Issac Alexander Guerrero Prado

Proyecto Final EDA

Código Fuente.

Jugador:

```
3
4  var velocidad = Vector2()
5  export (float) var GRAVEDAD = 100
6  export (float) var velocidadDeMovimiento = 4000
7  export (float) var vel_salto = 300
8  var puedeSaltar = false
9  var puedeDisparar = true
10 enum estados {atacar, idle, corriendo, muerto}
11 var estadoActual = estados.idle
12 export(PackedScene) var fuego
13 export(Vector2) var spawn_der
14 export var victoria = false
15
16 func _ready(): #Funcion contenida en Godot que corre el código dentro de la funcion al iniciarse el juego.
17     spawn_der = get_node("cuerpoJ1/spawnbala").position #inicia el punto de spawn de la bala que lanza el jugador
18     get_node("cuerpoJ1/CamaraJ1").set_limit(MARGIN_LEFT, get_tree().get_nodes_in_group("minimo")[0].global_position.x) #se establecen los limites de la camara con 2 puntos de referencia.
19     get_node("cuerpoJ1/CamaraJ1").set_limit(MARGIN_TOP, get_tree().get_nodes_in_group("minimo")[0].global_position.y)
20     get_node("cuerpoJ1/CamaraJ1").set_limit(MARGIN_RIGHT, get_tree().get_nodes_in_group("maximo")[0].global_position.x)
21     get_node("cuerpoJ1/CamaraJ1").set_limit(MARGIN_BOTTOM, get_tree().get_nodes_in_group("maximo")[0].global_position.y)
22
23 func _physics_process(delta): #Funcion de tiempo
24
25     velocidad.y += GRAVEDAD * delta
26
27     if(estadoActual != estados.muerto): #condicional de ejecutar las acciones del personaje solo cuando este vivo.
28
29         if(Input.is_action_pressed("tecla_d") && estadoActual != estados.atacar): #este if movera a nuestro personaje y ejecutara la animacion en la direccion original del sprite.
30             velocidad.x = velocidadDeMovimiento
31             get_node("cuerpoJ1/spawnbala").position = spawn_der
32             get_node("cuerpoJ1/Sprite").flip_h = false
33             estadoActual = estados.corriendo
34             if(Input.is_action_just_pressed("tecla_z") && (puedeSaltar)): #cuando se apriete la tecla z y la condicion puedeSaltar sea verdadera:
35                 get_node("cuerpoJ1/animacionJ1").play("correr") #se ejecutara la animacion de correr y aumentara la velocidad del personaje
36             if(Input.is_action_pressed("tecla_z") && (puedeSaltar)):
37                 velocidad.x = 23000
38                 get_node("cuerpoJ1/animacionJ1").play("correr")
39             elif(Input.is_action_just_released("tecla_z")): #cuando se deje deje de apretar la tecla z el personaje detendra su animacion y cambiara su estado a reposo.
40                 get_node("cuerpoJ1/animacionJ1").stop()
41                 velocidad.x = velocidadDeMovimiento
42                 estadoActual = estados.idle
43             elif(!get_node("cuerpoJ1/animacionJ1").is_playing() && (puedeSaltar)): #si no se esta ejecutando alguna animacion y el personaje puede saltar el jugador caminara.
44                 get_node("cuerpoJ1/animacionJ1").play("caminar")
45
46         elif(Input.is_action_pressed("tecla_i") && estadoActual != estados.atacar): #este segmento del codigo hace lo mismo que el anterior pero en espejo, invirtiendo el sprite y las animaciones
47             velocidad.x = -velocidadDeMovimiento
48             get_node("cuerpoJ1/spawnbala").position = Vector2(spawn_der.x * -1, spawn_der.y)
49             get_node("cuerpoJ1/Sprite").flip_h = true
50             estadoActual = estados.corriendo
51             if(Input.is_action_just_pressed("tecla_z") && (puedeSaltar)):
52                 get_node("cuerpoJ1/animacionJ1").play("correr")
53             if(Input.is_action_pressed("tecla_z") && (puedeSaltar)):
54                 velocidad.x = -23000
55                 get_node("cuerpoJ1/animacionJ1").play("correr")
56             elif(Input.is_action_just_released("tecla_z")):
57                 get_node("cuerpoJ1/animacionJ1").stop()
58                 estadoActual = estados.idle
59                 velocidad.x = -velocidadDeMovimiento
60             elif(!get_node("cuerpoJ1/animacionJ1").is_playing() && (puedeSaltar)):
61                 get_node("cuerpoJ1/animacionJ1").play("caminar")
62         else:
63             velocidad.x = 0 #si el jugador no se esta moviendo entonces se ejecutara el estado de reposo
64             if(puedeSaltar && estadoActual != estados.atacar):
65                 get_node("cuerpoJ1/animacionJ1").play("Idle")
66         if(Input.is_action_pressed("tecla_x") && puedeSaltar): #si se aprieta la tecla x y se puede saltar entonces habrá una velocidad en el eje y y se ejecutara la animacion del salto
67             velocidad.y = -vel_salto
68             get_node("cuerpoJ1/animacionJ1").play("salto")
69             puedeSaltar = false #el estado de salto se vuelve falso para que no pueda volver a saltar mientras realiza la accion
```

```
		if(Input.is_action_pressed("tecla_c") && (puedeSaltar) && puedeDisparar): #si podemos saltar y disparar y apretamos la tecla c en el punto de spawn de la bala aparecera una bala
			estadoActual = estados.atacar
			ataque()
			var newDisparo = fuego.instance()
			get_node("cuerpoJ1/animacionJ1").play("ataque1") #se ejecutara la animacion del ataque
			newDisparo.global_position = get_node("cuerpoJ1/spawnbala").global_position
			get_tree().get_nodes_in_group("main")[0].add_child(newDisparo)
			puedeDisparar = false #cambiamos el estado de disparo para que se pueda tener una taza de disparo controlada
			get_node("cuerpoJ1/timer_cd").start() #cuando se ejecute esta accion con ayuda de una funcion de godot se cuentan segundos y cuando acabe el conteo se podra volver a disparar
			if(get_node("cuerpoJ1/Sprite").flip_h): #aqui se invierte el sentido de la bala dependiendo de la direccion donde este mirando el jugador
				newDisparo.velocidad.x = -newDisparo.potencia

			else:
				newDisparo.velocidad.x = newDisparo.potencia


		if(get_node("cuerpoJ1").get_slide_collision(get_node("cuerpoJ1").get_slide_count()-1) != null): #si nuestro jugador colisiona con algo, como va a reaccionar
			var objetoColisionado = get_node("cuerpoJ1").get_slide_collision(get_node("cuerpoJ1").get_slide_count()-1).collider
			if(objetoColisionado.is_in_group("suelo")):#si es el suelo, podremos volver a saltar si nuestro estado de puedeSaltar es falso
				get_tree().get_nodes_in_group("spawnj1")[0].global_position = get_node("cuerpoJ1").global_position #en caso de que muera se spawneara al jugador en el ultimo segmento de suelo colisiona
				if(puedeSaltar == false):
					puedeSaltar = true
					get_node("cuerpoJ1/animacionJ1").stop()
			elif(objetoColisionado.is_in_group("enemigo")): #si se colisiona con un enemigo entonces se ejecutara la funcion de muerte del jugador
				muerte_jugador()
		elif(puedeSaltar): #si no se colisiona con nada no se puede saltar.
			puedeSaltar = false

	func _on_timer_cd_timeout(): #esta funcion maneja el tiempo que toma volver a activar la funcion puedeDisparar
		puedeDisparar = true
	func ataque(): #esta funcion se asegura de que termine la funcion de atacar antes de ejecutarse otra animacion
		get_node("cuerpoJ1/animacionJ1").play("ataque1")
		yield(get_node("cuerpoJ1/animacionJ1"), "animation_finished")
		estadoActual = estados.idle

	func muerte_jugador():
		velocidad.x = 0
		var enemigos = get_tree().get_nodes_in_group("enemigo") #cuando mueres no hay colision con enemigos y tu estado pasa a muerto
		for enemigo in enemigos:
			get_node("cuerpoJ1").add_collision_exception_with(enemigo)
		estadoActual = estados.muerto
		get_node("cuerpoJ1/animacionJ1").play("muerte") #se ejecuta la animacion de muerte
		yield(get_node("cuerpoJ1/animacionJ1"), "animation_finished")
		get_tree().get_nodes_in_group("main")[0].respawn_j1() #del script del main se ejecuta la funcion de respawn y tu conteo de vidas baja en 1
		get_tree().get_nodes_in_group("main")[0].jugadorVida -= 1
		get_tree().get_nodes_in_group("camara")[0].global_position = get_node("cuerpoJ1/CamaraJ1").global_position #el control de la camara pasa a una camara secundaria.
		queue_free() #libera el nodo del jugador quitandolo del juego


	func _on_VisibilityNotifier2D_screen_exited(): #funcion de godot que mide la visibilidad del jugador
		print("Salio de pantalla") #cuando el jugador salga de la pantalla morira automaticamente.
		muerte_jugador()

```

Nivel:

```
#Nivel
extends Node

export (PackedScene) var Jugador_1
export (PackedScene) var Vida
export (PackedScene) var gameover
export (PackedScene) var reiniciar
export (PackedScene) var juego

export var jugadorVida = 3
var vidas = []

func _ready():
	spawn_j1()
	get_tree().get_nodes_in_group("camara")[0].set_limit(MARGIN_LEFT, get_tree().get_nodes_in_group("minimo")[0].global_position.x) #se establecen los limites del nivel
	get_tree().get_nodes_in_group("camara")[0].set_limit(MARGIN_TOP, get_tree().get_nodes_in_group("minimo")[0].global_position.y)
	get_tree().get_nodes_in_group("camara")[0].set_limit(MARGIN_RIGHT, get_tree().get_nodes_in_group("maximo")[0].global_position.x)
	get_tree().get_nodes_in_group("camara")[0].set_limit(MARGIN_BOTTOM, get_tree().get_nodes_in_group("maximo")[0].global_position.y)
	crearVidas()

func respawn_j1():
	get_tree().get_nodes_in_group("camara")[0].make_current() #aqui se establece la segunda camara cuando el jugador muere
	if(jugadorVida > 0): #si tu conteo de vidas es mayor que 0 se contaran 2 segundos y despues se volvera a spawnear al jugador
		yield(get_tree().create_timer(2,0), "timeout")#esta funcion es la que cuenta 2 segundos
		get_tree().get_nodes_in_group("camara")[0].clear_current() #cuando spawnea el jugador la camara principal vuelve a tomar el control
		spawn_j1()
		actualizarVida()
	else: #cuando nuestras vidas llegan a 0 se cuentan 5 segundos y el juego vuelve a empezar
		gameover()
		yield(get_tree().create_timer(5,0), "timeout")
		get_tree().reload_current_scene()

func spawn_j1(): #se guarda el nodo del jugador en una variable y se spawnea en el nivel
	var j1 = Jugador_1.instance()
	get_tree().get_nodes_in_group("nivel")[0].add_child(j1)
	j1.global_position = get_tree().get_nodes_in_group("spawnj1")[0].global_position

func crearVidas(): #aqui se toma la escena del sprite de vidas y se crean en pantalla dependiendo del numero de vidas que se tengan
	for i in jugadorVida:
		var newVida = Vida.instance()
		vidas.append(newVida)
		get_tree().get_nodes_in_group("GUI")[0].add_child(newVida)
		newVida.global_position.x += i * 60

func actualizarVida(): #aqui se van restando los sprites de vidas conforme se va muriendo.
	vidas[jugadorVida].queue_free()

func gameover(): #cuando se pierde ya no aparece el jugador y aparece un mensaje de gameover
	var fin = gameover.instance()
	var reinicio = reiniciar.instance()
	get_tree().get_nodes_in_group("GUI")[0].add_child(fin)
```

Enemigo:

```gdscript
#Personaje Enemigo
extends KinematicBody2D

export (float) var GRAVEDAD = 100
var velocidad = Vector2()
export (float) var velocidadDeMovimiento = 4000
enum estados {corriendo, saltando, cayendo}
var estado_actual = estados.corriendo
var habilitado = false

func _ready():
	get_node("Sprite").flip_h = true #invertimos el sprite y hacemos que camine en direccion al jugador
	get_node("enemigo_animacion").play("enemigo_caminar")

func _physics_process(delta):

	if(habilitado):

		velocidad.y += GRAVEDAD * delta
		velocidad.x = -velocidadDeMovimiento

		var movimiento = velocidad * delta

		move_and_slide(movimiento)


		if(get_slide_collision(get_slide_count()-1) != null):
			var objetoColisionado = get_slide_collision(get_slide_count()-1).collider
			if(objetoColisionado.is_in_group("suelo") && estado_actual != estados.corriendo): #si el enemigo colisiona con una bala muere, si colisiona con un jugador el jugador muere.
				estado_actual = estados.corriendo
			elif(objetoColisionado.is_in_group("bala")):
				muerte_enemigo()
				objetoColisionado.queue_free()
			elif(objetoColisionado.is_in_group("jugador")):
				objetoColisionado.get_parent().muerte_jugador()
func muerte_enemigo(): #cuando el enemigo colisiona con una bala del jugador se ejecutara la animacion de muerte enemiga y despues se elimina el nodo
	get_node("enemigo_animacion").play("explosion")
	yield(get_node("enemigo_animacion"), "animation_finished")
	queue_free()

func _on_VisibilityNotifier2D_screen_entered(): #si el personaje no esta en pantalla no actuara ni se movera.
	habilitado = true


func _on_VisibilityNotifier2D_screen_exited(): #cuando el enemigo sale de la pantalla se elimina
	queue_free()

```

Disparo del jugador:

```gdscript
#Disparo del jugador
extends KinematicBody2D

var velocidad = Vector2()
export var potencia = 0

func _ready():
    velocidad.x = 0
    velocidad.y = 0
func _physics_process(delta):
    var movimiento = velocidad * delta #aqui se asigna la velocidad de la bala
    move_and_collide(movimiento)
    var suelos = get_tree().get_nodes_in_group("suelo") #nuestra bala no colisionara con objetos en el grupo suelo
    for suelo in suelos:
        add_collision_exception_with(suelo)

func _on_VisibilityNotifier2D_screen_exited(): #cuando la bala salga de la pantalla se eliminara
    queue_free()

```

Disparo del enemigo:

```gdscript
#Bala de cañon
extends KinematicBody2D

var velocidad = Vector2()
export var potencia = 0

func _ready():
	velocidad.x = 0
	velocidad.y = 0
func _physics_process(delta):
	var movimiento = -velocidad * delta #la bala avanzara hacia el jugador
	var todos_suelos = get_tree().get_nodes_in_group("suelo")
	for suelo in todos_suelos:
		add_collision_exception_with(suelo)#la bala no colisionara con objetos de colision tipo suelo
	var enemigos = get_tree().get_nodes_in_group("enemigo")#la bala no colisionara con enemigos
	for enemigo in enemigos:
		add_collision_exception_with(enemigo)

	var balas = get_tree().get_nodes_in_group("bala") #las balas tanto aliadas como enemigas no pueden colisionar entre si
	for bala in balas:
		add_collision_exception_with(bala)
	move_and_slide(movimiento)

	if(get_slide_collision(get_slide_count()-1) != null):
		var objetoColisionado = get_slide_collision(get_slide_count()-1).collider
		if(objetoColisionado.is_in_group("jugador")):
			objetoColisionado.get_parent().muerte_jugador() #si la bala colisiona con un jugador el jugador muere
			queue_free()

func _on_VisibilityNotifier2D_screen_exited(): #cuando la bala salga del campo de vision se eliminara
	queue_free()

```

Cañón:

```
#Cañon
extends KinematicBody2D
var habilitado = false
var puede_disparar = true
export (PackedScene) var balaEnemiga


func _ready():
    pass

func _physics_process(delta):
    if (habilitado):
        var enemigos = get_tree().get_nodes_in_group("enemigo")
        for enemigo in enemigos:
            add_collision_exception_with(enemigo) #la bala no colisionara con enemigos.
        if (puede_disparar):
            disparar()
            get_node("cd_bala").start() #cuando se dispare el cañon empieza el conteo para volver a habilitar el disparo.


func _on_VisibilityNotifier2D_screen_entered():
    habilitado = true
func disparar(): #aqui se establece donde apareceran las balas y se cambiara a falso el estado de disparo para controlar la cantidad de disparos
    puede_disparar = false
    var newBala = balaEnemiga.instance()
    get_tree().get_nodes_in_group("main")[0].add_child(newBala)
    newBala.global_position = get_node("spawnbala").global_position
    newBala.velocidad.x = newBala.potencia


func _on_cd_flecha_timeout(): #esta funcion maneja la cantidad de balas por segundo
    puede_disparar = true


func _on_VisibilityNotifier2D_screen_exited(): #el cañon disparara solo cuando este en campo de vision
    habilitado = false

```

Prototipo de Videojuego (Mighty Knight)

Objetivo: Aprender a usar los motores de videojuegos (en este caso Godot Engine) así como aplicar las estructuras de datos en los avances del proyecto de diversas formas. Y como resultado final poder proporcionar un juego que sea capaz de entretener a la audiencia y que tenga un nivel moderado de complejidad.

Alcance del proyecto: Un solo nivel que sea capaz de tener muchas interacciones, que muestre la jugabilidad y estética de este y que a su vez ocupen varias estructuras de datos.

Introducción

Godot Engine es un motor de videojuegos libre, lo que lo vuelve una herramienta accesible para todos y lo vuelve ideal para comenzar a programar videojuegos.

Este motor fue desarrollado principalmente por compañías latinoamericanas, sin embargo, al rivalizar con grandes motores, como lo es Unreal Engine y Unity, Godot no ha tenido una gran difusión así que es desconocido para muchos. Aun así, es un motor con grandes capacidades y es importante poder mostrarlas al mundo de los desarrolladores.

Por esto mi proyecto tiene la importancia de poner a prueba las capacidades de este motor y de ser posible posteriormente comparar el resultado contra cualquiera de los otros motores mencionados.

Desarrollo: El lenguaje de programación utilizado es Godot Script que es muy familiar a Python, es el lenguaje predeterminado del motor y lo utilice ya que complementa lo que he aprendido de Python durante las prácticas, aunque con las actualizaciones que ha ido recibiendo ahora también es capaz de aceptar otros 3 lenguajes: C++, C# y Visual Script

Las estructuras de datos están presentes prácticamente en todo el código, todo esta organizado por nodos y se van acomodando dentro de una lista principal, a su vez estos nodos pueden contener otras listas, por ejemplo, dentro de el nodo principal tenemos un nodo de jugador, este a su vez es una lista que contiene todas las características del personaje y dentro de las características se tiene un nodo de animaciones que contiene todas las animaciones que se van llamando conforme lo vaya solicitando el código.

Y si el juego tuviera mas niveles se acomodarían las escenas en forma de una cola, la primera escena que se reproduce es la primera que va a salir para dar paso a la siguiente etc.

Algoritmo:

Datos de entrada:

Teclas: izquierda, derecha, z, c y x

Salida:

Si se aprieta la tecla izquierda, el personaje debe moverse hacia la izquierda y reproducir la animación de movimiento.

Si se aprieta la tecla derecha, el personaje debe moverse hacia la derecha y reproducir la animación de movimiento.

Si se aprieta la tecla z el personaje debe aumentar la velocidad en la dirección donde este corriendo y ejecutar la animación de correr mientras el personaje no esté atacando

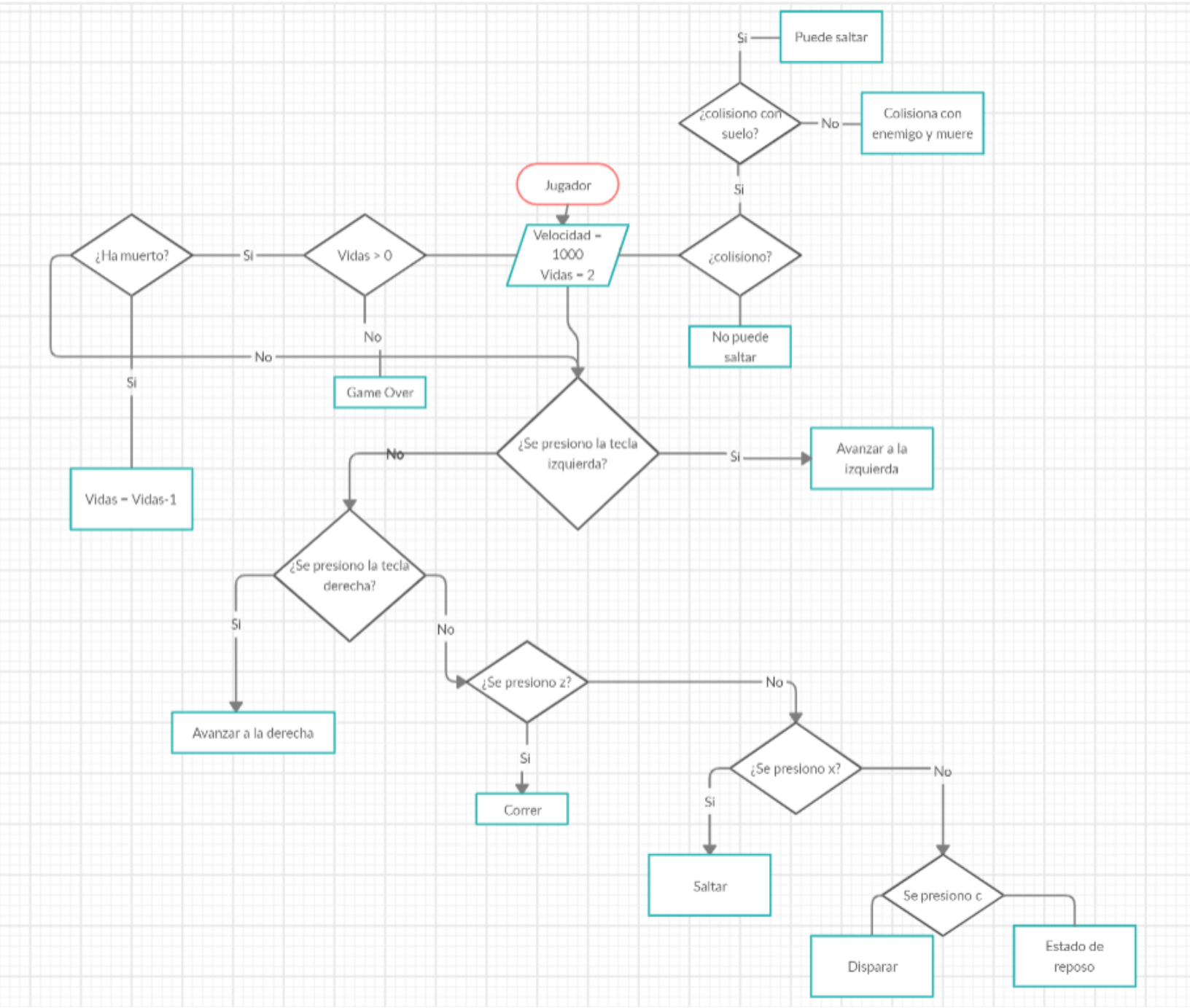
Si se aprieta la tecla x el personaje deberá adquirir una velocidad en el eje y y reproducir la velocidad de brinco mientras el personaje no este brincando en ese momento.

Si se aprieta la tecla z el personaje deberá ejecutar la animación de ataque y disparar un proyectil mientras el personaje no esté brincando.

Si el personaje colisiona con una bala enemiga o un enemigo el jugador perderá una vida.

Si el personaje se queda sin vidas se termina el juego.

Diagrama de flujo:



Pseudocódigo:

INICIO (Jugador)

Velocidad, Vidas, PosicionX, PosicionY: ENTERO

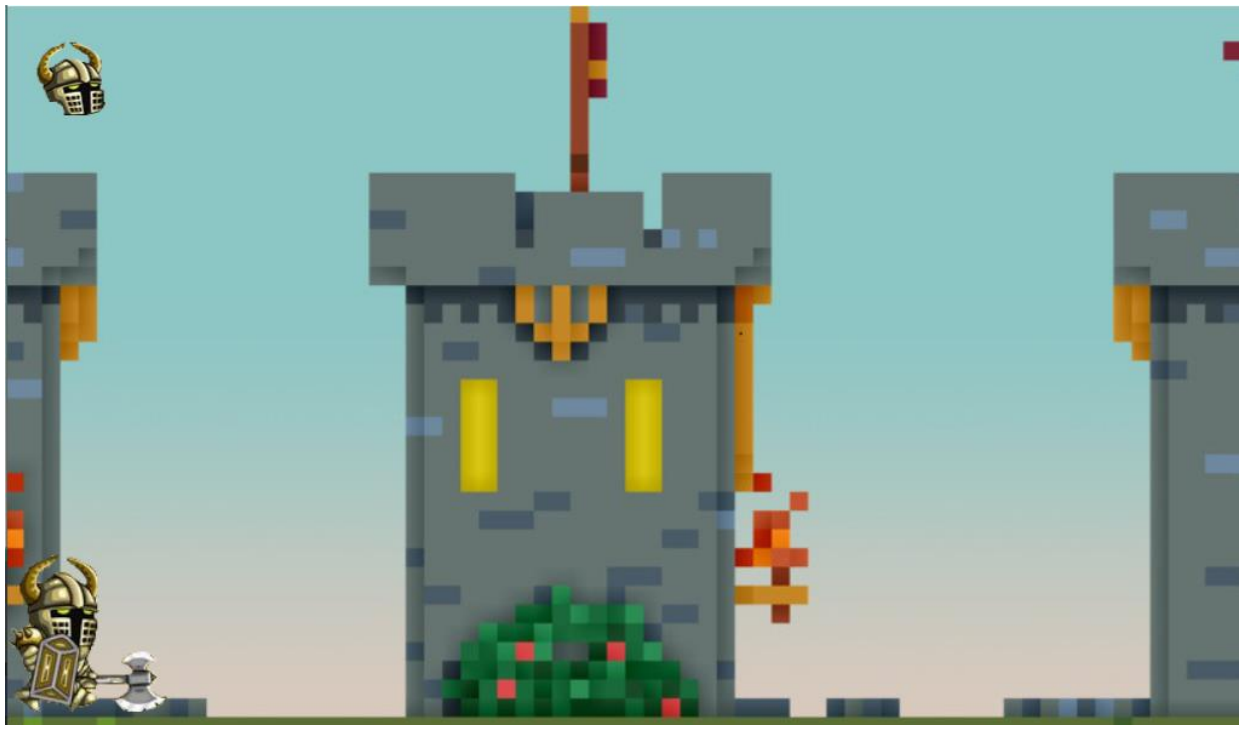
PuedeSaltar, Puede Disparar, Vida: BOOL

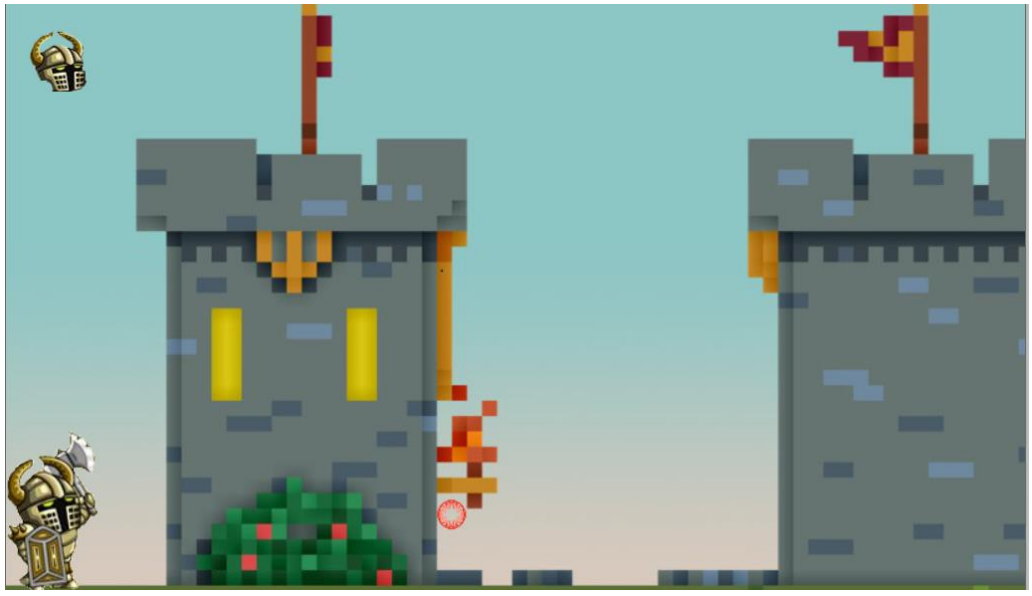
PuedeSaltar = FALSE

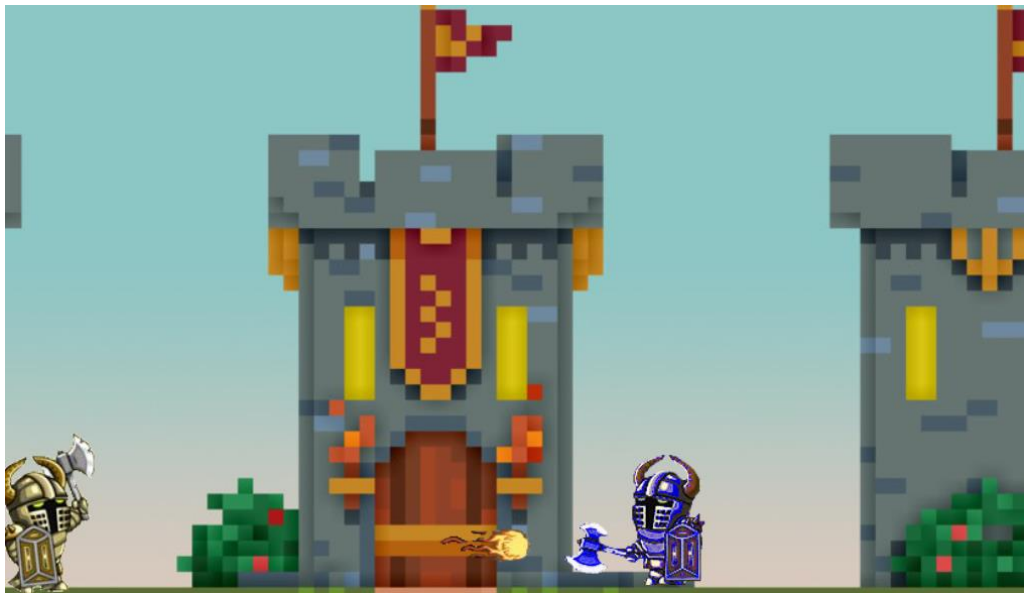
PuedeDisparar = TRUE

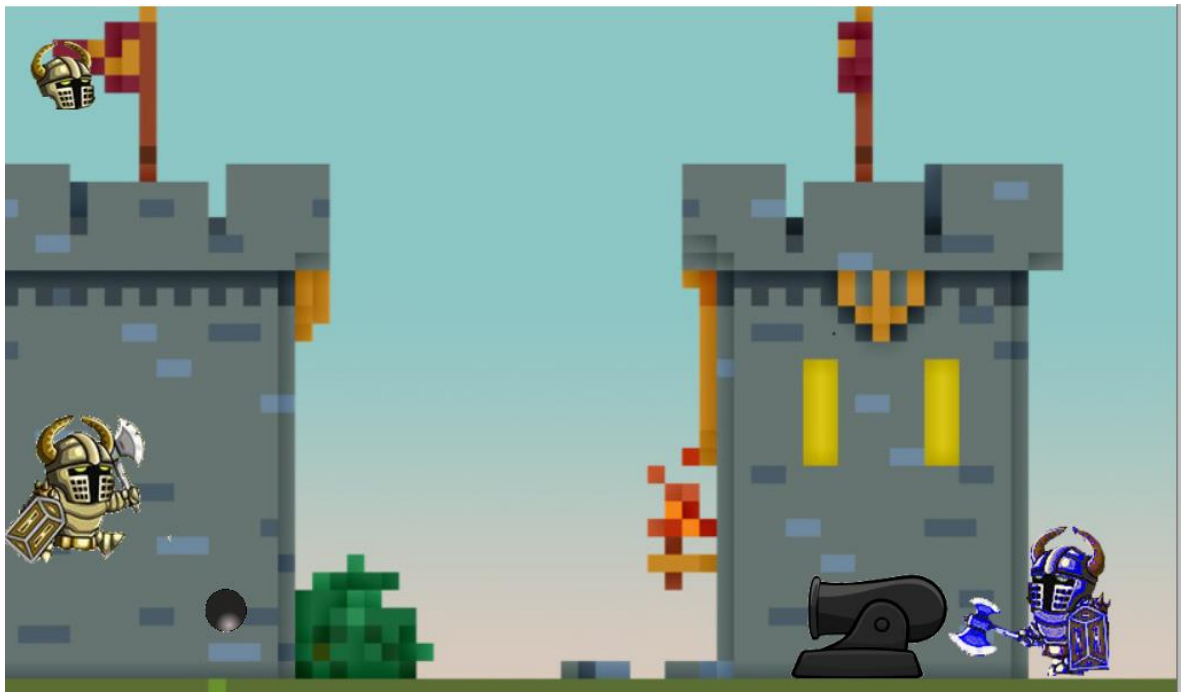
```
Vida = True
Velocidad := 1000
Vidas := 2
PosicionX, PosicionY := 0

HACER
    SI Se presiona Tecla derecha ENTONCES
        PosicionX = PosicionX+1
    SI Se presiona Tecla izquierda ENTONCES
        PsicionX = PosicionX-1
    SI Se presiona Tecla Z ENTONCES
        SI Se presiona Tecla derecha ENTONCES
            PosicionX = PosicionX+10
        SI Se presiona Tecla izquierda ENTONCES
            PosicionX = PosicionX-10
    SI se presiona Tecla X ENTONCES
        PosicionY = PosicionY+5
    Si se presiona Tecla C ENTONCES
        Disparar()
    Si Jugador Colisiona ENTONCES:
        SI colisiona con suelo ENTONCES
            PuedeSaltar= TRUE
        SI colisiona con enemigo ENTONCES
            Muertejugador()

MIENTRAS Vida = TRUE
SI Vida = False ENTONCES
    GameOver()
    Restart()
```

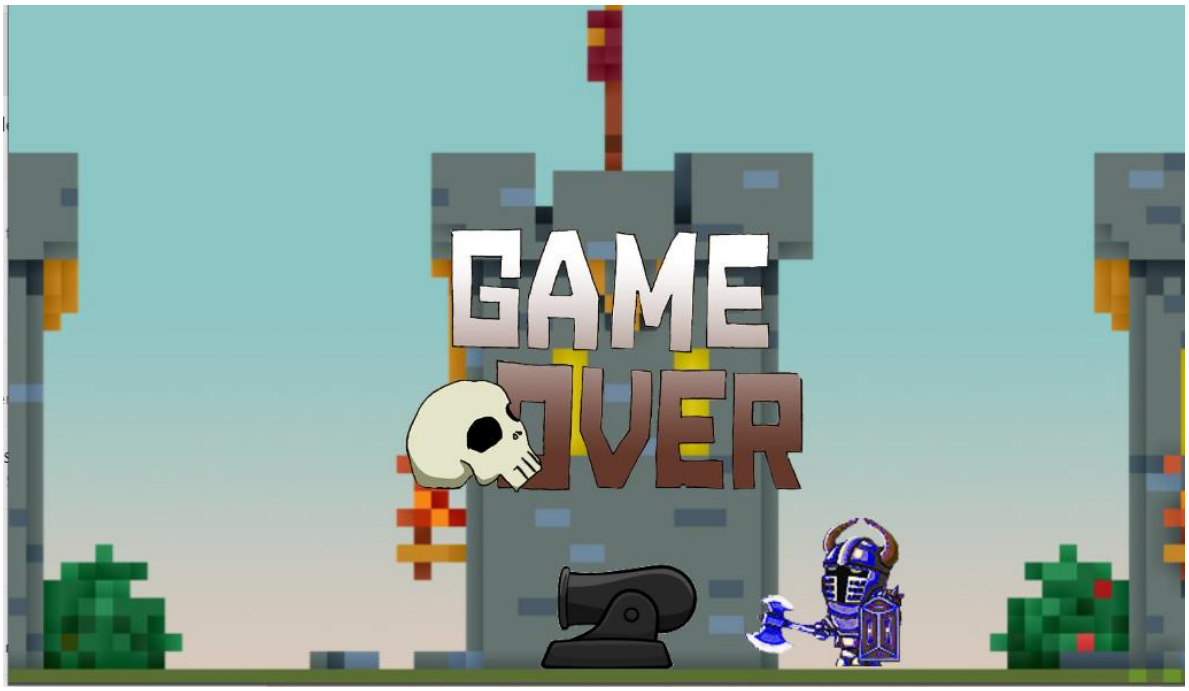
Resultados:











Conclusiones: Godot Engine es un motor de videojuegos con muchas capacidades, es cierto que todavía le faltan actualizaciones para ser tan bueno como Unreal o Unity pero teniendo en cuenta el estado inicial de este motor a como esta en este momento ha dado un gran avance y quizás en algún momento pueda ser utilizado en algún juego importante, mientras es una buena herramienta para aprender a programar videojuegos y los resultados son satisfactorios además de soporta varios lenguajes de programación lo que le da una ventaja para poder empezar sabiendo cualquiera de estos.

Yo estoy satisfecho con el resultado final de mi proyecto, pero seguiré avanzando hasta poder aprovechar al máximo las capacidades del motor y demostrar lo que se puede lograr en él.

Referencias:

https://docs.godotengine.org/en/stable/about/introduction.html

https://docs.godotengine.org/en/stable/tutorials/2d/index.html