UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FÍSICA PARA PROGRAMADORES DE VIDEOJUEGOS:
Proyecto Final (Donkey Kong: Salvando a la princesa)

Dra. García Canseco Eloísa del Carme

Aguilar Camacho Jesús Darío
351488
Ishihara Moros Ariana
354214

Ensenada, Baja California a 28 de mayo del 2021

Índice.

Objetivo:

- Implementar propiedades físicas al juego de Donkey Kong: Salvando a la princesa, de manera que se pueda percibir una sensación de realismo y lógica en los movimientos del jugador (Mario) y los elementos que componen el juego (Barriles, martillos, escenario).

1. Introducción

Para este proyecto final, decidimos realizar un videojuego basado en Donkey Kong del año 1981 desarrollado por Nintendo. La idea principal del videojuego se mantiene, utilizando los mismos *assets* y colores característicos del escenario principal.

Decidimos irnos por este videojuego ya que el recordar viejos tiempos es algo bonito, o simplemente recordar cuando lo jugamos de niños con esos sonidos de 8 bits tan reconocidos, pero quisimos darle un toque diferente con algunos cambios. A Mario le dimos la propiedad que lanzará martillos, para lo que elegimos un tiro parabólico que nos pareció lo más adecuado para justificar el alcance y contrastar con la dinámica del juego. Siendo las fórmulas de posición implementadas las siguientes:

$$p_x(t) = v \cdot cos(\theta) \cdot t$$

(1)

$$p_y(t) = v \cdot sin(\theta) \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

(2)

Donde *v* es la velocidad inicial, *g* es la gravedad, *t* es el tiempo, y θ es el ángulo de tiro.

2. Descripción del videojuego

El videojuego que desarrollamos consta de un personaje principal llamado Mario, en donde el villano Donkey Kong tratará de derrotar a Mario y así este no pueda salvar a la princesa, que estará en la espera de ser rescatada.
Mario tendrá la capacidad de lanzar martillos para destruir los barriles que Donkey Kong le estará lanzando. También podrá brincar los barriles que rodaran a lo largo de las plataformas y bajarán de plataforma cuando encuentren una escalera.
Una vez Mario llegue con la princesa se reinicia el juego y se aumenta el puntaje a un juego ganado, si Mario es alcanzado por algún barril pierde una vida y regresa al punto de partida, si pierde las tres vidas regresa a la pantalla de inicio y el puntaje de juegos ganados se reinicia a cero.

Imagen 1

a. PANTALLA DE INICIO

Una vez que iniciamos el juego se nos mostrará la pantalla de inicio (Imagen 1), esta será la pantalla donde muestra el nombre del juego y el botón para iniciar la partida.

b. BARRIL

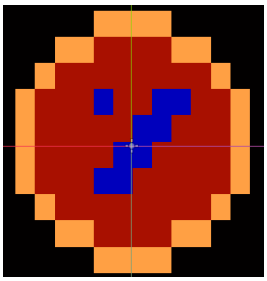


Imagen 2

Tiene la capacidad de rodar por las plataformas (Imagen 2) y si se encuentra con una escalera baja a la siguiente plataforma (Imagen 3), cuando cae o choca este rebota. Hay fuerzas aleatorias que actúan sobre ellos para hacer más dinámico el juego.

Imagen 3

c. DONKEY KONG

Nuestro villano, que intentará a toda costa que Mario llegue con la princesa Peach para que la rescate, estará lanzando barriles constantemente (Imagen 4 y 5) para poder acabar con Mario. Lanzará un total de 25 barriles, y cuando termine de lanzarlos empezará darse golpes en el pecho (Imagen 6) esperando a que los barriles hagan su trabajo.

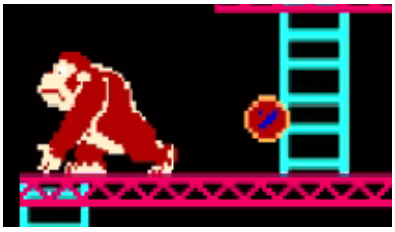


Imagen 4

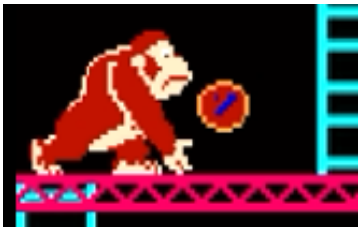


Imagen 5

Imagen 6

d. PRINCESA

La princesa estará esperando impacientemente con ansias, mientras se mueve de izquierda a derecha en la misma posición, en la cima de las plataformas (Imagen 7) a ser rescatada por Mario. Cuando llegue Mario aquí (Imagen 8), se adjudica una victoria en el contador de juegos ganados (Imagen 9) y regresa a la *pantalla de inicio*.

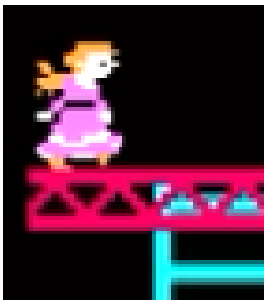


Imagen 7

Imagen 8

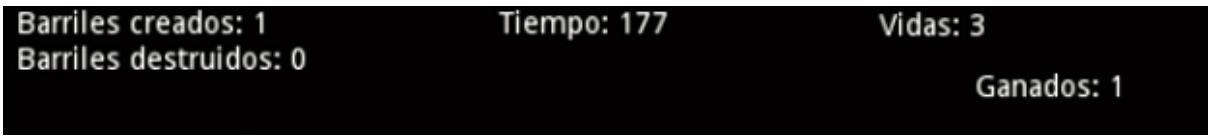


Imagen 9

e. MARIO

Nuestro personaje principal desesperado por salvar a su princesa, el amor de su vida, se moverá de izquierda a derecha por el nivel (Imagen 10 y 11). Buscará a toda costa salvarla subiendo plataformas haciendo uso de escaleras (Imagen 12), además de ir destruyendo barriles (Imagen 13 y 14), lanzados por el villano y malvado Donkey Kong, con martillos infinitos que lanzará a diestra y siniestra. También podrá esquivar los barriles brincándolos (Imagen 15) y evitar ser golpeado por estos, ya que si lo golpean (Imagen 16) perderá una vida y será posicionado en el principio del nivel. Si Mario es derrotado rotundamente por los barriles y pierde sus tres vidas, regresará a la *pantalla de inicio* (Imagen 17), y el contador de victorias arrasadoras será reiniciado.

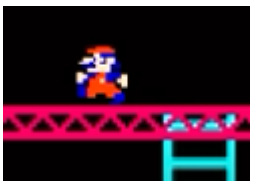


Imagen 10

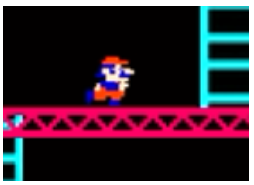


Imagen 11

Imagen 12

Imagen 13

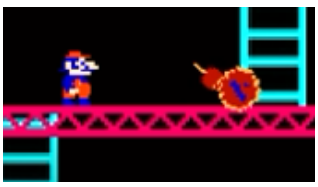


Imagen 14

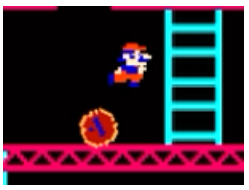


Imagen 15

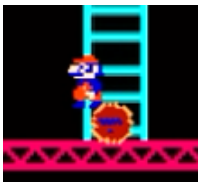


Imagen 16

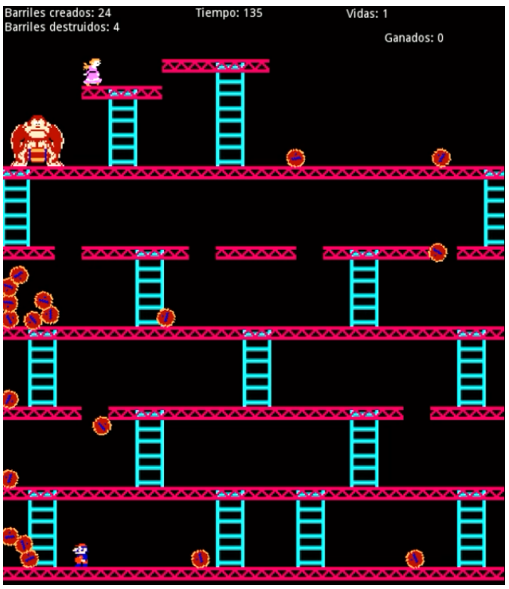
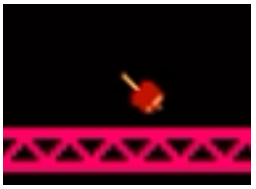


Imagen 17

f. *MARTILLOS*

Estos serán los proyectiles de nuestro personaje principal, Mario, que serán lanzados mientras da un salto. Al chocar con el piso de las plataformas estos desaparecen drásticamente, mientras no desaparezcan Mario no podrá lanzar otro martillo. Tienen la capacidad de acabar con los *Barriles* lanzados por el Malvado Donkey Kong.



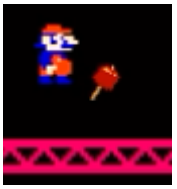


Imagen 18 Imagen 19

g. SCORE

Aquí se mostrará el progreso del juego. La cantidad de barriles que el villano está lanzando hasta que llegue a su límite de 25. Los barriles que nuestro héroe principal ha destruido con sus martillos infinitos. El tiempo que consta de 180 segundos, si llega a cero Mario regresa a la *pantalla de inicio*. Aparece el contador de la vida que tiene Mario y su marcador de juegos ganados.

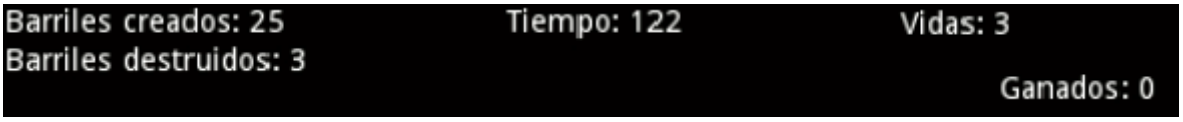


Imagen 20

3. Desarrollo del proyecto

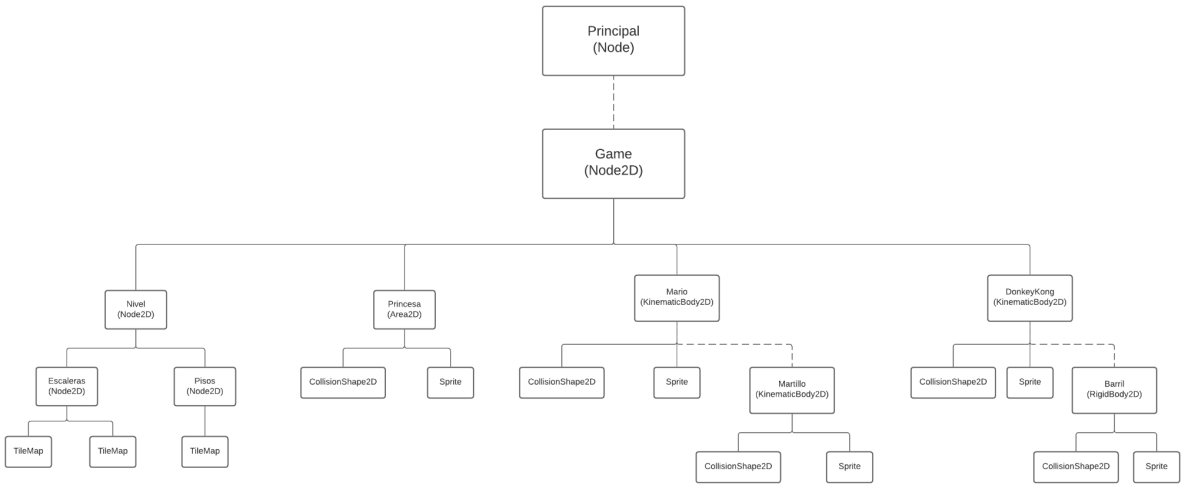


Imagen 21

Para comenzar el desarrollo del proyecto nos fuimos enfocando primero en los nodos de menor jerarquía dentro de nuestro diagrama para realizar la gran escena.

Para comenzar tenemos un script *variables_globales.gd* en el que se almacenan todas las variables que necesitan y modifican uno o más nodos a lo largo del juego, por lo que este documento no está afiliado a algún nodo en particular ya que se compila automáticamente durante la ejecución del programa.

```
extends Node

var b
var p = Vector2(0, 0)
var barriles #barriles creados
var b_dest #barriles destruidos
var vida_mario #Contabilidad de las vidas de mario
var vidas #Marcadores que nos dirá si Mario murió
var tiempo_juego = 180 #Tiempo que tiene mario para ganar la partida
var gano = 0 #Contador de juegos ganados en una sesión
var g_b = false #Marcador para avisarnos si Mario ganó una partida

# Called when the node enters the scene tree for the first time.
func _ready():
    b = 0
    barriles = 0
    b_dest = -1
    vida_mario = true
    vidas = 3
```

Imagen 22

Imagen: variables_globales.gd

a. MARTILLO

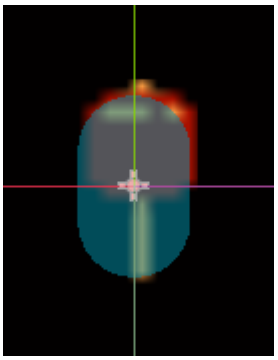


Imagen 23

Martillo es un nodo KinematicBody2D, lo elegimos hacer de este tipo para tener control total de cada movimiento y reacción del nodo, ya que afecta directamente a los nodos de Barril y Mario. En este elemento del juego aplicamos las fórmulas *(1)* y *(2)* presentes en la Introducción del documento. Estas serán las que calcularán la posición al instante de nuestro martillo con valores iniciales predeterminados y otros dados por la posición de *Mario*.

A este nodo se encuentra afiliado el script Martillo.gd, donde se comienza inicializando las siguientes variables:

```
extends KinematicBody2D

#	Se inicializan las variables para la trayectoria
#	del martillo
var velocidad = 5 #Velocidad inicial
var pos = Vector2() #Posición del Martillo
var ang = 60 #Ángulo inicial de Tiro
var g = 9.81 #Gravedad
var t = 0 #Tiempo actualizado por delta
const up_direction = Vector2(0, -1)
```

Imagen 24

Está compuesto también por las funciones *_ready(), get_pos(), _physics_process()* que se describen a continuación:

- ***_ready()***
  - Esta función se ejecuta sólo una vez en el programa y es donde aplicamos la función con la que creamos nuestro script de variables globales.
  - Mario puede estar orientado a derecha o izquierda y, según la orientación, es a dónde se lanzará el martillo, por lo que hacemos uso de un marcador actualizado por Mario.gd y cambiamos el ángulo inicial respecto a este.

```
func _ready():
	if VariablesGlobales.p.x < 0:
		ang = 120
	else:
		ang = 60
```

Imagen 25

- ***get_pos()***
  - En esta función se actualizará la posición del martillo según las fórmulas (1) y (2).

```
func get_pos(delta):
	t += delta
	pos = Vector2(velocidad * cos(ang * (PI /180)) * t, -(velocidad * sin(ang * (PI /180)) * t - 0.5 * g * t * t))
```

Imagen 26

- Para la posición en *y* se aplicó el negativo por la configuración del plano que maneja Godot Engine:

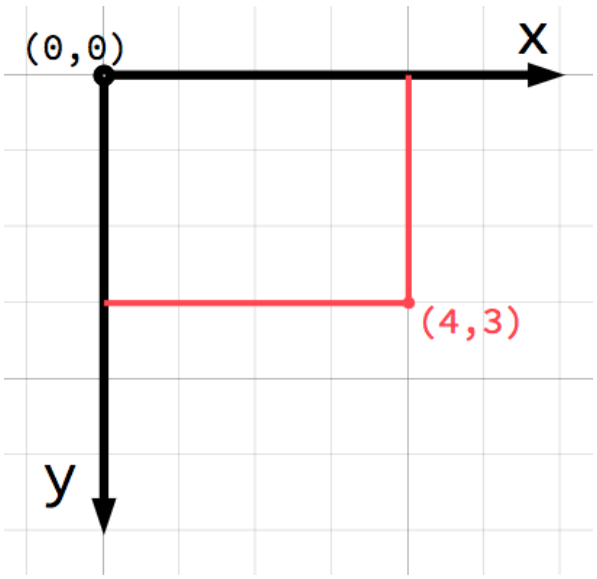
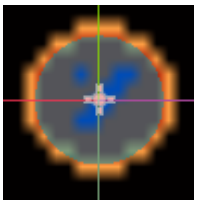


Imagen 27

- ***_physics_process(delta)***
  - En esta función en particular, Godot nos da un marcador del tiempo con que trabajar (delta), este se manda como parámetro a la función *get_pos(),* posteriormente utilizamos *move_and_slide()* para posicionar el martillo en la pantalla según vayamos actualizando *pos*.
  - El sprite del martillo es rotado durante el vuelo, lo cual da una sensación de pesadez al martillo.
  - Liberamos la memoria con *queue_free()* si el martillo toca el piso y mandamos llamar a la variable global *b* para avisar que ya no hay ningún martillo en la escena.

b. BARRIL



Este nodo es de tipo RigidBody2D, lo que nos permite jugar con sus propiedades hasta cierto punto y cumplir con su funcionalidad en el juego de “alocarse”, esto con la única finalidad de agregar complejidad.

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

    pos = move_and_slide(pos, up_direction)

    $Sprite.rotate(sin(ang * (PI /180))*delta*2)
    # De pasar este límite, se libera la memoria.
    if is_on_floor():
        queue_free()
        VariablesGlobales.b = 0
```

Imagen 28

Tiene como script afiliado a Barril.gd, el cual está compuesto por las funciones *_ready(),* _physics*_process(delta)* e *_integrate_forces(state):*

- ***_ready()***
    - En esta función se aplica una fuerza con valores en el vector de (200, 0) N, esto para dar la ilusión de ser lanzado en un inicio por Donkey Kong (Esta escena será instanciada por ese nodo, explicación más adelante).
- **_physics*_process(delta)***
    - Como cuerpo rígido, ya que no lo podemos modificar, Godot nos permite acceder a la posición y velocidad instantánea del nodo, por lo que al llegar a una velocidad 0 en *x*, aplicamos una fuerza en el centro del nodo de (-100, 80) N, esto da la sensación de que está descontrolado y no podamos predecir el movimiento, a menos que conozcas el código.

- ***_integrate_forces(state)***
    - Esta función nos permitirá detectar las colisiones del barril siendo el valor state el que nos proporcionará aquellas fuerzas con las que ha colisionado.En el ciclo for detectará aquellas fuerzas con las que está colisionando.
    - Si el objeto con el que tiene colisión está dentro del grupo llamado Martillo, el barril colisionado se libera con queuwe_free().
    - Si el objeto con el que tiene colisión está dentro del grupo llamado Player, la variable global vida_mario se le resta el valor de 1.

c. MARIO

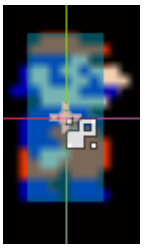


Imagen 29

Este nodo es un Kinematic Body 2D y será nuestro actor principal, el usuario puede desplazarse por las plataformas, hacer que salte, lanzar martillos y rescatar a Peach (princesa). El script afiliado a este nodo, Mario.gd, es el más extenso del programa por la implementación de las funcionalidades mencionadas anteriormente, tiene las siguientes variables iniciales:

```
const ACELERATION = 50
const MAX_SPEED = 100
const JUMP_H = -280
const up_direction = Vector2(0, -1)
var gravity = 20
var t = 0
var on_ladder = false #Se encuentra en escalera o no
var motion = Vector2() #Vector de posición
```

Imagen 30

Y sus funciones se describen a continuación:

- ***_physics_process(_delta)***
    - Esta función es el núcleo de nuestro script, comenzando por el código de aplicar gravedad a Mario, esta se da incrementando el valor de gravedad en *motion.y.*

      ```
      motion.y += gravity
      ```

      Imagen 31
    - El deslizamiento de Mario depende de dónde se encuentre, es decir, sólo puede saltar cuando se encuentre en el piso, sólo se puede subir cuando se encuentre en una escalera, pero se puede deslizar de izquierda a derecha y lanzar martillos cuando sea.

➔ Deslizar izquierda/derecha

- ◆ Se comienza actualizando un marcador de *variables_globales.gd* que se utiliza para saber hacia dónde está orientado Mario.
- ◆ Se voltea el Sprite de Mario hacia el lado al que esté caminando según la entrada del usuario; *ui_right* para tecla →, *ui_left* para tecla ←
- ◆ La línea *$AnimationPlayer.play()* corre la animación preprogramada con los sprites, en este caso *Walk.*

- Finalmente se incrementa la posición horizontal en *motion.x*, se puede notar que se toma el valor más pequeño entre el incremento de *x + ACELERATION* y *MAX_SPEED* para evitar que Mario se mueva demasiado rápido y dificultar la precisión del usuario.

```
if Input.is_action_pressed("ui_right"):
    VariablesGlobales.p.x = 10
    $sprite.flip_h = false
    $AnimationPlayer.play("Walk")
    motion.x = min(motion.x + ACELERATION, MAX_SPEED)
```

```
elif Input.is_action_pressed("ui_left"):
    VariablesGlobales.p.x = -10
    $sprite.flip_h = true
    $AnimationPlayer.play("Walk")
    motion.x = min(motion.x + ACELERATION, -MAX_SPEED)
```

Imagen 32 y 33

➔ Saltar

- Primero se verifica que *Mario* esté sobre el piso.
- De tener de entrada la tecla *Espacio* (*ui_accept*), *Mario* salta y se reproduce el sonido correspondiente.
- *lerp()* disminuirá la animación de caída un 50% para que no se vea como si apareciera en otro lugar de repente.
- Caerá libremente debido a la línea ya descrita (gravedad)

```
if is_on_floor():
    if Input.is_action_just_pressed("ui_accept"):
        motion.y = JUMP_H
        $SonidoBrinca.play()
    if friction == true:
        motion.x = lerp(motion.x, 0, 0.5)
else:
    $SonidoBrinca.stop()
    if friction == true:
        motion.x = lerp(motion.x, 0, 0.01)
```

Imagen 34

➔ Disparo

- En este caso, al presionar la tecla *E*nter (definida manualmente como *d* en configuración del programa), se mandará llamar a la función *shoot()*

```
if Input.is_action_pressed("d"):
    if VariablesGlobales.b == 0:
        shoot()
```

Imagen 35

➔ Escaleras

- *on_ladder* es una variable actualizada por una señal conectada entre *Mario* y el *Area2D* parte del nodo de escaleras, esta nos avisa cuando si puede o no subir una escalera cuando se presiona la tecla ↑.
- Por puros fines funcionales, la gravedad se hace 0, esto es necesario para que realmente el jugador pueda subir, vuelve a la normalidad cuando el jugador sale del *Area2D.*

```
if on_ladder == true:
    if Input.is_action_just_pressed("ui_accept"):
        motion.y = JUMP_H
        print("Brinca")
    if Input.is_action_pressed("ui_up"):
        #sprite.visible = false
        #sprite2.visible = true
        $AnimationPlayer.play("Sb_Esc")
        gravity = 0
        motion.y = min(motion.y + ACELERATION, -MAX_SPEED)

    elif Input.is_action_pressed("ui_down"):
        gravity = 0
        #sprite.visible = false
        #sprite2.visible = true
        $AnimationPlayer.play("Sb_Esc")
        motion.y = min(motion.y + ACELERATION, MAX_SPEED)
    else:
        motion.y = 0
```

Imagen 36

➔ *Muerte/Ganó*

- Aquí nuevamente se utilizan las variables globales para reiniciarse según el tipo de muerte que tenga *M*ario (por tiempo, resta de vidas o vidas terminadas) o si gana.
- Al ser resta de vidas se verifica si Mario ha muerto, se le resta una vida. Se le vuelve a "*dar vida"*, por decirlo de alguna forma, posicionándolo desde el comienzo del nivel.

- Al ser muerte definitiva, se manda llamar a la función *reinicia()*
- Cuando *Mario* gana, nos avisa la variable *g_b* de variables globales, se manda llamar a la función *reinicia(),* actualizamos *g_b* y regresamos a la escena principal del juego, dónde el usuario decidirá si quiere volver a jugar.

```
if VariablesGlobales.g_b == true:
    reinicia()
    VariablesGlobales.g_b = false
    get_tree().change_scene("res://Node.tscn")

if VariablesGlobales.vida_mario == false:
    VariablesGlobales.vidas -= 1
    VariablesGlobales.vida_mario = true
    position = Vector2(94, 643)

if VariablesGlobales.vidas == 0:
    reinicia()

if VariablesGlobales.tiempo_juego == 0:
    reinicia()
```

Imagen 37

- ***reinicia()***
  - Esta función se utiliza para reiniciar todas las variables globales correspondientes a las vidas y barriles del juego.
  -

```
func reinicia():
    VariablesGlobales.vidas = 3
    VariablesGlobales.vida_mario == true
    VariablesGlobales.barriles = 0
    VariablesGlobales.tiempo_juego = 180
    VariablesGlobales.b_dest = -1
    position = Vector2(94, 643)
    get_tree().change_scene("res://Node.tscn")
```

Imagen 38

- ***shoot()***
  - Esta función es utilizada para disparar, pero lo que la describe mejor es instanciar, ya que es precisamente lo que hace cuando se manda llamar al presionar *Enter*.

- Lo primero es ‘avisar’ en las variables globales que hay un martillo en la escena, a través de *b*, instanciamos las escena de martillo y la posicionamos donde se encuentre *M*ario.
    - Hay que notar la variable *p* de variables globales, la cual sumamos a la posición, para que el martillo no salga directamente desde el centro de *Mario*, sino unos píxeles más en frente de él.

```
func shoot():
    VariablesGlobales.b = 1
    var proyectil = load("res://Martillo.tscn")
    var bullet = proyectil.instance()
    bullet.position = get_global_position() + VariablesGlobales.p
    get_parent().add_child(bullet)
```

Imagen 39

d. DONKEY KONG

Este nodo es un Kinematic Body 2D y es nuestro villano, su función es lanzar los barriles para que se desplacen por las plataformas. El script que está afiliado a este nodo es el de DonkeyKong.gd, su única variable inicial se muestra a continuación (Imagen 41):

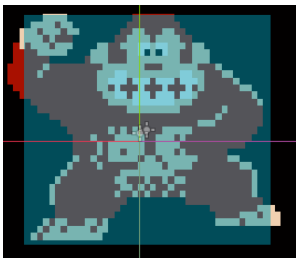


Imagen 40

```
3  var t
```

Imagen 41

Sus funciones son las siguientes:

- ***ready()***
    - Esta función se ejecutará al principio y cuando se inicia el juego, solo contiene el valor de t que será 0.

```
5 func _ready():
6     t = 0
```

Imagen 42

- ***lanza_barril()***
    - Esta función se usa para que Donkey Kong lance los barriles, instancia la escena de los barriles, y así parece que los lanza por un lado.

```gdscript
func lanza_barril():
    var barril = load("res://Barril.tscn")
    var barr = barril.instance()
    barr.position = get_global_position() + Vector2(45, 0)
    get_parent().add_child(barr)
```

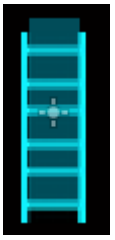
Imagen 43

- ***_process(delta)***
    - Siendo la función de proceso principal de nuestro Donkey Kong, la labor es verificar el lanzamiento de los barriles que serán lanzados 100 ciclos mientras no sobrepase el valor de 25, ya que 25 barriles solo son los que lanzará personaje. Además incrementa la variable global barriles que contará los barriles que se han lanzado.
    - El segundo if verifica que si se lanzaron los 25 barriles, el personaje cambiará de animación, mientras siga lanzando barriles hará la animación de lanzar barriles.

```gdscript
func _process(delta):
    t += 1
    #Donkey Kong realiza los lanzamientos de barriles cada 100 ciclos
    #transcurridos.
    if t > 100:
        if VariablesGlobales.barriles < 25:
            lanza_barril()
            t = 0
            VariablesGlobales.barriles += 1
    #Si ya lanzó todos los barriles, hace la animación de golpearse
    #el pecho
    if VariablesGlobales.barriles == 25:
        get_node("AnimationPlayer").play("golpe")
    #Si no, mantiene la animación de lanzar barriles
    else:
        $Sprite.flip_h = false
        get_node("AnimationPlayer").play("recogerAventar")
```

Imagen 44

e. ESCALERA



Este es un nodo Area2D que es hijo de nuestra escena principal *Game*. La función es detectar las escaleras que están posicionadas a lo largo del nivel por medio de un CollisionShape2D.

Imagen 45

- ***_on_esc_body_entered(body)***
    - Esta función detectará si hay una colisión de cuerpos en las escaleras a lo largo del nivel en especial con el nodo llamado “Mario”. Si la hay, cambia el valor de on_ladder a true, que se encuentra en el script de Mario.

```
func _on_esc_body_entered(body):
    if body.name == "Mario":
        get_node("../Mario").on_ladder = true
```

Imagen 46

- ***_on_esc_body_exited(body)***
    - En este caso, si un cuerpo deja de estar en contacto con las escaleras es detectado, en especial para el nodo llamado “Mario”. Si no hay colisión cambia el valor de on_ladder a false, que se encuentra en el script de Mario.

```
func _on_esc_body_exited(body):
    if body.name == "Mario":
        get_node("../Mario").on_ladder = false
```

Imagen 47

f. NODE

Esta escena será la encargada de mostrar la pantalla de inicio y su script asociado es el de Node.gd el cual solo consta de una sola función.

Imagen 48

- ***_on_Button_pressed()***
    - La pantalla de inicio consta de un botón, si este es presionado se reproduce el sonido de inicio además de cambiar la escena a Game, en donde se encuentra el juego.

```
11 func _on_Button_pressed():
12     $Intro.play()
13     get_tree().change_scene("res://Game.tscn")
```

Imagen 49

g. PRINCESA

Esta escena es un Area2D que estará situada en la cima de las plataformas del nivel, su script asociado es el de Princesa.gd y tendrá dons funciones:

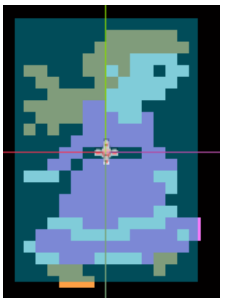


Imagen 50

- ***_ready()***
  - Función que se ejecuta al principio y su funcionalidad es reproducir la animación de la princesa con $AnimationPlayer.play(), la cual consiste en voltear de derecha a izquierda.

```
3 func _ready():
4     $AnimationPlayer.play("movimiento")
```

Imagen 51

- ***_on_Princesa_body_entered(body)***
  - El trabajo de esta función es detectar la colisión con un objeto, en donde se especifica si se detecta el nodo llamado “Mario”, si es así se actualizan tres variables globales:
    - gano, que incrementa el valor de juegos ganados.
    - g_b, un booleano que cambia su valor a true, que nos dice que Mario ha ganado.
    - donkey_die, un booleano que cambia su valor a true, y nos dice que Donkey Kong ha perdido.

```
 8 func _on_Princesa_body_entered(body):
 9     if body.name == "Mario":
10         VariablesGlobales.gano += 1
11         VariablesGlobales.g_b = true
12         VariablesGlobales.donkey_die = true
```

Imagen 52

4. Conclusiones

Aguilar Camacho Jesús Darío:

El trabajar con un software que desconocía al igual que su lenguaje de programación como lo es *Godot* fue todo un reto, ya que me llevó tiempo adaptarme a su entorno. El seguir tutoriales de esa comunidad me facilitó las cosas y hasta obtuve más conocimientos que no hubiera encontrado solo leyendo la documentación. Me gustaría seguir desarrollando este tipo de juegos y algunos otros como shooters, o tan solo no alejarme tanto del tema.

En general me parece que realizamos un buen trabajo, aún faltan algunos detalles extras que no resolvimos al cien por ciento pero no afectan mucho la funcionalidad del juego, excepto por brincar mientras colisiona Mario con las escaleras. Aún puede mejorarse pero nos faltó invertirle más tiempo.

Me gustó el trabajo logrado y haberlo terminado en su mayoría como lo teníamos en mente, aunque cruzó por muchos cambios e ideas, se logró plasmar lo que queríamos.

Ariana Ishihara Moros:

Al comienzo del proyecto, cuando poníamos nuestros primeros nodos y escribíamos los primeros scripts, me sentí un poco perdida a cómo manejar las unidades de trabajo de Godot y a ubicarme en cómo se maneja el plano ahí, pero investigando un poco y revisando la documentación que ofrece Godot, comenzó a fluir y desarrollarse con mayor facilidad el proyecto. Se hace muy intuitivo ya que pasadas algunas horas programando te familiarizas con el entorno y es muy parecido al lenguaje *Python*.

En lo personal tuve algunos problemas con la detección de colisiones con los barriles y el martillo, ya que como son de diferentes clases, no podía hacer la relación entre los nodos, pero mi compañero lo resolvió creando *grupos* y aplicando algunas funcionalidades de Kinematic Body a su favor.

Creo que, hasta dónde avanzamos, quedó muy bien, pero un proyecto siempre puede ser mejor, por lo que arreglaría algunos efectos de animación, ya que cuando Mario sube las escaleras se queda estático, también agregaría algunos niveles más dinámicos, como unos presentes en el juego original que tienen elevadores y llamas de fuego.

5. Referencias:

➔ Assets

Imagen 53

- Son assets tomados de sheets de la web presentes en este link:
- http://www.mariouniverse.com/sprites-nes-dk/

➔ Videos de referencia:

- Escaleras: https://www.youtube.com/watch?v=QUFuzqaG93o&t=429s
- Martillo (bullets): https://www.youtube.com/watch?v=8BBR03L3He4
- Colisión barriles/martillos: https://www.youtube.com/watch?v=g0yNeSkg5JU