual o experto para enfrentarse a los diferentes desafíos que el juego le propone.

1.2 Objetivos del Trabajo

A la hora de cumplir con éxito el trabajo queremos conseguir los siguientes apartados. Son los objetivos que marcados para finalizar el proyecto.

- Conseguir que el juego funcione de forma fluida
- Implementar varios NIVELES
- Añadir recompensas desbloqueables con monedas de juego
- Poner música de fondo mientras juegas la partida
- Compartir tu Score

1.3 Enfoque y método seguido

Visto que se trata de un proyecto nuevo, hemos llegado a la conclusión de que, una vez enfocada la idea del proyecto, la estrategia principal a seguir es repartir el trabajo en dos fases, para resolver con facilidad la gestión de las tareas.

Dos de nosotros trabajarán en el desarrollo del juego, ejecutando y probando que desempeñen las funcionalidades y interacciones propuestas en el anterior apartado.

El otro trabajará la parte del diseño, proponiendo nuevas ideas de funcionalidades y mejoras para implementar en el proyecto, como por ejemplo el diseño o mejora de un personaje. También probará que estas ideas propuestas, funcionen correctamente para la versión final del proyecto.

1.4 Planificación del proyecto

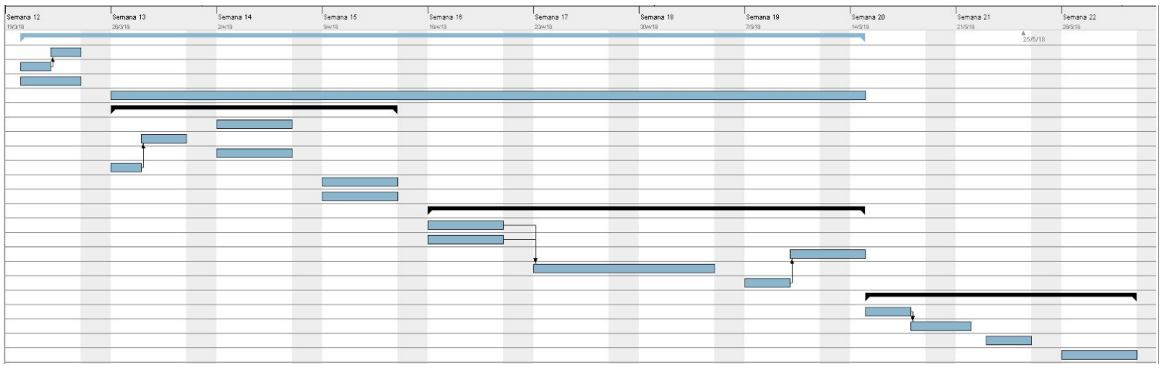
El proyecto lo hacemos en un grupo de 3 personas, la gestión será primordial para cumplir los objetivos y fecha de la entrega, así que efectuará las tareas con un diagrama planificado como referencia principal para alcanzar el trabajo por hacer.

Para realizar este proyecto necesitábamos la herramienta llamada Godot. Se puede descargar este motor de juego 2D y 3D para cualquier Sistema Operativo.

Una vez descargada la versión estable del Godot, realizamos una serie de diagramas para tratar de organizar, orientar y preparar la organización de las tareas.

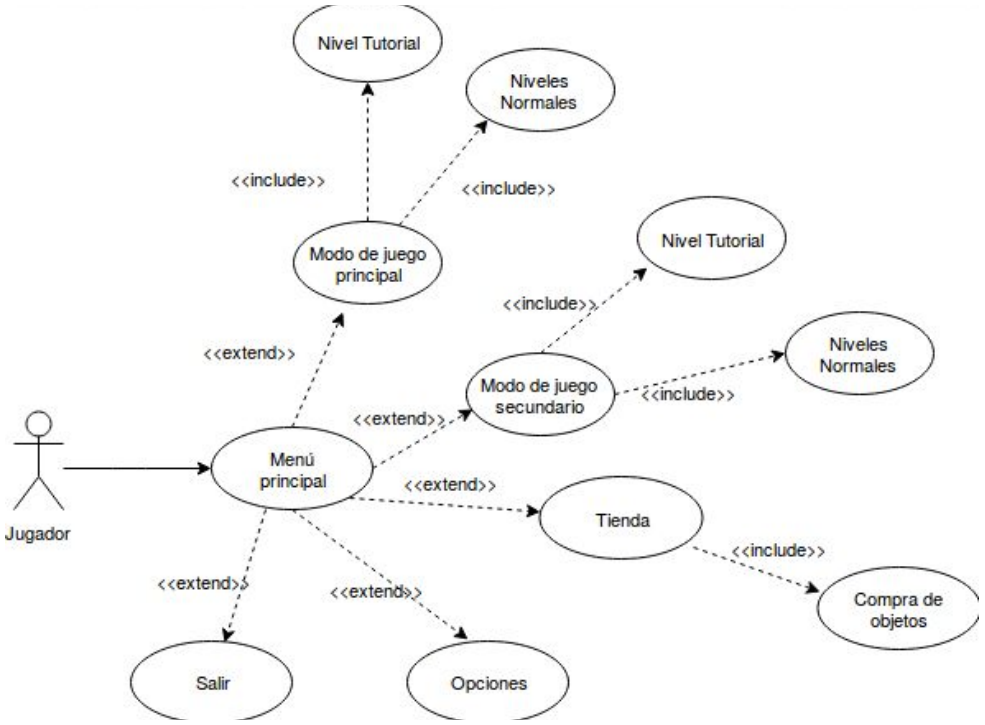
Sin embargo, cabe destacar que el godot es muy nuevo y cada vez sacan una versión más recientes y añaden más cosas nuevas al motor lo le que es muy extenso para poder aprender en este periodo de proyecto, con esto teniendo en cuenta en un principio diseñamos con una cierta viabilidad para poder llevar el aprendizaje del este motor a acabar las tareas propuestas a tiempo.

A continuación adjuntamos nuestro diagrama de Gantt sobre la planificación del proyecto.



Nombre	Fecha de inicio	Fecha de fin
Planificacion Inicial	20/3/18	14/5/18
Elaboración del informe Proyecto	22/3/18	23/3/18
Elaboracion del Diagrama de Gantt	20/3/18	21/3/18
Recoger informacion de Godot	20/3/18	23/3/18
Aprendizaje GDScript	26/3/18	14/5/18
Diseño	26/3/18	13/4/18
Seleccion de Interficies	2/4/18	6/4/18
Busqueda de Sprites	28/3/18	30/3/18
Seleccion de Personajes	2/4/18	6/4/18
Plantear tematica del Juego	26/3/18	27/3/18
Seleccion de Escenarios	9/4/18	13/4/18
Seleccion de Audio	9/4/18	13/4/18
Implementacion del Juego	16/4/18	14/5/18
Implantacion de Interacciones	16/4/18	20/4/18
Implantacion de Animaciones	16/4/18	20/4/18
Implantacion de Ranking	10/5/18	14/5/18
Implantacion de Niveles	23/4/18	4/5/18
Implantacion de Puntuacion	7/5/18	9/5/18
Finalizacion	15/5/18	1/6/18
Test de jugabilidad	15/5/18	17/5/18
Millora de Rendiment	18/5/18	21/5/18
Preparar Exposicion	23/5/18	25/5/18
Revisar Version final de la Memoria	28/5/18	1/6/18

Diagrama de Casos de Uso



Para representar el funcionamiento del nuestro juego hemos hecho este diagrama de clase para explicar cómo funciona nuestro juego.

El juego se inicia dando acceso al menú principal permite elegir entre las siguientes opciones:

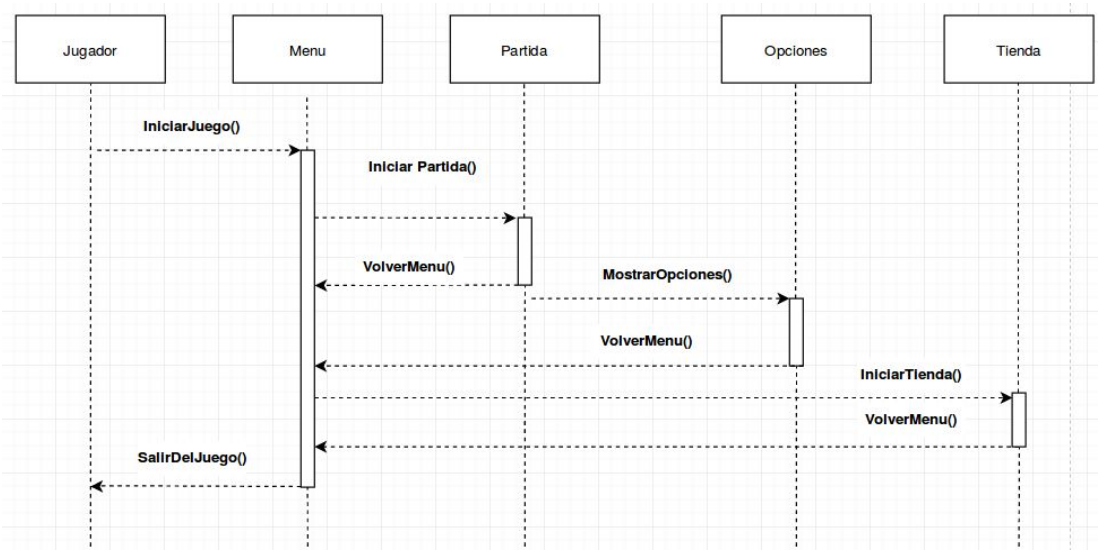
- **Modo principal de juego :** En este modo puedes elegir entre jugar el tutorial o elegir los niveles normales para avanzar en la historia del juego.

- **Modo de juego secundario :** Se trata de un modo supervivencia donde se tendrà que aguantar el máximo tiempo posible en un recinto cerrado con todo tipo de enemigos, con ayuda de tus armas y tu habilidad.

- **Opciones :** En esta opción se encuentran los ajustes del juego como bajar/subir el volumen, cambiar la música de fondo, etc.

- **Ir a la tienda :** En este apartado puedes consultar el dinero conseguido en el juego para comprar objetos relacionados con el personaje con la idea de mejorarlo para que sea más fuerte.

Diagrama de Secuencia



Para mostrar el funcionamiento del menú principal hemos realizado este diagrama de secuencia para ver el funcionamiento en que efectuará nuestro juego.

Cuando el usuario inicia el juego se ejecuta una función llamada **IniciarJuego().** Esta función muestra el menú principal del juego con las opciones que tiene el menú.Dependiendo de la opción que escoja el usuario se ejecutará una función de entre las siguientes:

- **IniciarPartida()** → Cuando el usuario elija en el menú la opción de Iniciar partida, se ejecuta esta función para que se inicie el juego. Luego dentro del juego, cuando pulse el botón para ir atrás, se ejecuta la función de VolverMenu() que permite ir al menú principal del juego.

- **MostrarOpciones()** → Cuando el usuario pulse sobre el botón de opciones se ejecuta este método para mostrar todas las opciones que dispone el juego. Cuando pulsa el botón de ir al menú se ejecuta la función VolverMenu() que permite ir al menú principal del juego.

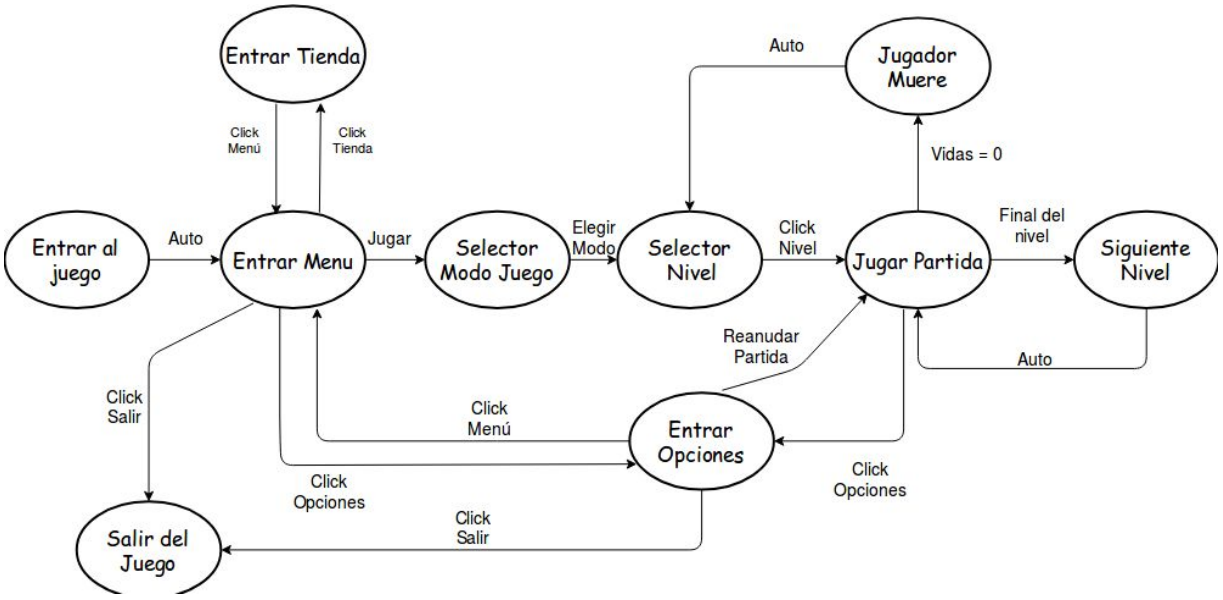
- **IniciarTienda()** → Se ejecuta este método cuando el usuario pulsa al botón de Tienda, lo que muestra todas las armas que posee el jugador y el dinero que ha conseguido en el juego para comprar otras armas.

- **SalirdelJuego()** → Cuando el usuario pulse el botón de salir se llama a método para que se cierre el juego.

Diagrama de Estados

Diagrama: Menú Principal

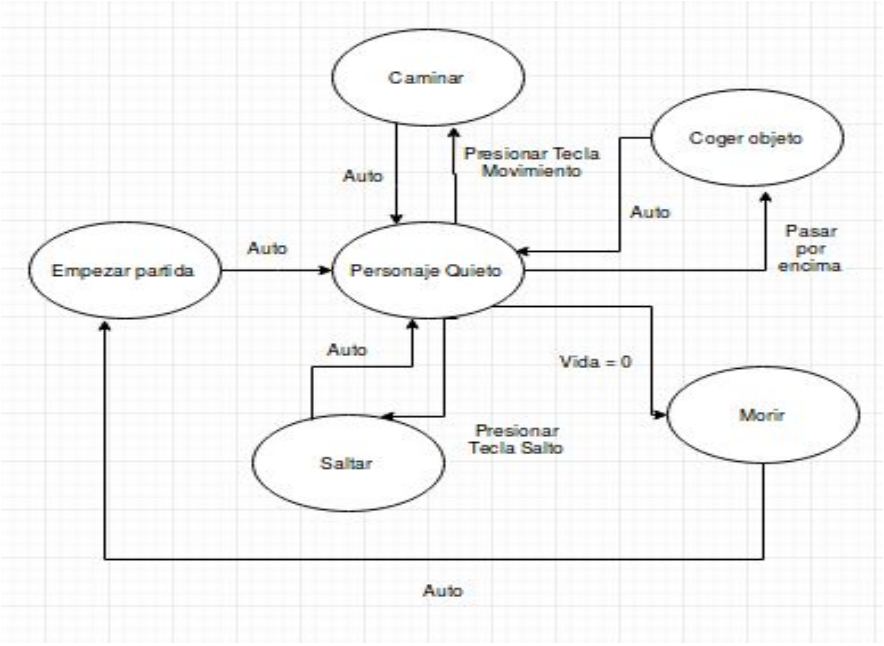
Este diagrama es la representación del menú, es decir, las interacciones de estado qué podrá hacer el usuario. El primer estado que se encontrará es una lista de opciones. Cada opción contarà con características diferentes a la anterior.



Las opciones del menú son: Selector de Modo Juego, Salir, Tienda y Opciones. Cada opción del menú seleccionada hará una acción diferente, por ejemplo abrir las Opciones, confirmar los cambios y volver al Menú Principal. Se explicará más detalladamente la utilidad de cada una de las acciones en los Diagramas de Estados que se exponen a continuación.

Diagrama: Jugador

Este diagrama es la representación de las acciones que podrá realizar el jugador durante la partida. El primer estado será el jugador quieto una vez empiece la partida, y a partir de los botones que presione o las acciones que haga se darán los diferentes estados.



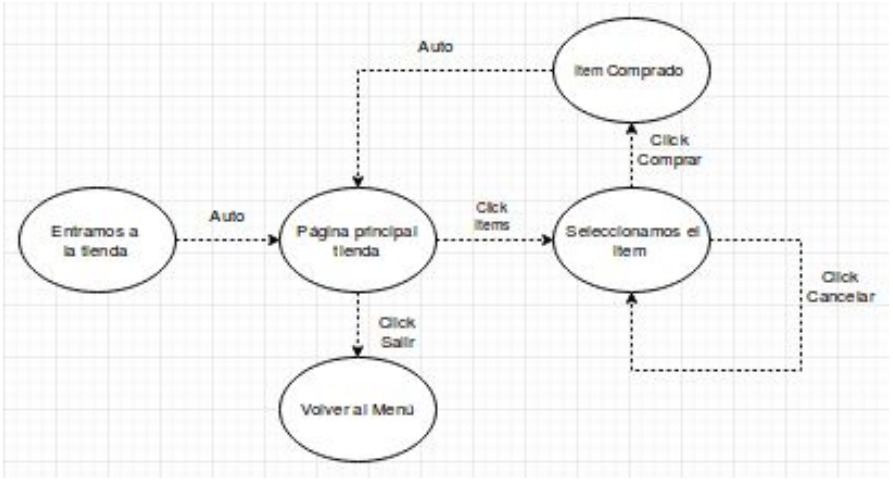
Las opciones del jugador dentro de la partida son: Personaje Quieto, Caminar, Coger Objeto, Saltar y Morir. Cada opción se realizará en función de las teclas o acciones que realice el jugador.

Diagrama: Enemigos

Este diagrama es la representación de las acciones que podrán realizar los enemigos durante la partida. El primer estado que tienen es movimiento automático, y las demás acciones dependen del movimiento que haga el jugador.

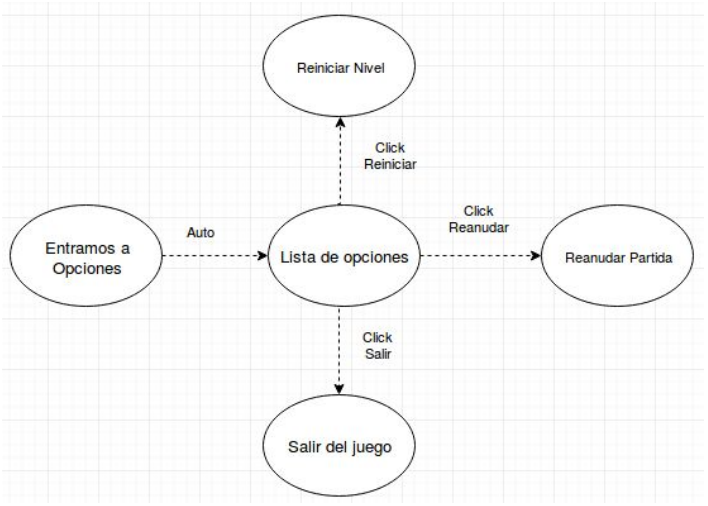
Cuando la partida empieza, el enemigo empieza a moverse automáticamente siguiendo una ruta, si el jugador le ataca, muere y desaparece. Si el enemigo toca al al jugador, le resta vida.

Diagrama: Tienda



Este diagrama es la representación de las acciones que podemos realizar en la tienda. El primer estado tenemos es cargar la página principal de la tienda, después podremos seleccionar los ítems y nos aparecerá una pantalla de confirmación. Si aceptamos volvemos a la página principal del menú principal y si cancelamos volveremos a escoger otro item que queramos.

Diagrama: Opciones



Este diagrama es la representación de las acciones que podrá realizar el jugador cuando entre a las opciones. El primer estado que tenemos es cargar la lista de opciones, podemos reanudar la partida pulsando “Reanudar Partida”, reiniciar el nivel pulsando “Reiniciar” o salir del juego pulsando “Salir”.

1.5 Breve descripción de los otros capítulos de la memoria

El desarrollo de esta aplicación móvil dará inicio a una experiencia para todos, en el ámbito educativo y autodidacta para un desarrollador un juego en 2D. Al realizar un proyecto en grupo, nos formaremos en el aprendizaje del nuevo motor 2D y 3D Godot para llevar a cabo la gestión de un proyecto, elaborando diferentes diagramas para efectuar la tarea de la forma más óptima para el grupo.

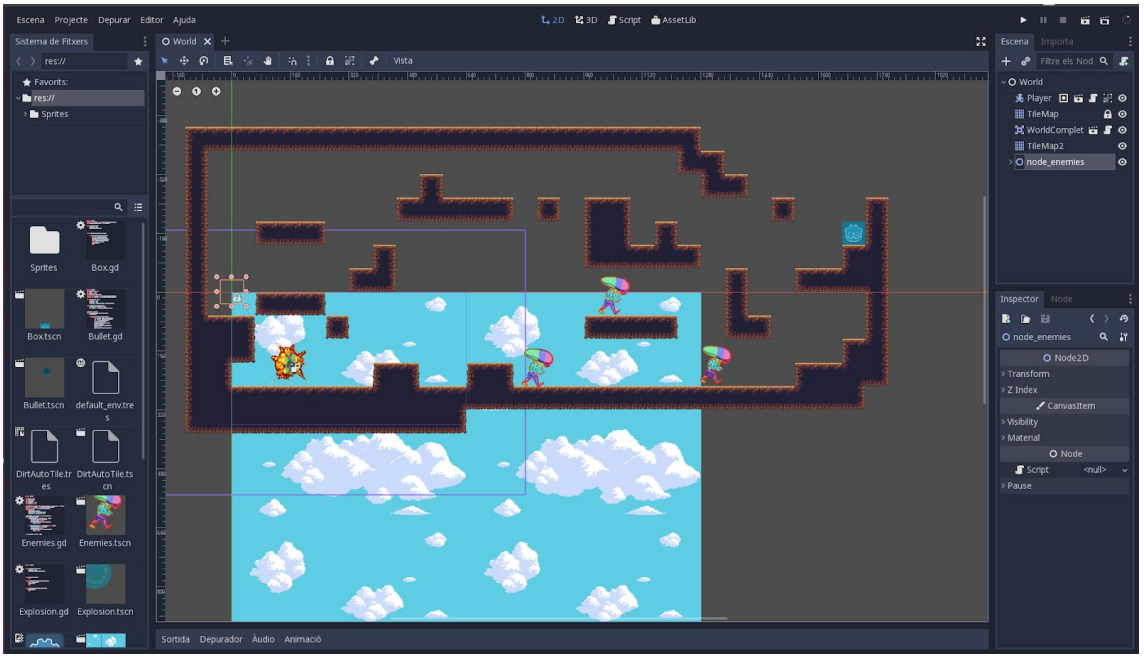
2. Desarrollo

Escenas

Las escenas es una característica muy importante en Godot Engine, ya que contendrá todo el contenido de sus nodos y scripts correspondientes. Esto quiere decir que utilizar esta escena podrá tener acceso a todos los nodos hijo.

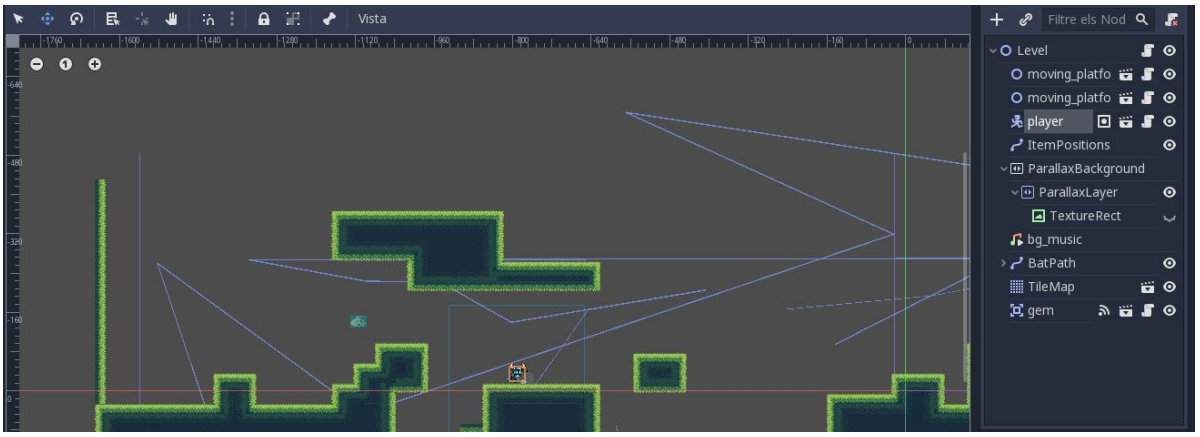
Comienzo del Desarrollo

Para empezar a desarrollar se tendrá en cuenta las características que utiliza la plataforma Godot, como las relaciones que tienen las Escenas y sus nodos entre sí, para que los siguientes conceptos explicado se comprendan intentaré explicarlo con capturas del proyecto final.



Ejemplo de la escena Level_1, donde se instancia las otras escenas creadas anteriormente.

Escena : Escenario de juego



La otra escena se trata del escenario (World). Principalmente contarà con el entorno

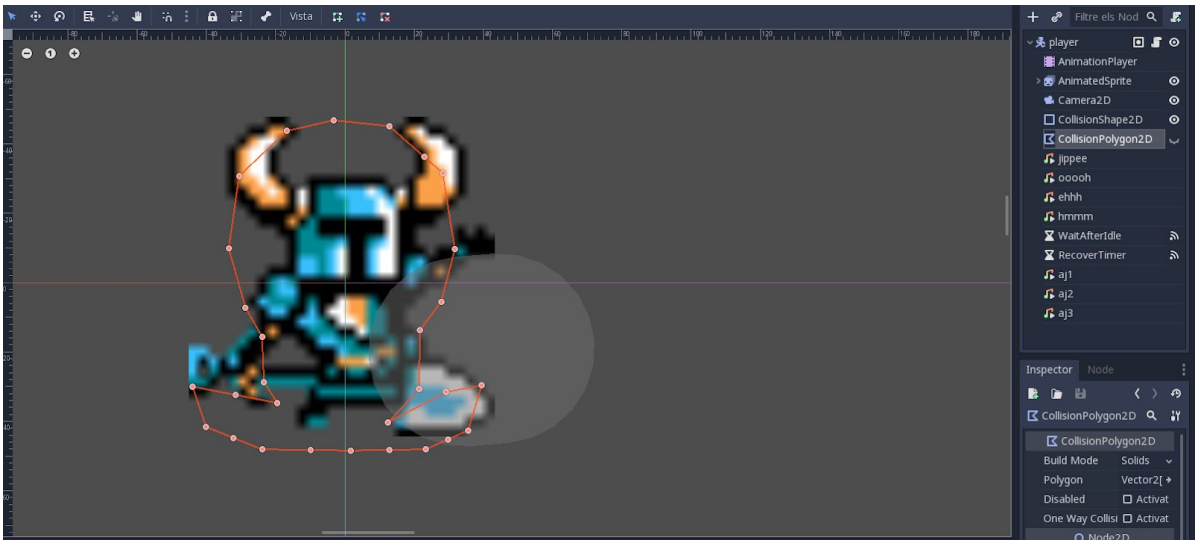
básico d'un escenario, objetos, partículas del escenarios, estructuras, etc… El escenario principal contarà con las instancias realizadas para poder visualizar en el nodo principal, como las escenas que apareceràn son las del Jugador los enemigos etc...

Como se trata de una instancia, en estas escenas no se mostrará los nodos hijos, no preocuparse porque estos nodos estaràn en la escena principal, donde ahí se podrà editar sus parámetros y atributos correspondientes.

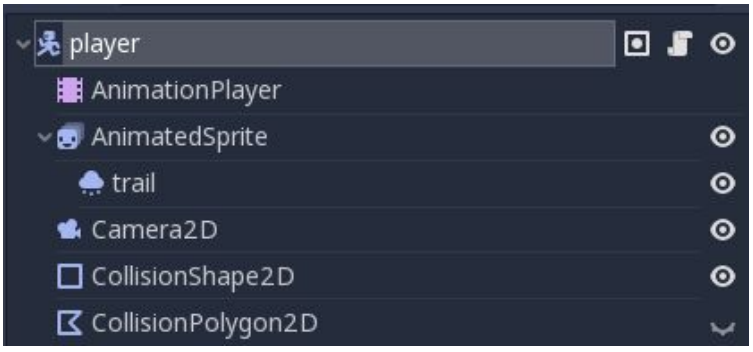
Como se puede ver la escena principal tiene el nombre World, donde el tipo de nodo es ***Node*** y esto nos permitirá poder instanciar todo tipo de nodos hijos, porque es el nodo estándar y padre de todos los anteriores.

Escena : Jugador

Resultado final de la escena Player, en principio todo lo que se ve en la captura es la interfaz del proyecto, junto a los nodos que contiene la escena Jugador.



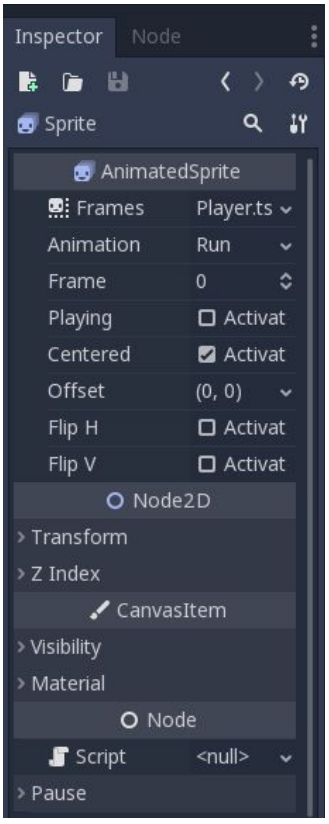
Caja herramientas de godot



La primera escena creada es la del Jugador Principal “Player.tscn”, el Nodo principal que contará es el de ***KinematicBody2D***, el cual nos proporciona un mejor control para un cuerpo controlado, por eso nos servirà para nuestro Personaje Principal. Nuestro personaje necesita una forma, una animación... por eso se tiene que añadir dentro del nodo principal, **AnimatedSprite**.

Al seleccionar el nodo **AnimatedSprite** nos aparecerà un montón de opciones para poder modificar nuestras animaciones. Con estas opciones podremos monitorizar nuestras animaciones dentro de la área del Proyecto.

En este caso podemos primero añadimos un nuevo frame. Al crear la opción *Frames* se abrirà una nueva ventana para poder gestionar nuestras animaciones.
La primera vez después de añadir el nodo, aparecerà vacío ya que por defecto no viene ninguna. Seleccionamos la opción de añadir una animación, después modificamos el nombre según queramos definir esa animación en concreto.

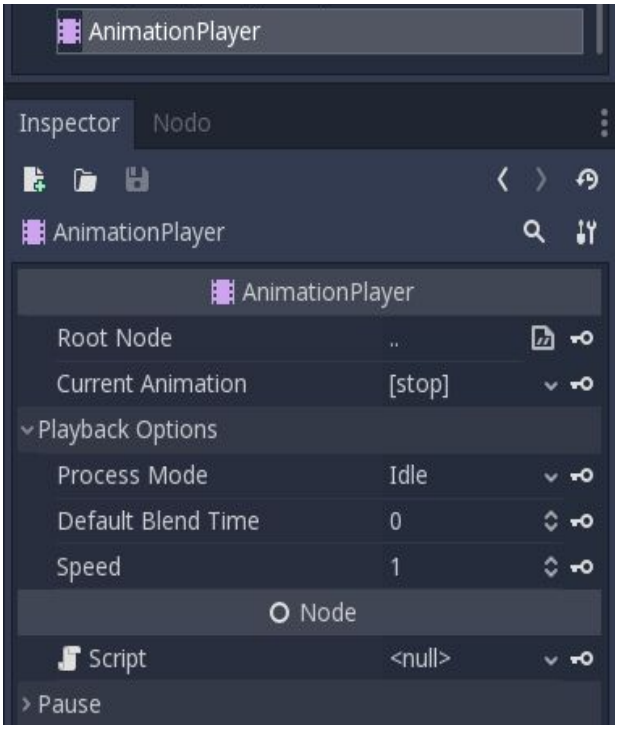


La opción que tomamos es la manera más común de crear animaciones, colocando las imágenes consecuentes para que una vez ejecutado cumpla las animaciones correspondientes. Los sprites del personaje sé tendràn que añadir dentro del

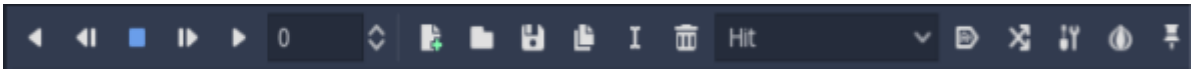
proyecto, preferiblemente en un carpeta, para que esté ordenado según los personajes, objetos, mapas, menús…

AnimationPlayer

Este nodo complementa las animaciones junto al escenario. Estas animaciones serán llamadas como una animación de frames normal, con la función ***.play()***



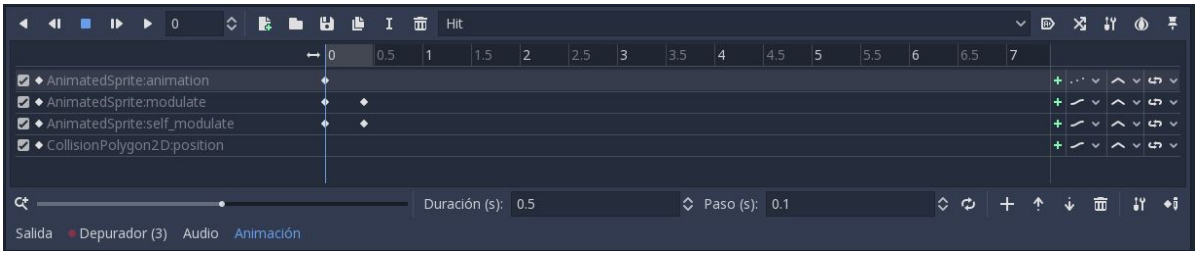
Para que quede todo ordenado e intuitivo creó una nueva escena para la animación. En esa escena se añadirá los sprites que uno quiera, poniéndolo dentro de sus partes padre. Ordenandolo de esta forma nos facilitarà al crear una animación desde cero. Primero añadiremos un nuevo nodo hijo **“AnimationPlayer”**, el cual nos proporcionarà gestionar las animaciones que queramos. Para añadir una nueva animación pulsamos en el icono siguiente (+), lo nombraremos como queramos y ya estaría la plantilla vacía.



Si seleccionamos todos los sprites independientes, y después pulsamos en la llave que se encuentra en la barra de menú principal. conseguiremos rellenar la plantilla de animación.

Al principio nos creará una referencia inicial que tendrà el personaje. Para modificar las posiciones del personaje u otras interacciones las haremos en el entorno gráfico, pero antes de confirmar esa animación habrà que seleccionar el tiempo en que desee que empiece la animación a cambiar, una vez hecho lo anterior y pulsando la "**key**" para confirmar la animación realizada, se podrà ver que se añaden en la línea de tiempo los cambios realizados.

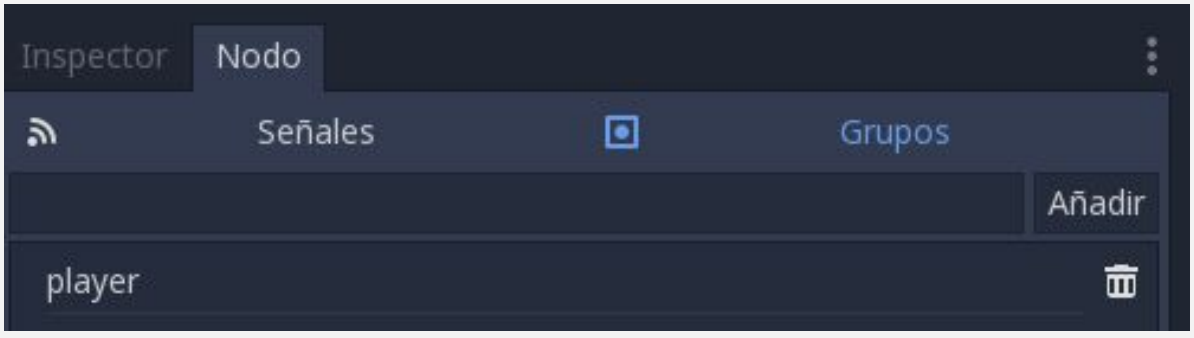
Las animaciones por ejemplo de "**Idle**", queremos que empiece al instanciar la escena así que pulsamos en el botón siguiente (->>).



Ahora es ir jugando poco a poco con las animaciones que queramos conseguir para nuestro personaje.

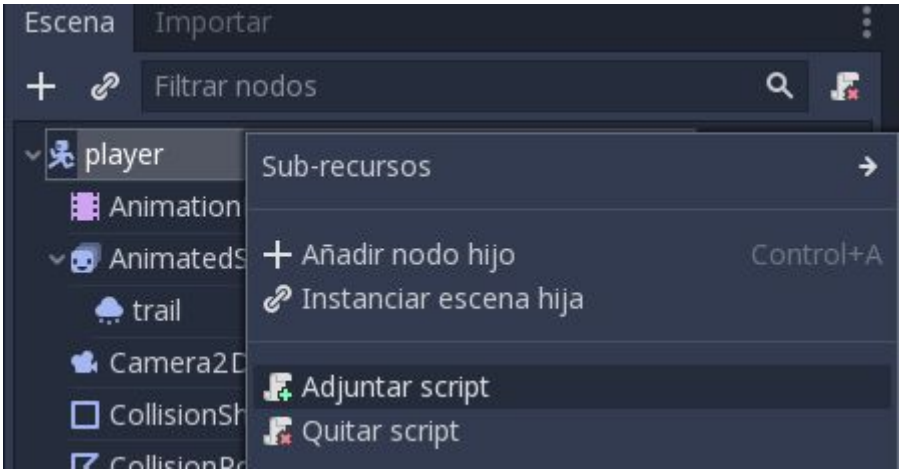
Grupos

Es recomendable añadir las escenas creadas en grupos, para que después podamos reconocerlos y hacer las acciones que queramos, como ignorar colisiones o recuperar posiciones de objetos etc…

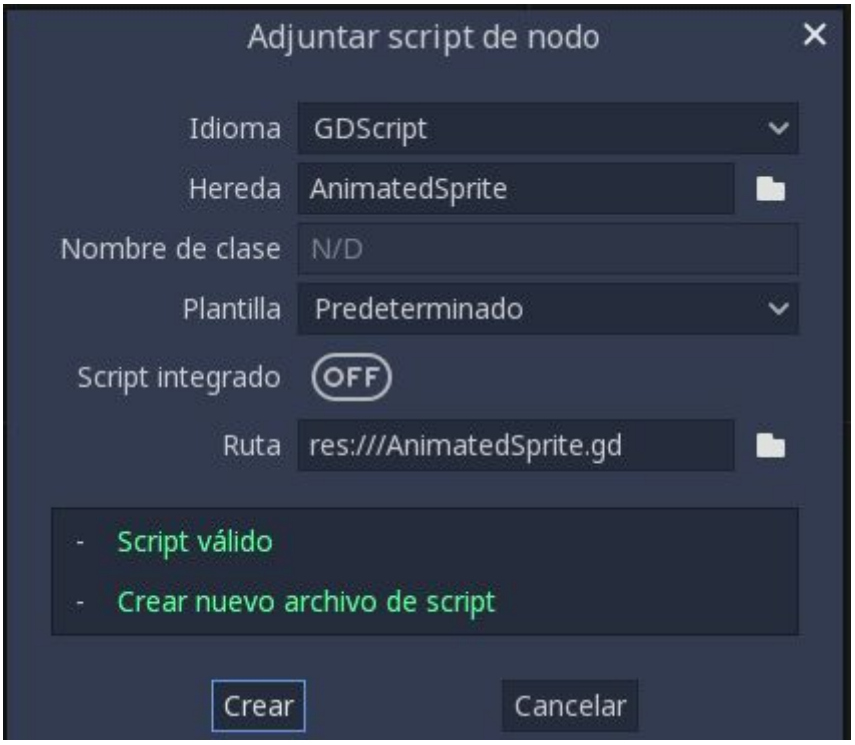


IMPLEMENTACIÓN SCRIPTS

Para crear un script y añadirlo a un node simplemente nos dirigimos al nodo donde queremos añadir el script, click derecho, adjuntar script :



Se abrirà una nueva pestaña. donde podrás modificar el Lenguaje de script, la herencia del script, añadir una nueva plantilla (commentarios, métodos predeterminados, etc...) o la ruta donde se guardará el script. Lo dejaremos por defecto.



player.gd

Por defecto heredara del nodo principal en este caso **KinematicBody2D**. **#_process** se ejecuta una vez por frame, también hay otras variantes, como **_fixed_process**, **_physics_process,**

```
extends KinematicBody2D
```

Constantes

Las siguientes constantes definen el comportamiento del juego, por lo tanto estas constantes no podrán variar durante la ejecución del código.

Como indica el nombre de cada constante, controlan la jugabilidad de la partida , la gravedad, velocidad minima y maxima, velocidad al saltar, o el tiempo máximo en el aire y velocidad al subir una escalera

```
const GRAVITY = 1620.0
const WALK_FORCE = 1600
const WALK_MIN_SPEED = 350
const WALK_MAX_SPEED = 390
const STOP_FORCE = 3000
const JUMP_SPEED = 720
const JUMP_MAX_AIRBORNE_TIME = 0.1
const CLIMB_SPEED = 3.5
```

Variables

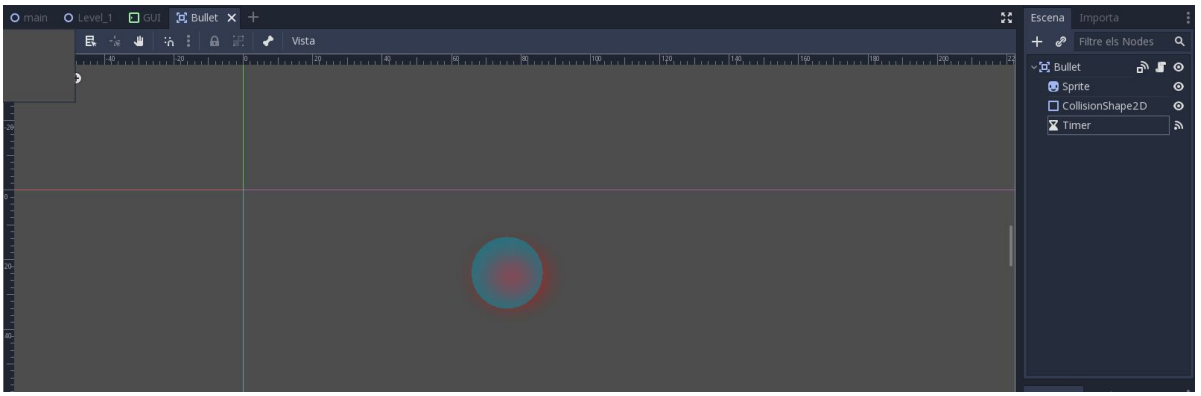
Estas variables son creadas básicamente para poder controlar algunas condiciones durante la partida, como por ejemplo inicializar la velocidad del jugador, variables booleanos para comprobar el estado del jugador etc...

```
var on_air_time =  280
var velocity = Vector2()
var jumping = false
var on_ladder = false
var hitted = 0
var prev_jump_pressed = false
```

Carga de recursos

Instanciar la escena Bullet, esta es una forma de instanciar definiendo la ruta.

```
var Bullet = preload('res://Bullet.tscn');
```



Antes de instanciar la escena bala necesitamos una posición donde se creará, el siguiente método nos devolverá un número el cual serà posición sumada un valor, esto serà la posición final donde el disparo saldrá.

```
func checkDirection():
    var sit
    if $AnimatedSprite.flip_h  == false :
         sit = Vector2(position.x+1000,  position.y)
    elif  $AnimatedSprite.flip_h  == true:
          sit = Vector2(position.x-200,  position.y)
    #devolver la posicion donde se instanciará la escena bullet
    return sit
```

El método se utilizará dentro del método **_physics_process**, donde se comprueba cada frame la dirección del sprite y la posición.

```
func _physics_process(delta):
    var shootDire = checkDirection()
```

Ataque: Disparo

Método instanciar la escena Bullet, se le pasarà la posición donde se instancia la escena, dependiendo de la dirección (izquierda o derecha) del jugador también una velocidad del disparo. Empieza la escena Bullet en su posición, luego se añade al nodo principal como un nodo hijo.

```
func shoot(pos):
	var b = Bullet.instance()
	var a = (pos - global_position).angle()
	var c = Vector2()
	#Si se encuentra mirando para la derecha
	if $AnimatedSprite.flip_h  == false :
		c = position + Vector2(50,20)
	elif  $AnimatedSprite.flip_h  == true:
		c = position + Vector2(-45,0)
	b.start(c, a + rand_range(-0.05, 0.05))
	get_parent().add_child(b)
```

Control de movimiento

Asignaremos una variable por cada tecla pulsada, esto lo comprobaremos cada frame, por lo tanto irá dentro del método ***_physics_process(delta)***

```
var walk_left = Input.is_action_pressed("ui_left")
var walk_right = Input.is_action_pressed("ui_right")
```

Tambien se comprueba si esta subiendo una escalera o no, para poder asignar una nueva variable

```
var climb_up = false
if not jumping:
	climb_up = Input.is_action_pressed("ui_up")
var climb_down = Input.is_action_pressed("ui_down")
var jump = Input.is_action_pressed("jump")
var jump2 = Input.is_action_pressed("ui_down")
var attack = Input.is_action_just_pressed("attack")
```

Gestionar movimiento del personaje, tanto las velocidades aplicando las fuerzas dada, en las 2 modalidades en izquierda y derecha. También se controla cuando la velocidad es 0 (IDLE),

```
var stop = true
if walk_left:
	if velocity.x <= WALK_MIN_SPEED and velocity.x > -WALK_MAX_SPEED:
		force.x -= WALK_FORCE
		stop = false

elif walk_right:
	if velocity.x >= -WALK_MIN_SPEED and velocity.x < WALK_MAX_SPEED:
		force.x += WALK_FORCE
		stop = false
if stop:
	var vsign = sign(velocity.x)
	var vlen = abs(velocity.x)
	vlen -= STOP_FORCE * delta
	if vlen < 0:
	vlen = 0
	velocity.x = vlen * vsign
velocity += force * delta
```

Control Salto Personaje

El siguiente método se llama para modificar el estado del jugador, en resumen es un método auxiliar para gestionar una variable de tipo boolean, y también la velocidad del personaje

```
func jumpBounceAction():
	if (hitted == 1  ):
		velocity.y -= velocity.y + 850
		hitted = 0
```

Modificando variable si se encuentra en el suelo o en un escalera, según el estado en que se encuentre realizará una animación, asignando con el método .play() junto el nombre de la animación. Podrá realizar el ataque 2(**JumpAttack**) si no se encuentra en el suelo o en una escalera.

```
if is_on_floor() or on_ladder:
	on_air_time = 0
	if Input.is_action_pressed("attack") && is_on_floor() :
		print(getAnim())
		$AnimatedSprite.play("attack")
else:
	jumpBounceAction()
#Podra realizar el ataque 2(JumpAttack) si no se encuentra en el suelo o en una escalera
	if(Input.is_action_pressed("ui_down")&& !on_ladder && !walk_right && !walk_right):
		$AnimatedSprite.play("jump2")
```

Para que encuentre movimiento se incrementara a delta

```
on_air_time += delta
prev_jump_pressed = jump
```

Detectar un Tile

Método que nos retornara el nombre del tile en la posición que nos encontramos, en este caso se utiliza para detectar una escalera para poder interaccionar con ella, tanto como subir y bajar. Obtenemos el nodo Tilemap que contiene todos los elementos del mapa

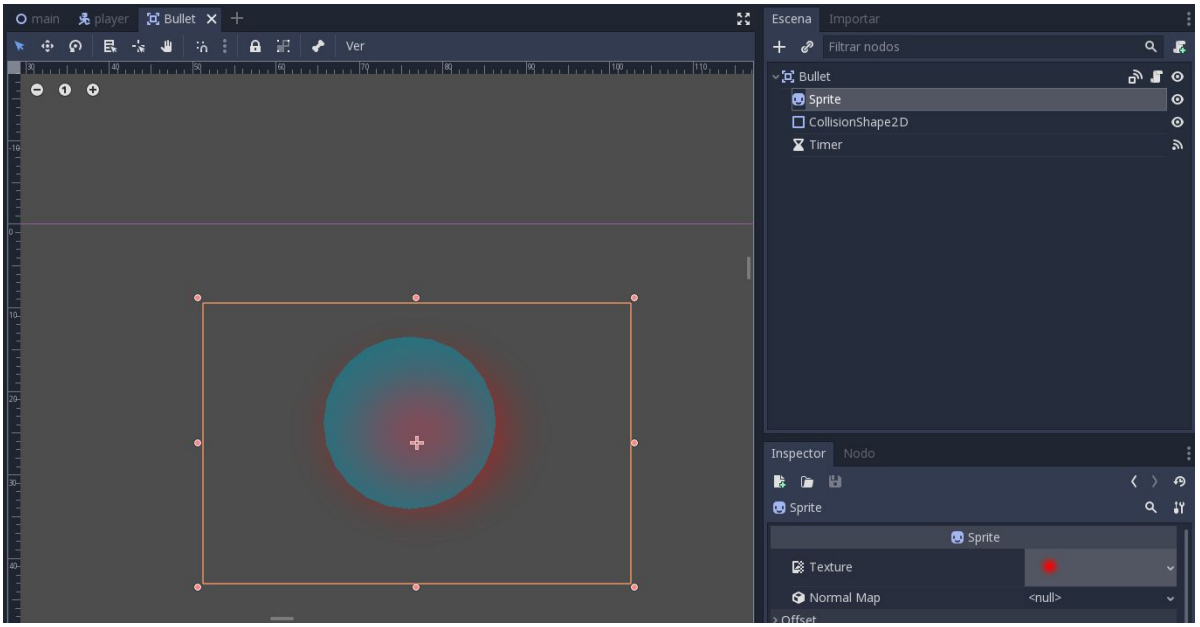
```
func get_tile_on_position(x,y):
	var tilemap = get_parent().get_node("TileMap")
	if not tilemap == null:
		#comparamos la posición del jugador con el tile en concreto
		var map_pos = tilemap.world_to_map(Vector2(x,y))
		# Obtenemos el id de la celda en concreto que se encuentra en la posición
del jugador
		var id = tilemap.get_cellv(map_pos)
		if id > -1:
```

Al saber el id simplemente cogemos el nombre de ese tile en concreto y devolvemos el nombre

```
		var tilename = tilemap.get_tileset().tile_get_name(id)
		return tilename
else:
		return ""
```

Escena: Bullet

Esta escena contiene las características del proyectil, como se trata de un objeto tiene que tener un Sprite y también un **CollisionShape2d** para poder detectar colisiones con otros objetos.



bullet.gd

Variables principales para la velocidad del proyectil, también se encuentra una variable que contiene la escena explosión que es externa

```
export (bool) var is_bullet = true
var speed = 1000
var velocity = Vector2()
onready var explo = load("res://Explosion.tscn")
```

Este método se creó básicamente para acceder desde otra escena externa y poder recuperar la posición en que se inciará el proyectil.

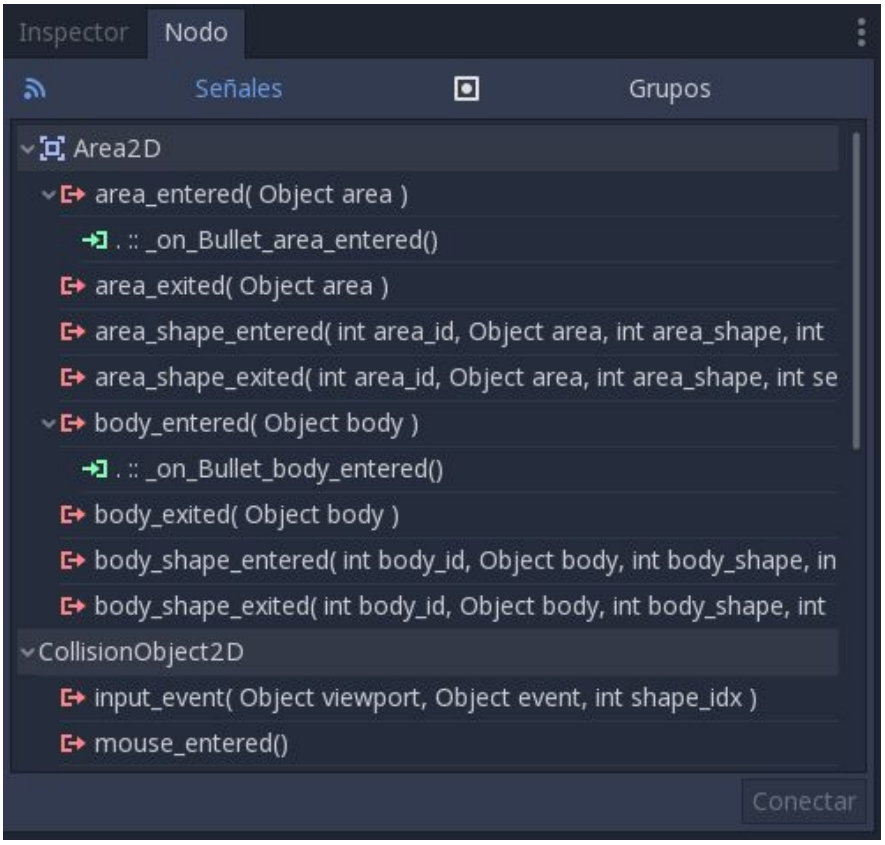
```
func start(pos, dir):
	position = pos
	rotation = dir
	velocity = Vector2(speed, 0).rotated(dir)
```

Dentro del método **_physics_process(delta)** estará la velocidad que tendrá el objeto, por lo tanto por cada frame tendrá una posición nueva.

```
func _physics_process(delta):
	position += velocity * delta
	var bodies =  get_overlapping_bodies()
```

Collision Shape

Para enlazar el método **collisionshape** con el objeto necesitamos entrar en la opción Nodo, y crear el método que queramos en este caso sera **area_entered**, para la detectar áreas al colisionar con la bala, y el **body_entered**, para detectar cuerpos físicos. Tambien se añade el grupo en que pertenece



Este método se ejecutará cuando colisione con un área, en este caso los enemigos, porque son **Nodos** de tipo **Area2D**. Se comprueba el grupo en que pertenece el Área, si es un enemigo se continua, también se instancia una explosión al colisionar con el enemigo. Importante el método queue_free() para eliminarlo y que no ocupe memoria.

```
func _on_Bullet_area_entered(area):
	if area.is_in_group("Enemies"):
                    emit_signal("enemy_hit")
                    var e = explo.instance()
                    e.set_name("scene")
                    var body_element = false
                    e.explosionStart(position,body_element)
                    get_node('../').add_child(e)
                    queue_free()
```

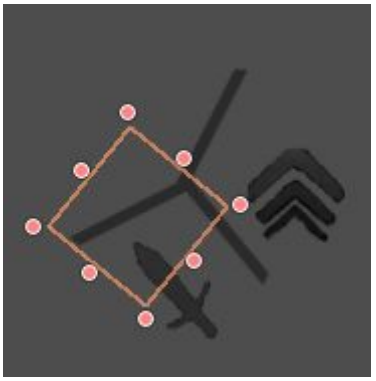
Para crear el mismo método se debe realizar de la misma forma que el anterior paso, pero en este caso se detectara para un nodo de tipo Body, como los TILES del mapa, o un jugador. Se comprueba que sea el mapa y no un jugador, eliminamos el objeto después añadimos la explosión al nodo principal.

```
func _on_Bullet_body_entered(body):
	if not body.is_in_group("player"):
		queue_free()
		var bodies =  get_overlapping_bodies()
		var e = explo.instance()
		e.set_name("scene")
		var body_element = true
		e.explosionStart(position,body_element)
		get_node('../').add_child(e)
```

Creación de Botones para Dispositivos Móviles

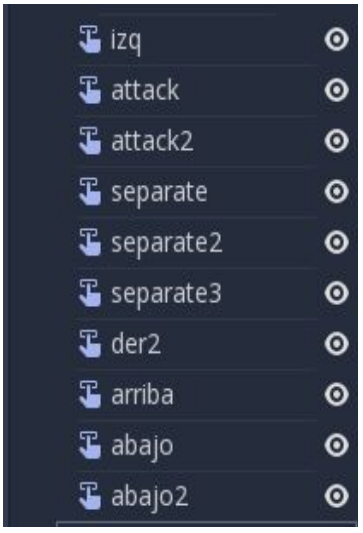
Para crear botones para nuestro juego vamos a **res://** y creamos una carpeta llamada Botones.

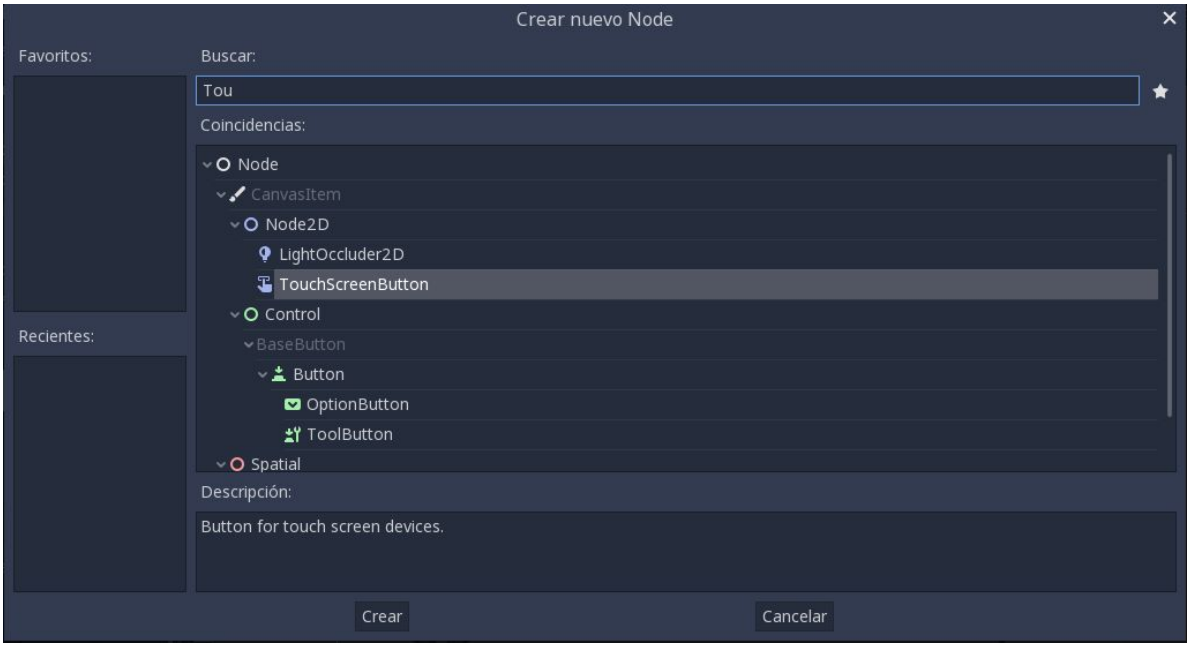
Luego en esa carpeta añadimos todos los sprites de los botones que queremos por ejemplo los sprites que se muestran a continuación para diseñar la interfaz del control del juego. Luego podemos posicionar cada imagen a donde queramos y para girar la imagen podemos hacer Ctrl + click para cambiar el ángulo de la imagen.



Para diseñar la interfaz de botones tenemos que ir a la **Escena** y luego pulsamos al botón de **+** y seleccionamos un **Node** de tipo **TouchScreenButton**. Este nodo sirve cuando en un dispositivo móvil se haga click encima de algún nodo de este tipo haga una acción que se le haya puesto por ejemplo un botón de saltar.

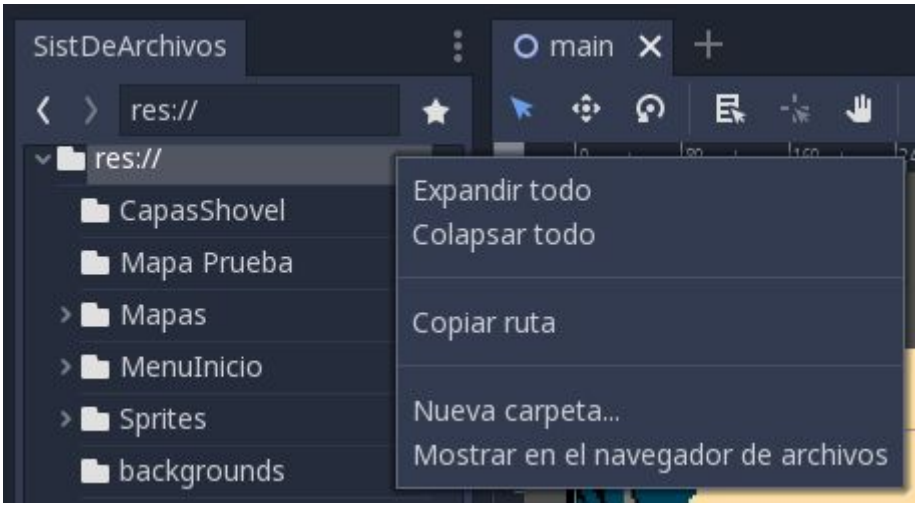
Tenemos que crear un nodo de tipo **TouchScreenButton** por cada botón que queremos añadir en la interfaz del control.



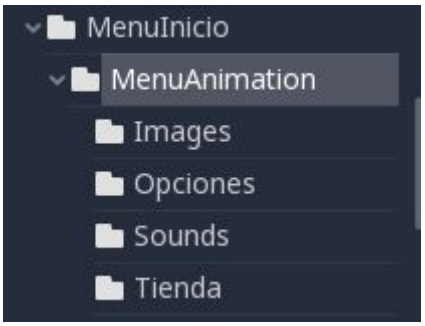


Creación del Menú Principal del Juego

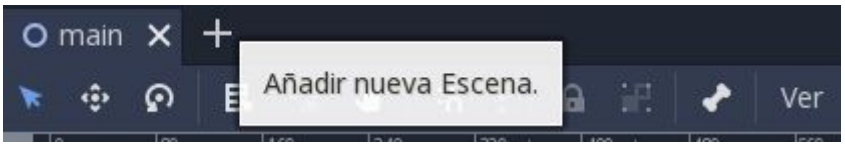
Para crear el menú principal del juego hay que crear una carpeta para tenerlo mejor organizado y ordenado. Para crear la carpeta hacemos **click derecho NuevoCarpeta** y ponemos un nombre por ejemplo **MenuInicio.**



Dentro de la carpeta principal creamos más subCarpetas para acceder a las imágenes , sonido y las escenas que crearemos a continuación.

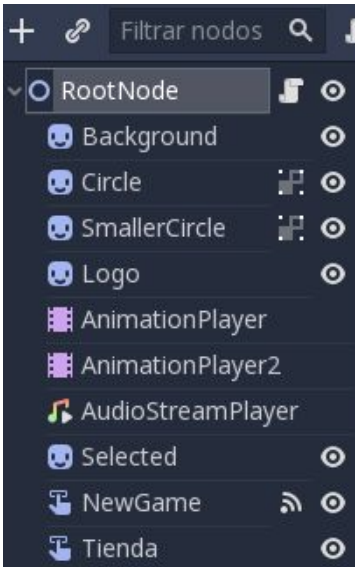


Para crear una nueva Escena pulsamos al botón de **+** y para guardar la escena pulsamos **ctrl + shift + s**, nos saldrá una ventana para guardar la escena elegimos la carpeta MenuInicio y pulsamos a aceptar para guardar.

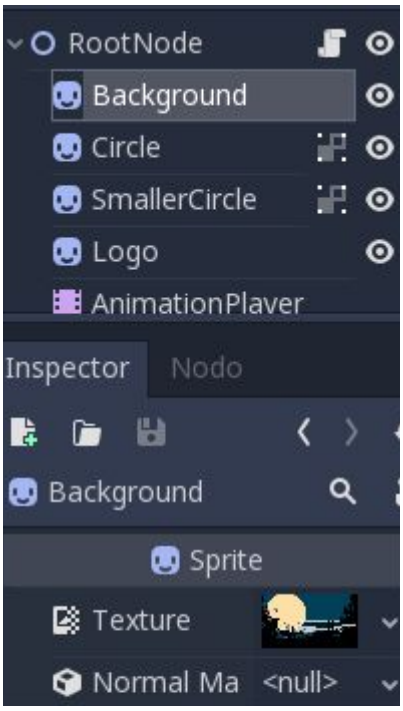


Una vez la escena creada añadimos los elementos necesarios para diseñar el menú. Pulsando al botón del **+** podemos ir creando los siguientes elementos :

- **Nodo 2D :** Este nodo es el padre
- **Sprites :** Sirve para añadir Imágenes de fondo que queramos al menú
- **AnimationPlayer:** Este nodo sirve para crear animaciones en el menú se crean de la misma manera que se ha explicado en el apartado de **Animation Player.**
- **AudioStreamPlayer:** Con este nodo podemos poner una música de fondo al menu y el formato del audio tiene que ser **.ogg** o **.wav**
- **TouchButton:** Este nodo sirve para detectar en dispositivos el click con el dedo



Para añadir una imagen al sprite seleccionamos ese sprite y en **Texture** añadimos la imagen que queramos.



Una vez añadido todos los sprites quedará de esta forma nuestro menú principal del nuestro juego.

Script del Menú para Android y PC

Para hacer que la barra de seleccionar las opciones se mueva arriba o abajo hacemos que el nodo “Selected” qué es la barra que selecciona las opciones mueva la posición abajo o arriba dependiendo del event.

La posición habria que ajustarlo dependiendo de la separación entre las opciones en nuestro caso es de 56 o -56.

El **event** es una acción que recibe por el teclado que puede ser cualquier tecla, pero tenemos que indicar que que cuando el event se flecha para arriba se mueva -56 que sería para arriba en case contrario sería lo contrario.

Hace falta una variable para controlar la posición en la que se encuentra y a esa variable ir sumando +1 o restando -1 dependiendo de la acción que se realice. La posición de esa variable será 0.

A continuación se muestran algunas parte del script con el que funciona el menú.

```
if event.is_action("ui_down") && event.is_pressed() && !event.is_echo():
        if(index !=3):
                index += 1
                x = get_node("Selected").position.x
                y = get_node("Selected").position.y + 56
                $Selected.position.x = x
                $Selected.position.y = y


if event.is_action("ui_up") && event.is_pressed() && !event.is_echo():
        if(index !=3):
                index += 1
                x = get_node("Selected").position.x
                y = get_node("Selected").position.y - 56
                $Selected.position.x = x
                $Selected.position.y = y
```

Cuando pulsemos al **Intro** dependiendo en la posición que este que cargue una escena u otra. Una parte del script que se muestra a continuación hace eso, sería hacer lo mismo por cada una de las opciones que tenga el menú comprobando en la posición que está y abrir una escena u otra.

```
if    event.is_action("ui_accept")    &&    event.is_pressed()    &&
!event.is_echo():
       if (index == 0):
            get_tree().change_scene('res://main.tscn')
       if (index == 1):
              get_tree().change_scene('res:Tienda.tscn')
```

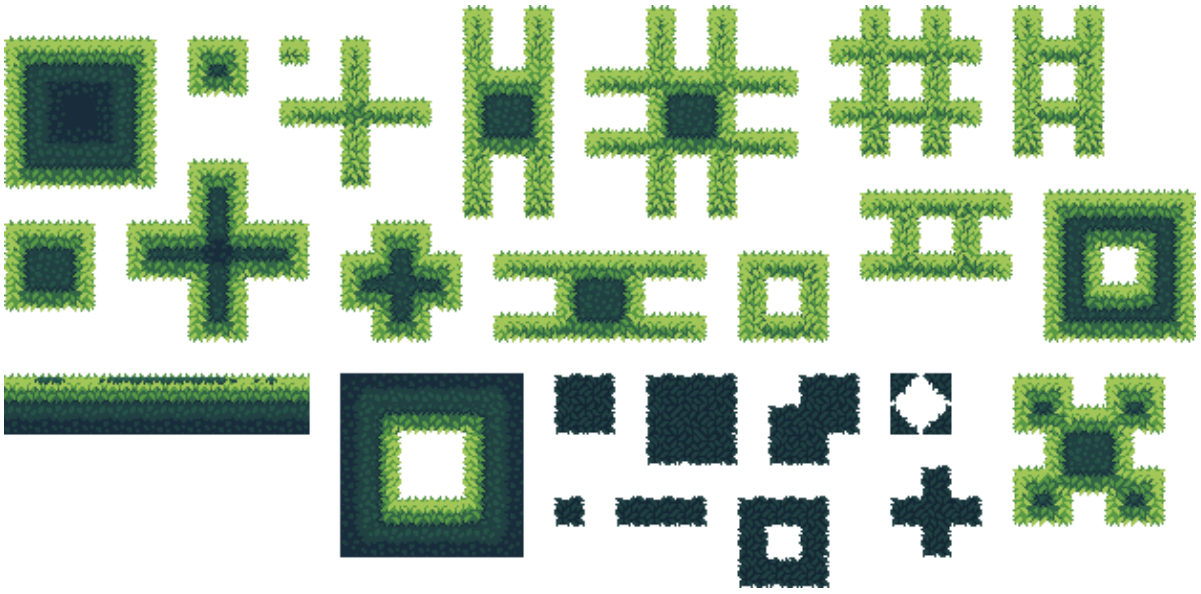
Para Android hay que comprobar si se ha pulsado el botón o no. Los **TouchButtons** que hemos puesto en el diseño del **Menú** tienen unas series de funciones por defecto que nos indican el estado del botón en el que se encuentra.
A continuación se muestra la función del botón **NewGame.**

Esta función sirve para saber si se ha **click** el botón de **NewGame**, si se ha pulsado entonces abre la escena del juego.

```
func _on_NewGame_pressed():
    get_tree().change_scene('res://main.tscn')
```

Creación de mapas

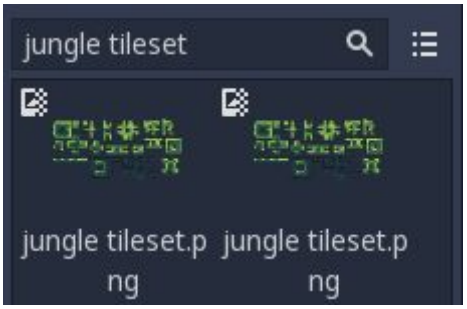
A continuación voy a explicar cómo hemos creado los mapas para nuestro juego. Lo primero que necesitamos es un tiles del map. Para esta explicación voy a usar este.



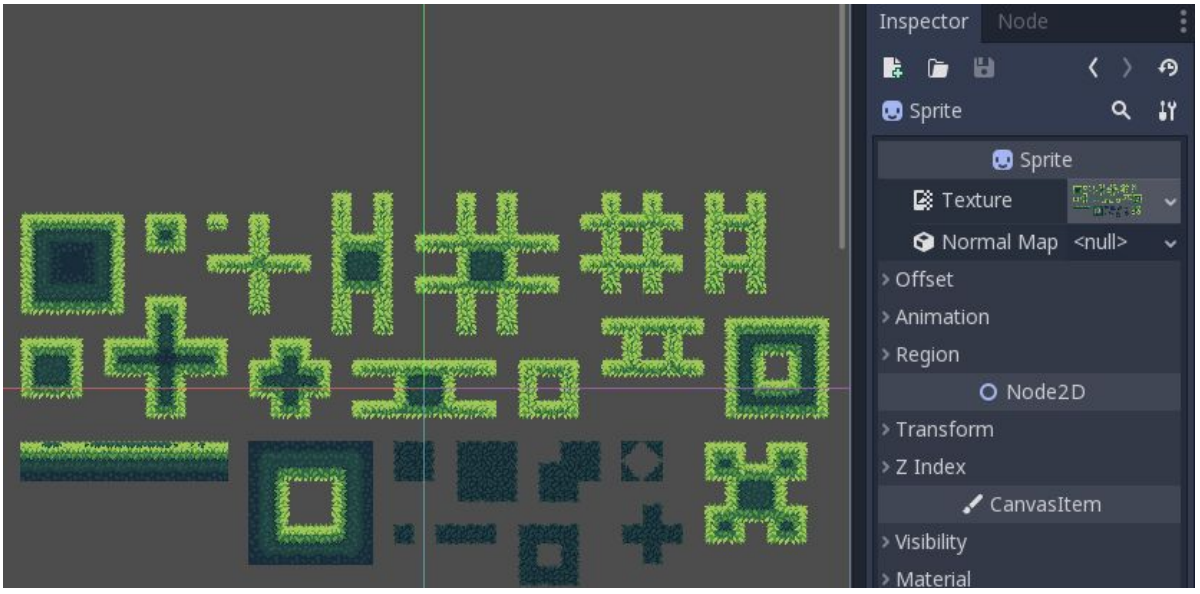
Vamos a Godot, abrimos el proyecto donde vayamos a crear el mapa. Pulsamos el “+” para trabajar con una escena nueva.

Añadimos a la escena un nuevo nodo, en este caso será el nodo, donde su nodo hijo tendrá un Sprite.

Ahora vamos a la parte de la izquierda, y buscamos el nombre que le hayamos puesto a nuestro tiles.

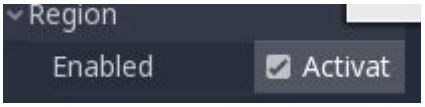


Una vez lo tenemos, lo arrastramos hasta la sección **Texture** del Sprite que hemos creado. Y nos queda así:

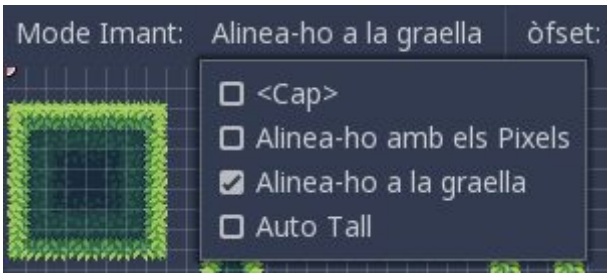


Para importar los tileset y comprobar que esté todo correcto vamos a la parte superior donde Aparece la opcion de "**Importa**", en el menú de configuración seleccionamos 2D Pixel, y continuamos

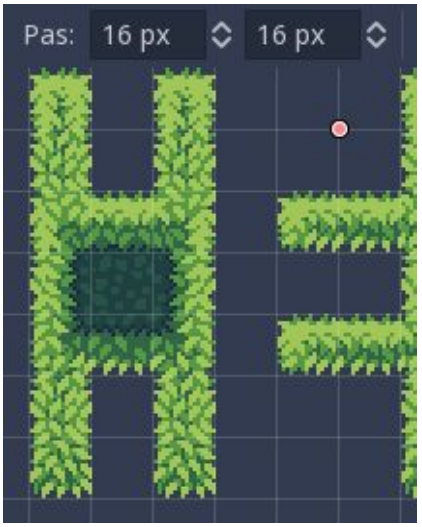
Después volvemos al Sprite, vamos a la sección de Región y activamos “**Enabled**”, para activar el menú de configuración.



Ahora vamos abajo, a la sección de Regió de Textura. Le damos a Mode Imant y seleccionamos la opción de “graella”.

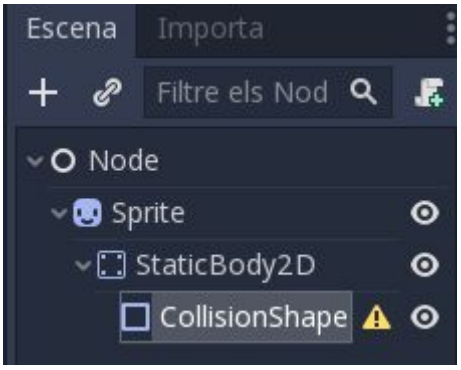


Ahora en el **Pas** tenemos que seleccionar el tamaño de nuestros tiles, en mi caso es 16 x 16. Podemos ver que encaja perfectamente en las cuadrículas.

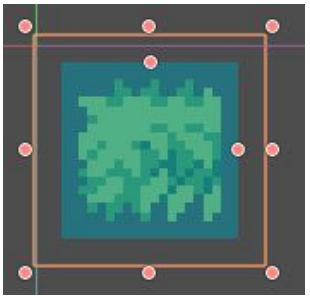


Ahora podemos seleccionar las cuadrículas que queramos y se mostrarán arriba.

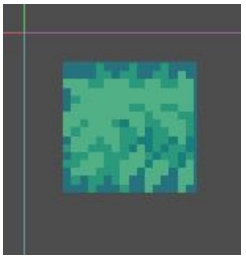
Pero aún no hemos terminado, ahora vamos a poner colisión a este Sprite, así lo podremos usar para el mapeado. Añadimos dentro del Sprite “**StaticBody2D**” y dentro “**CollisionShape**”. Nos quedaría de la siguiente manera.



Seleccionamos el Collision Shape y vamos a la opción de “**Shape**” y seleccionamos “Nou **RectangleShape2D**”. Nos aparecerá esto.



La zona azul es la zona donde colisionará nuestro personaje, lo ajustamos a nuestro sprite. Y nos quedaría así.

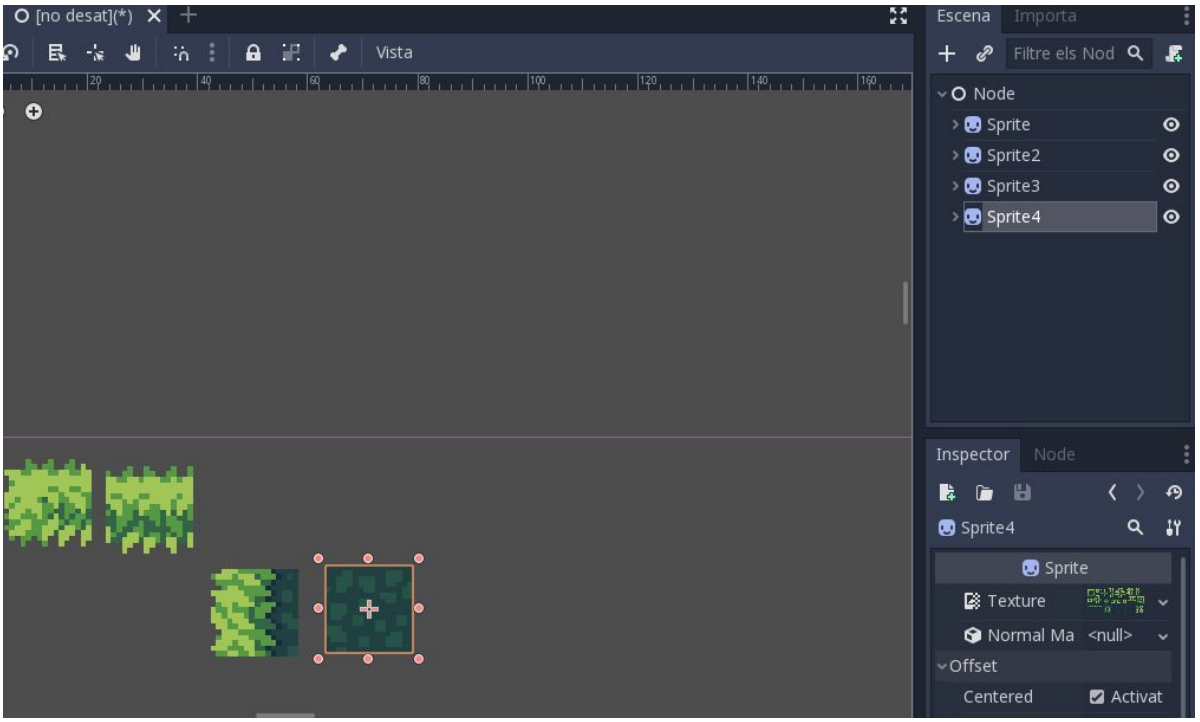


Ahora la ocultamos para que no se vea y cerramos el Sprite.

El resultado final tendría que ser algo así, si no se consigue el resultado deseado recomendamos volver a seguir los pasos.

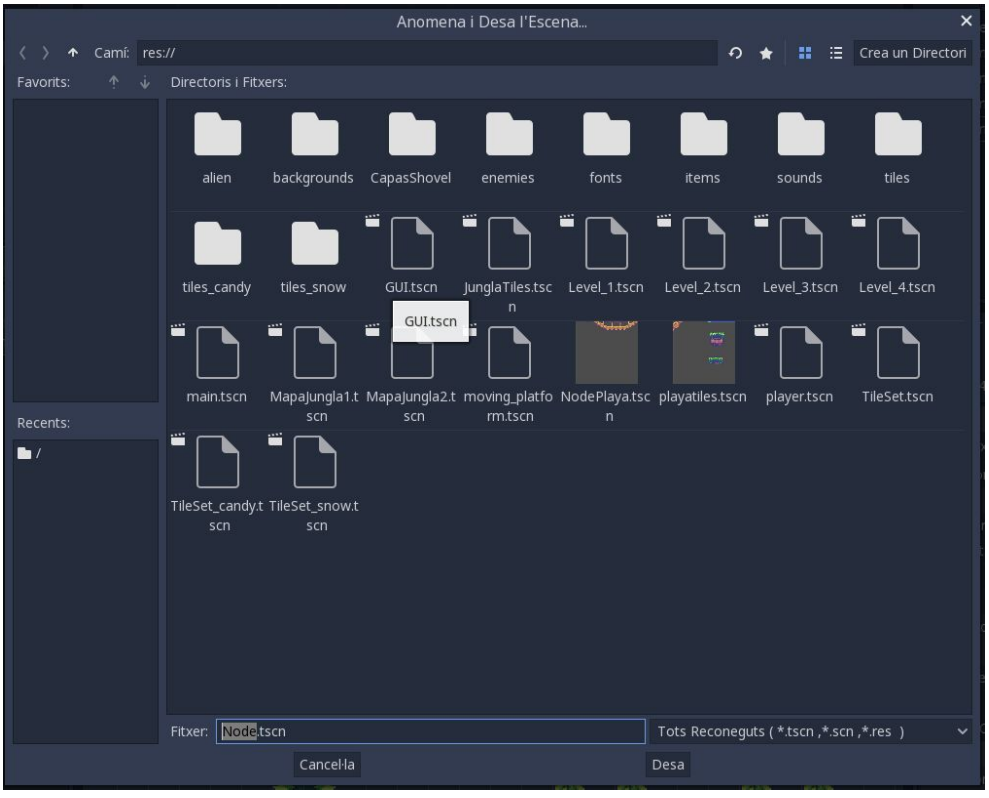
Ahora lo que tenemos que hacer es hacer **Ctrl + D** encima del Sprite para copiarlo, e ir seleccionando las cuadrículas que vamos a usar en nuestro mapa.

Una vez lo hayamos hecho, tendríamos algo así.

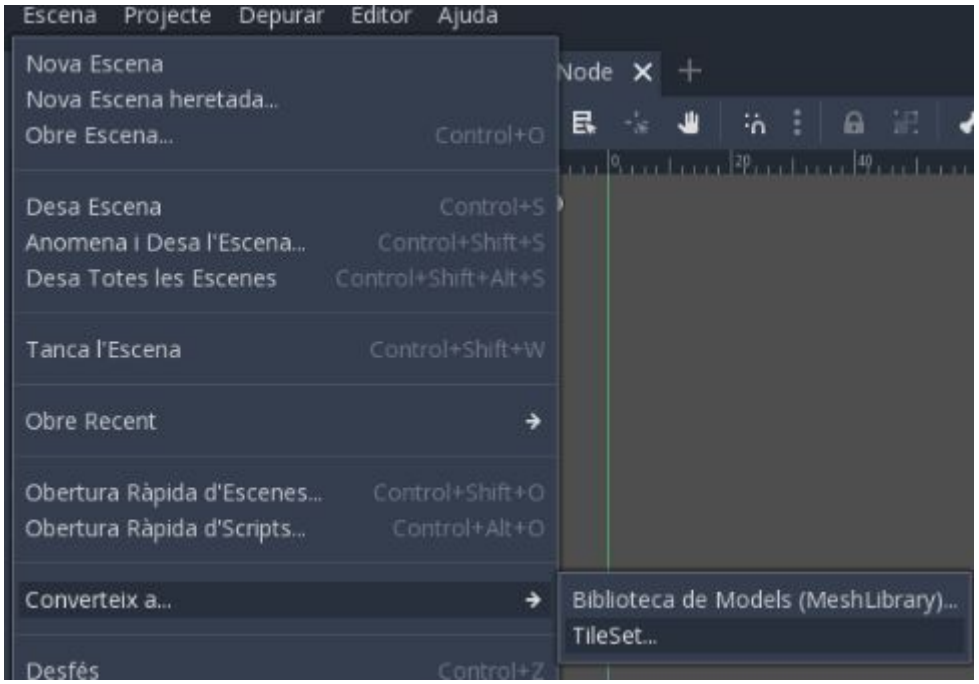


Cuando hayamos terminado de seleccionar todo pulsas unas series de teclas que se muestran a continuación: **Ctrl + S** para guardarlo.

Le ponemos el nombre que queramos con extensión .tscn



Ahora le damos a Escena, Converteix a, TileSet

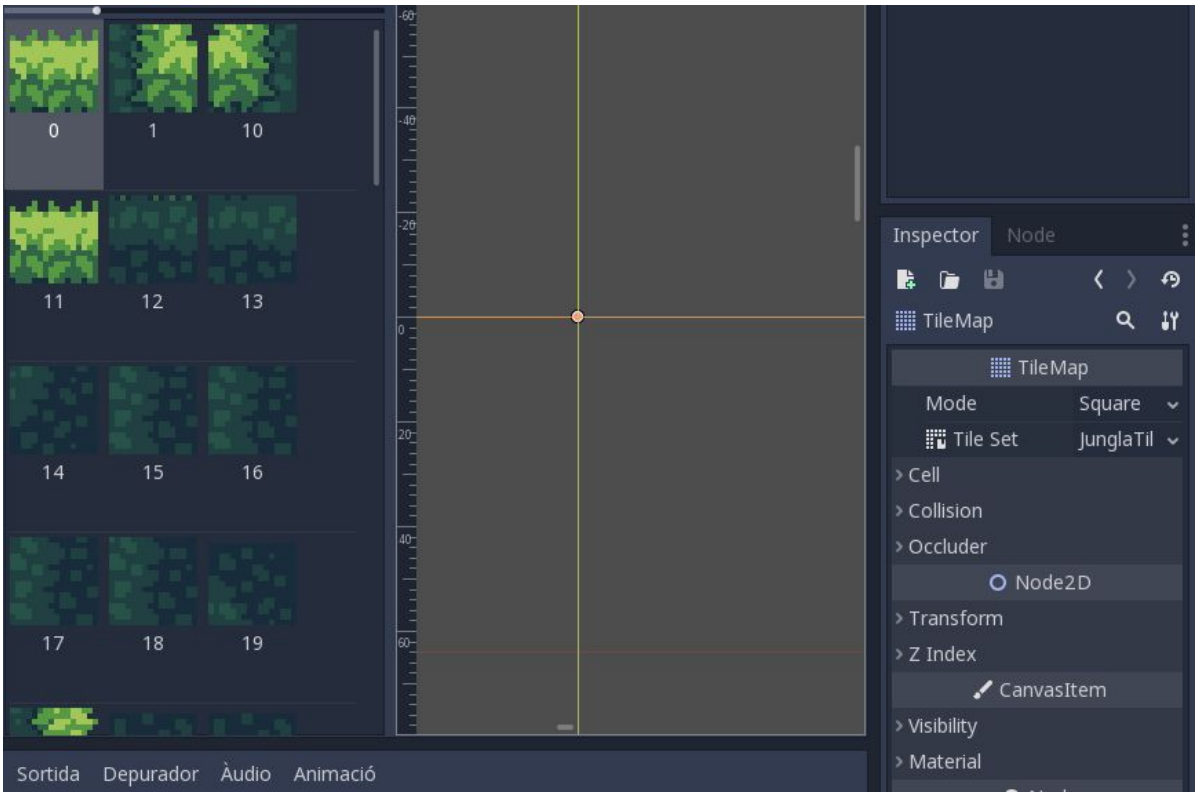


Y lo guardamos con el nombre que queramos y con extensión .tres

Una vez lo hayamos convertido, ya podemos proceder a diseñar nuestro mapa. Volvemos a crear una escena nueva.

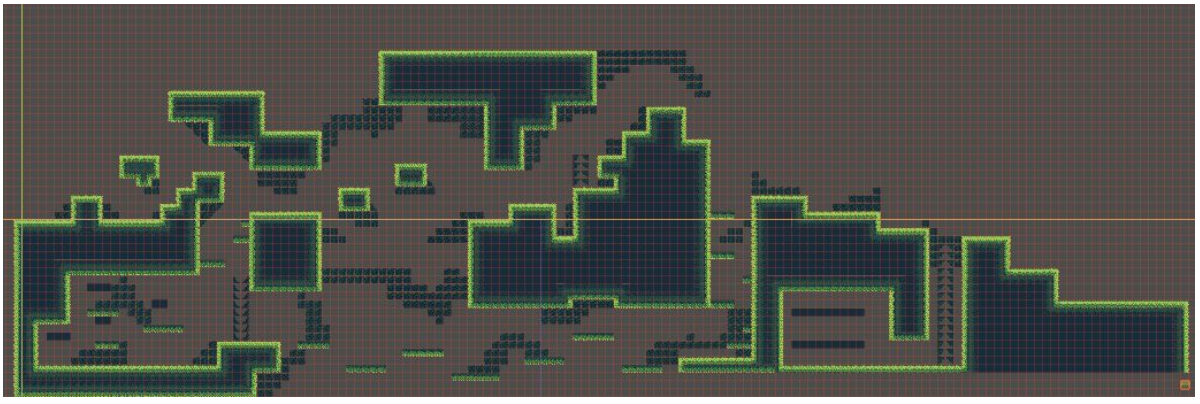
Le añadimos un Node y dentro un TIleMap. Ahora vamos al buscador y buscamos .tres

Y lo arrastramos hasta la opción Tile Set. Una vez añadido a nuestra izquierda nos aparecerán todos los sprite que hayamos ido seleccionando anteriormente.



Para empezar a crear el mapa, solo nos falta ir a Cell y ajustar el tamaño de nuestras casillas. El mio era 16 x 16 así que lo pongo.

Ahora ya podemos ir añadiendo los Sprites creando el mapa que queramos. Este es un uno de los que hemos usado en el juego.



3. Bug Fixing

Durante el proyecto hemos sido víctimas de diferentes bugs que molestaban durante la jugabilidad y acciones de nuestro juego. Algunos de estos fallos han sido bugs con las colisiones, ya que a veces el personaje colisionaba en lugares donde no había nada o ciertos bugs con las acciones que realiza el personaje.

Para solucionar este tipo de bugs, hemos realizado diversas pruebas para localizar dónde estaba el error y poder arreglarlo. Lo primero ver donde estaban ocurriendo estos bugs. Una vez localizado, hacemos varias pruebas cambiando distintos valores de las colisiones o el elemento que falle, hasta encontrar el adecuado.

Una vez solucionado el error, tenemos que comprobar que se ha arreglado correctamente.

Esto nos ha funciona con la mayoría de bugs, aunque tenemos que admitir que no en todas las ocasiones, y debido a ello hay pequeñas acciones o colisiones que hemos tenido que cambiar completamente para no tener estos bugs.

Bug del Menú Principal

Este bug fue el más difícil de arreglar porque a veces la escena dejaba de funcionar y a veces esa misma escena si funcionaba y no sabíamos que podría ser, pero al final nos dimos cuenta que era por la versión del godot que se había actualizado sólo a una versión superior.

Navegando por internet encontramos en una pagina que decia que le había pasado lo mismo y era por la versión del godot y para cambiar bien la escena tenemos que hacer lo de siguiente forma :

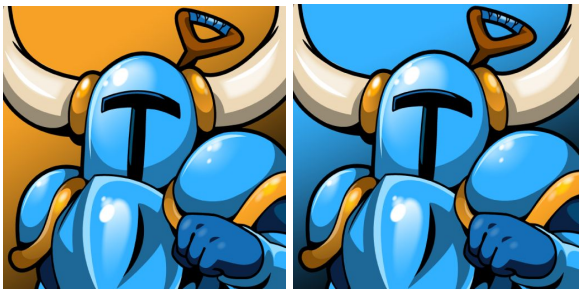
```
get_tree().change_scene('res://MenuInicio/MenuAnimation/main.scn')
```

4. Test A/B

El objetivo de este test AB será escoger el icono ganador para nuestro juego, preguntando a nuestros compañeros de clase. Una vez obtenga el ganador, repetiré el test con una variación del mismo. Cuando tenga el ganador final sacaremos conclusiones del experimento.

1er Iteración

A - B



Número	Observaciones	Puntuación
1	Le gusta más el icono A.	Se queda con el **A** A: **9** B: **7**
2	Le gusta más el icono B. Le gusta que sea todo Azul.	Se queda con el **B** A: **7** B: **8.5**
3	Le gusta más el icono B	Se queda con el **B** A: **7** B: **7**
4	Le gusta más el icono A. No le gusta nada el Azul.	Se queda con el **A** A: **9** B: **6**
5	Le gusta más el icono B	Se queda con el **B** A: **8** B: **8**
6	Le gusta más el icono A	Se queda con el **A** A: **8** B: **6**

7	Le gusta más el icono A	Se queda con el **A** A: **8** B: **7**
8	Le gusta más el icono B	Se queda con el **B** A: **7** B: **8**
9	Le gusta más el icono A	Se queda con el **A** A: **7** B: **5**
10	Le gusta más el icono A. Le gusta mucho que sea naranja el fondo.	Se queda con el **A** A: **10** B: **6**

Una vez terminado la primera iteración, saco los siguientes **datos**:

- Ficha **A**:

Nota media: **8 puntos**
Veces escogida: **6 veces**

Observaciones: Ha gustado en general por el contraste del fondo, ya que les gusta el resultado y queda muy bien el conjunto

- Ficha **B**:

Nota media: **6.85 puntos**
Veces escogida: **4 veces**

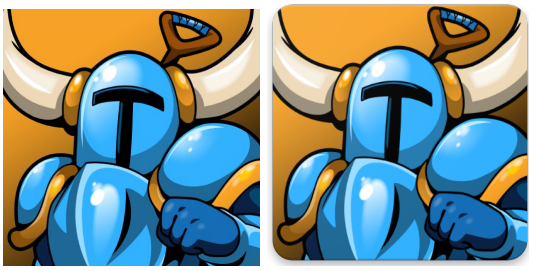
Observaciones: Ha gustado pero no tanto. El icono no tiene buen contraste y cuesta diferenciar.

Conclusión de la iteración:

Ha ganado la ficha **A** con buena ventaja, gracias a su contraste, en el próximo test haremos una prueba con otro icono muy parecido.

2n Iteración

A - B



Número	Observaciones	Puntuación
1	Le gusta más el icono B.	Se queda con el **B** A: **8.5** B: **9**
2	Le gusta más el icono A.	Se queda con el **A** A: **8.5** B: **8**
3	Le gusta más el icono B. Cree que queda mejor con bordes.	Se queda con el **B** A: **8** B: **9**
4	Le gusta más el icono B.	Se queda con el **B** A: **7** B: **9**
5	Le gusta más el icono B. Le parece más un icono.	Se queda con el **B** A: **7** B: **9**
6	Le gusta más el icono B.	Se queda con el **B** A: **8** B: **9**
7	Le gusta más el icono A. No le gusta redondo.	Se queda con el **A** A: **8.5** B: **7**
8	Le gusta más el icono B.	Se queda con el **B** A: **9** B: **10**
9	Le gusta más el icono B.	Se queda con el **B** A: **7** B: **9**

10	Le gusta más el icono B.	Se queda con el **B** A**: 9** B**: 10**

Una vez terminado la segunda iteración, saco los siguientes **datos**:

- Ficha **A**:

Nota media: **7.1 puntos**
Veces escogida: **2 veces**

Observaciones:
Ha mantenido una nota bastante alta, aunque ha sido mucho menos escogido.

- Ficha **B**:

Nota media: **8.9 puntos**
Veces escogida: **10 veces**

Observaciones: Ha gustado mucho, resultando con una nota muy alta. Les ha gustado ya que les recuerda más a un icono.

Conclusión de la iteración:

Ha ganado la icono B con una buena ventaja en elecciones y también en puntuación. Esto ha sido debido a que la primera ya ganó gracias a su **contraste de color**. La ficha B potenciaba lo que había gustado de la primera iteración, añadiendo un borde redondeado, creando así el “icono definitivo” y el ganador.

5. Exportación de APK para Android

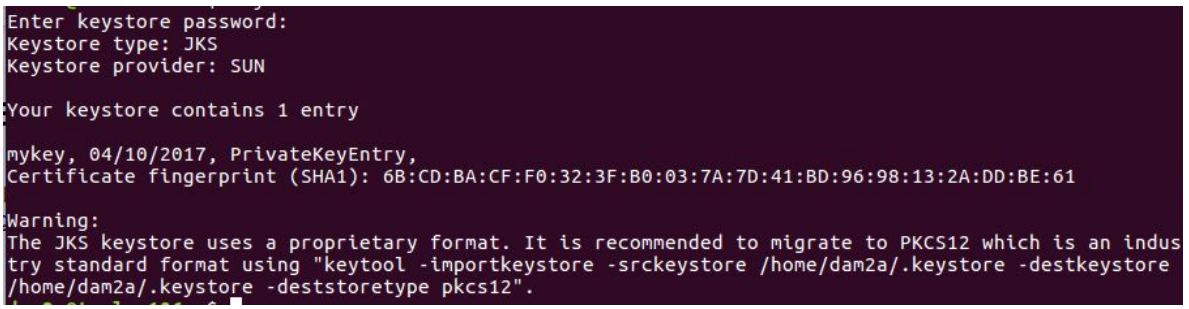
Para generar el APK para android en godot tenemos que hacer unas series de pasos para descargar, instalar y configurar el Godot que veremos a continuación:

- ➔ Descargar y instalar el SDK de android
- ➔ Descargar y instalar el OpenJDK o Oracle JDK que tenga una versión superior o igual a JDK 8
- ➔ Generar una KeyStore
- ➔ Android Debug Bridge (adb)

Generando KeyStore

Una vez descargado y instalado todo, tendremos que crear un Debug.KeyStore que se crea con el siguiente comando :

keytool -keyalg RSA -genkeypair -alias androiddebugkey -keypass android-keystore debug.keystore -storepass android -dname "CN=Android Debug,O=Android, C=US" -validity 9999



Android Debug Bridge (adb)

Qué es ADB

Android Debug Bridge (adb) es la herramienta de línea de comandos utilizada para comunicarse con dispositivos Android. Está instalado con el SDK, pero es posible que deba instalar uno (cualquiera) de los niveles de la API de Android para que se instale en el directorio SDK.

Para los usuarios que usan Windows o Mac el proceso de instalación seria un poco diferente al que veremos a continuación. A continuación se explica el proceso de instalación de **ADB** para **Linux.**

Instalación adb para Linux

- Descargar el archivo ZIP ADB para Linux
- Extraiga el archivo ZIP en una ubicación de fácil acceso (como el escritorio, por ejemplo).
- Abra una ventana de Terminal.
- Ingrese el siguiente comando: *cd / ruta / a / extraído / carpeta /*
- Esto cambiará el directorio donde hemos extraido el archivos ADB
- Entonces, por ejemplo: cd / Users / Doug / Desktop / platform-tools /
- Conecte su dispositivo a su máquina Linux con su cable USB. Cambie el modo de conexión al modo "transferencia de archivos (MTP)". Esto no siempre es necesario para todos los dispositivos, pero se recomienda para que no se encuentre con ningún problema.

- Una vez que la Terminal esté en la misma carpeta en la que se encuentran sus herramientas ADB, puede ejecutar el siguiente comando para iniciar el daemon ADB: ***. /adb devices***

- De vuelta en su teléfono inteligente o tableta, verá un mensaje pidiéndole que permita la depuración del USB.

Finalmente, vuelva a ingresar el comando desde el paso **# 8**. Si todo fue exitoso, ahora debería ver el número de serie de su dispositivo en la salida de la ventana del Terminal.

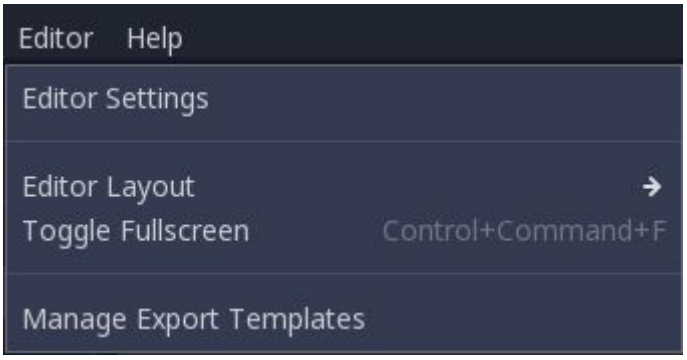
Algunos usuarios de Linux deben saber que puede haber una manera más fácil de instalar ADB en sus computadoras. La guía anterior sin duda funcionará para usted, pero aquellos que poseen una distribución de Linux basada en Debian o Fedora / SUSE pueden omitir los pasos 1 y 2 de la guía anterior.

Los usuarios de Linux basados en Debian pueden escribir el siguiente comando para instalar ADB: ***sudo apt-get install adb***

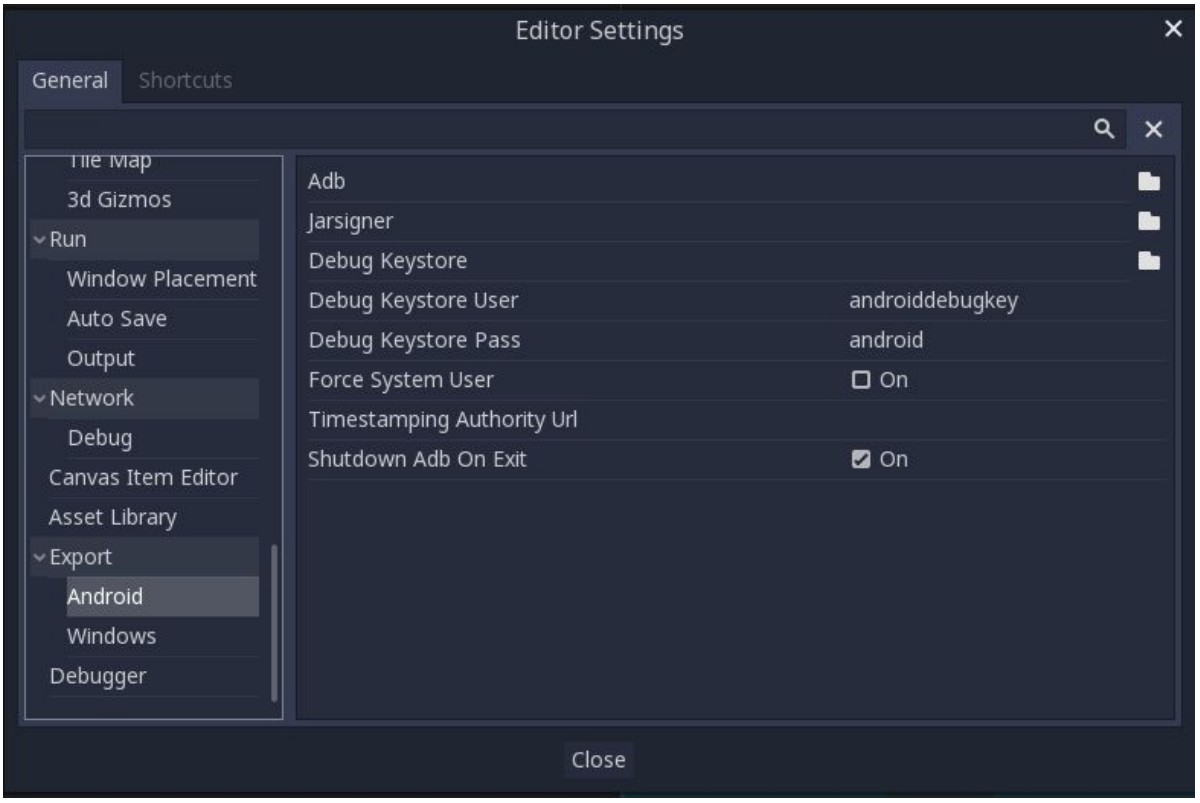
Los usuarios de Linux basados en Fedora / SUSE pueden escribir el siguiente comando para instalar ADB: ***sudo yum install android-tools***

Configurarlo en Godot

Vamos a Configuración del Editor. Esta pantalla contiene la configuración del editor para la cuenta de usuario en la computadora (es independiente del proyecto).



Bajamos hasta la sección donde se encuentran las configuraciones de Android:

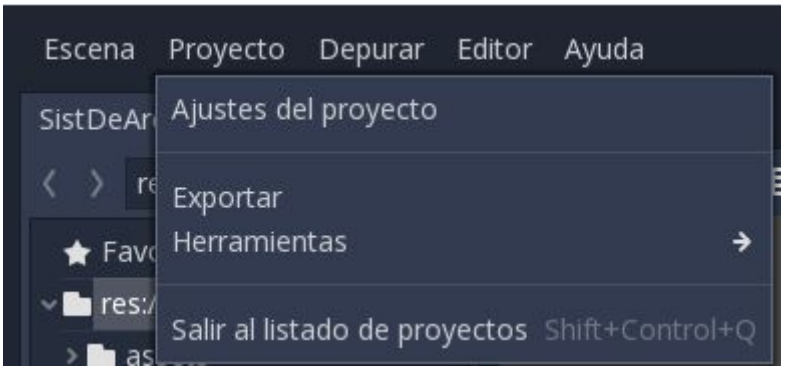


En esa pantalla, se debe establecer la ruta a 3 archivos:

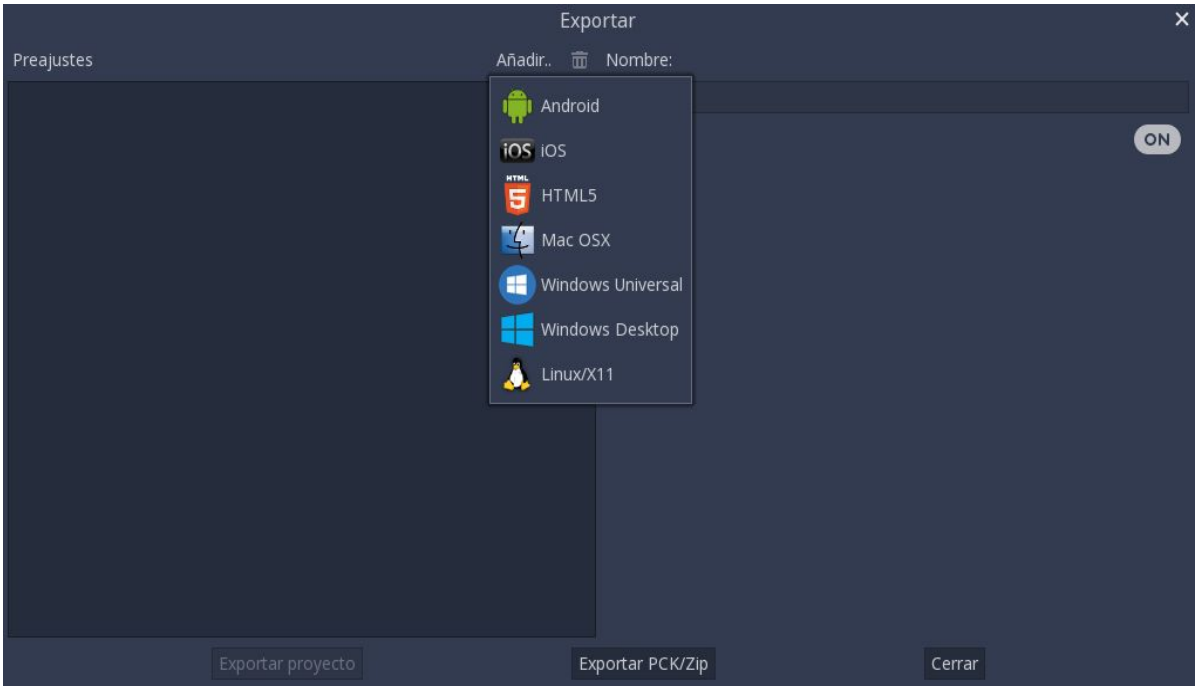
- El ejecutable adb (adb.exe en Windows)
- El ejecutable jarsigner (de JDK 8 o versiones superiores)
- El almacén de claves de depuración (Keystore que hemos creado antes)

Una vez que está configurado, ¡todo está listo para exportar a Android!

Para exportar el **APK** para los dispositivos android vamos a Proyecto y pulsamos a Exportar.



Nos saldrá esta ventana para exportar a diferentes plataformas, En nuestro caso seleccionaremos Android y pulsamos a Exportar Proyecto. Esto nos generará un **APK** para instalar en nuestro **dispositivo Android.**



Para exportar a otras plataformas tendremos que configurarlo de manera adecuada para cada plataforma correspondiente. En nuestro caso exportamos para Android porque lo hemos configurado todo para Android.

6. Conclusiones

Antes de acabar la documentación del proyecto, se pudo comprobar que trabajar en equipo tiene su complicación, por eso es primordial tener un plan extenso planteado al principio, antes de comenzar el desarrollo del proyecto, por lo tanto el no poder llegar a completar el proyecto en su totalidad fue una de ellas, el otro punto a comentar es que el proyecto se realizó en un entorno gráfico, Godot Engine, un engine con gran potencial, y es una razón de porqué decidimos utilizarlo para desarrollar este proyecto, queríamos un reto al desarrollar nuestro primer juego amateur al utilizar este engine, ya que debemos de aprender un lenguaje nuevo GDScript(Python 3.0) y un engine que nunca habíamos utilizado antes, ni en clase o en casa por su parte.

Cada dia al avanzar con el proyecto buscando información para seguir con el proyecto nos encontramos con un gran problema que no contemplamos y no esperamos, una actualización grande del entorno gráfico que utilizamos en el proyecto Godot Engine.

Como dije antes al empezar el proyecto, no esperamos ese gran cambio en el entorno, debido a que esta actualización lleva poco tiempo en marcha, la majoria de documentación no estaba actualizada o directamente no había información oficial sobre los cambios, etc... por lo tanto se tuvo que consultar en páginas no oficiales o en foros que la gente suele comentar sus problemas y resolverlos. Cabe resaltar que la comunidad de Godot ayudó bastante para solucionar algunas dudas al desenvolupar en esta plataforma, pero en ocasiones hemos tenido que solventar las cosas nosotros mismos realizando métodos no muy prácticos, cuando la solución es más sencilla ya que nos lo aporta Godot por defecto o parecidos.

A medida que se acercaba el tiempo de la entrega, hemos tenido que cambiar conceptos de nuestra planificación inicial, optimizando el código y rehaciendo todo hecho lo anterior ya que al final del proyecto adquirimos algunos conocimientos necesarios para optimizar el código y relacionar las escenas de mejor forma, por eso hemos priorizado mejorar el código para que el juego funcione bien en todas las plataformas sin ningún problema.

Nos hubiera gustado añadir más niveles, añadir más profundidad a la historia del juego y sumar algunos personajes más, así como un modo de dos jugadores, pero por tiempo no hemos podido incluir todo a la versión final del proyecto. Se podría decir que hemos sido demasiado avariciosos con el concepto original, pero aun con las dificultades encontradas mediante el desarrollo, hemos aprendido mucho a realizar un proyecto en grupo, tanto como solucionar los problemas que iban apareciendo y estamos contentos con nuestro esfuerzo realizado en este proyecto.

7. WebGrafía

http://godot-doc-en-espanol.readthedocs.io/es/latest/

http://godot-doc-en-espanol.readthedocs.io/es/latest/#sec-tutorials

http://docs.godotengine.org/en/3.0/getting_started/scripting/visual_script/nodes_purp
oses.html

http://docs.godotengine.org/en/3.0/getting_started/step_by_step/ui_main_menu.html

https://www.youtube.com/channel/UCNaPQ5uLX5iIEHUCLmfAgKg

http://docs.godotengine.org/en/3.0/getting_started/step_by_step/your_first_game.html

http://www.gamefromscratch.com/post/2015/02/20/Godot-Engine-Tutorial-Part-5-GUI-Programming-Using-Controls-Widgets-and-Containers.aspx

http://codetuto.com/2016/12/godot-engine-game-tutorial-beginners-create-2d-racing-game-1/

8. Glosario

- **Sprite:** una serie de imágenes unidas en un mismo archivo una al lado de otra y que representan al mismo personaje

- **AnimationPlayer**: Un reproductor de animación se usa para la reproducción general de recursos de animación. Contiene un diccionario de animaciones y tiempos de mezcla personalizados entre sus transiciones.

- **Node2D**: Un objeto de juego 2D, con una posición, rotación, escala y le da control al orden de renderizado del nodo. Todos los nodos de física 2D y los sprites heredan de Node2D.

- **AudioStreamPlayer**: Sirve para reproducir música de fondo.

- **Escena**: Ejecutar un juego significa ejecutar una escena. Un proyecto puede contener varias escenas, cada nivel del juego es una escena

- **BoxCollider**: es un componente básico de un GameObject y consiste en una caja
  invisible alrededor de un objeto y trata sus colisiones. A partir de estas colisiones, se pueden tratar en algún script para realizar diferentes funciones.

- **GDScript**: Es el lenguaje nativo de programación del godot, que se encarga de controlar todas las acciones del juego.

- **APK**: un paquete para el sistema operativo Android. Este formato es una variante del formato JAR de Java y se usa para distribuir e instalar componentes empaquetados para la plataforma Android

- **KeyStore**: es un almacén de certificados con claves privadas.

- **JDK**: Provee herramientas de desarrollo para la creación de programas en android que utiliza el java.

- **ADB**: es la herramienta de línea de comandos utilizada para comunicarse con dispositivos Android

9. Anexo

Manual de Usuario

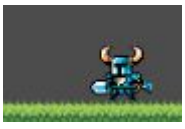
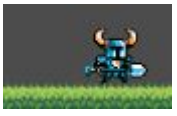
A continuación voy a explicar una guía rápida para el usuario.

New Game:

En esta opción podemos jugar al modo de juego principal. En el debemos recoger todos los coleccionables para avanzar al siguiente nivel. Deberemos esquivar o derrotar a los enemigos, porque si nos quitan toda la barra de vida deberemos volver a comenzar el nivel.

Mientras jugamos al modo de juego principal, podemos hacer realizar diversas acciones:

- Movernos a la izquierda o derecha: Pulsando las flechas de izquierda o derecha.

 o 

- Saltar: Pulsando el espacio o botón de salto (versión android)

- Ataque: Mientras saltamos, pulsamos la flecha de abajo.

- Disparo: Pulsando la tecla Z o el botón de disparo (versión Android)

Survival Game:

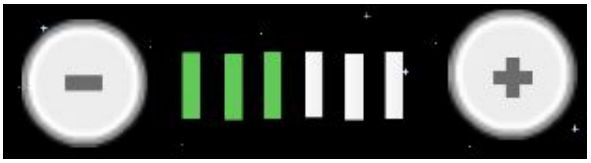
[añadir foto]

En esta opción podemos jugar al modo de juego secundario. En este modo debemos sobrevivir el máximo tiempo posible en un área cerraba mientras van apareciendo enemigos. Habrá un contador que contará cuantos segundos llevamos. En este modo de juego podemos realizar las **mismas** acciones que en el modo de juego principal.

Opciones:

Desde esta opción, podremos acceder a las opciones del juego. En ellas podremos realizar las siguientes acciones:

- Sonido: Podemos subir y bajar el sonido con los botones de “-” y “+”



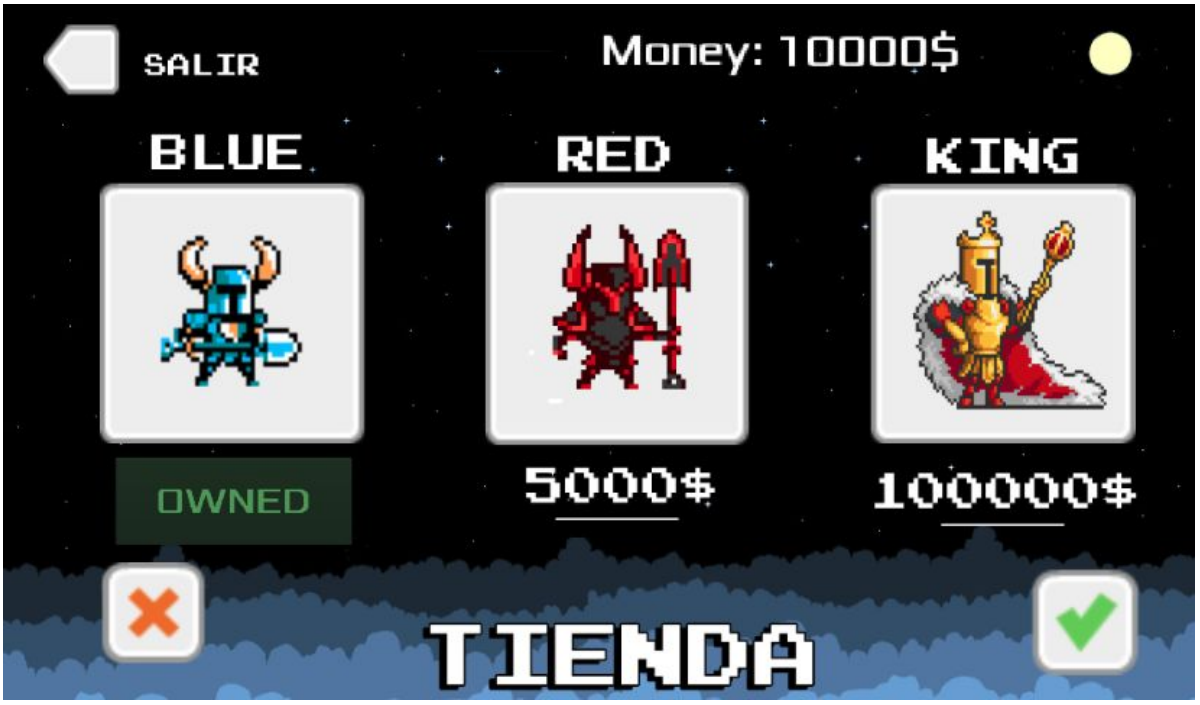
- Dificultad: Podemos subir y bajar la dificultad del juego pulsando el botón.

- Idioma: Podemos cambiar el idioma del juego pulsando el botón.

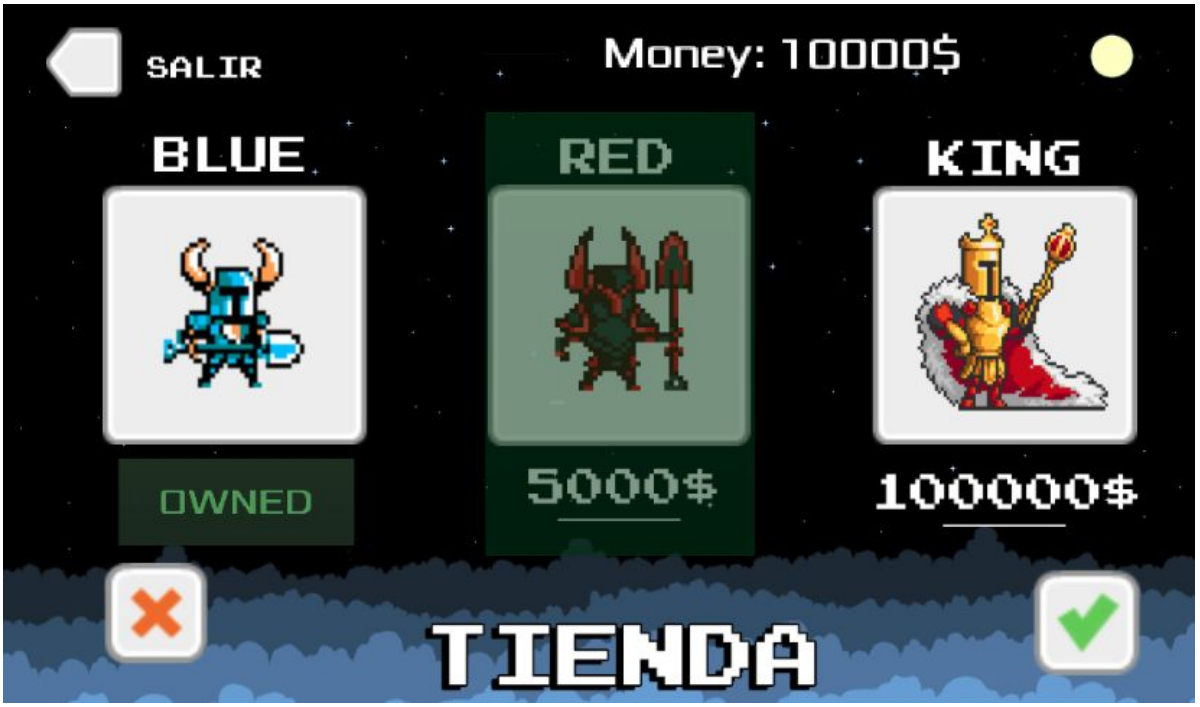
Tienda:

En esta opción podremos comprar nuevos personajes con el dinero que hayamos conseguido durante las partidas.

Cuando entremos por primera vez lo tendremos así:

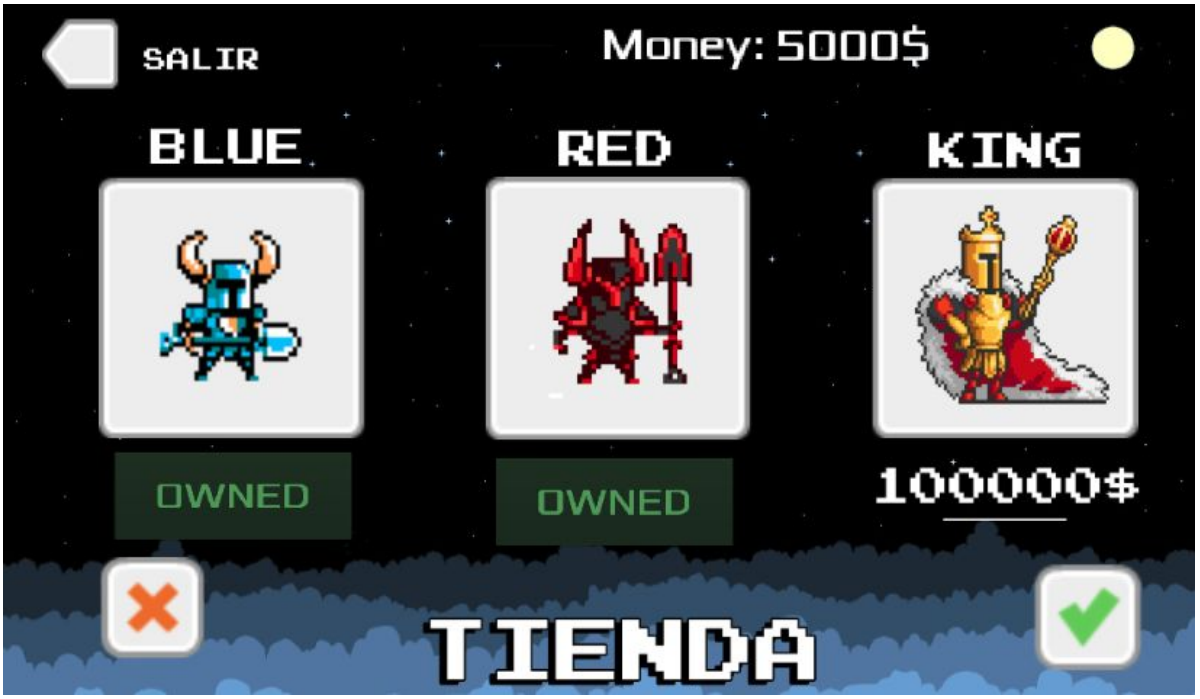


Ahora seleccionamos un personaje para comprar, en este caso será el rojo.



Y pulsamos el botón verde para confirmar la compra.

Y el personaje nos aparece como comprado, además que se ha restado el dinero que teníamos.


SALIR
Money: 5000$
BLUE
RED
KING
OWNED
OWNED
100000$
TIENDA