

Documentatie Proiect

Cuprins:

I.Creare nivel

II.Creare inamic care arunca flacari

III.Creare calaret care alearga pe cal (creare inamic)

IV.Creare Coin

V.Creare Portal

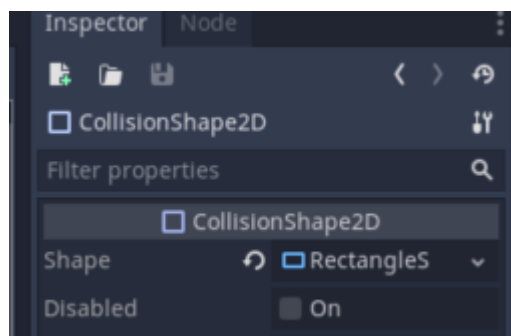
VI. Adaugare Muzica

VII.Testare

I.Creare nivel

1. Creare player

- Other node-> KinematicBody2D¹ -> se va numi Player + adăugare detector de coliziune (CollisionShape2D)
- Forma detectorului: Inspector -> Shape -> Rectangle
- Adăugare imaginea(Sprite²) playerului cu drag and drop player.png



Atașarea celor 2 scripturi:

- Player.gd (clasa jucătorului): Add -> Language GDScript, Inherits Actor, Empty Template
- Actor.gd (clasa prin care se va stabili relația jucător-inamici): extinde KinematicBody2D și reprezintă clasa de baza pentru Enemy și Player
 - Creez clasa Actor : class_name Actor
 - func **_physics_process**(delta³: float) -> void:
 - > Godot o să o cheme automat în fiecare frame (de 40 sau 60 ori pe secundă)
 - > tot aici se efectuează mutările actorilor
 - Metoda din KinematicBody2D: Vector2⁴ **move_and_slide**(linear_velocity: Vector2, up_direction: Vector2 = Vector2(0, 0), stop_on_slope: bool = false, max_slides: int = 4, floor_max_angle: float = 0.785398, infinite_inertia: bool = true)

¹ Este destinat corpurilor de implementare care sunt controlate prin cod. Corpurile cinematice detectează coliziuni cu alte corpuri atunci când se deplasează, dar nu sunt afectate de proprietățile fizice ale motorului (...) înseamnă că trebuie să scrieți un cod pentru a le crea comportamentul și că aveți un control mai precis asupra modului în care se mișcă și reacționează.

² Un nod care afișează o textură 2D.

³ Valoare dată de Godot care reprezintă timpul scurs de la ultimul frame.

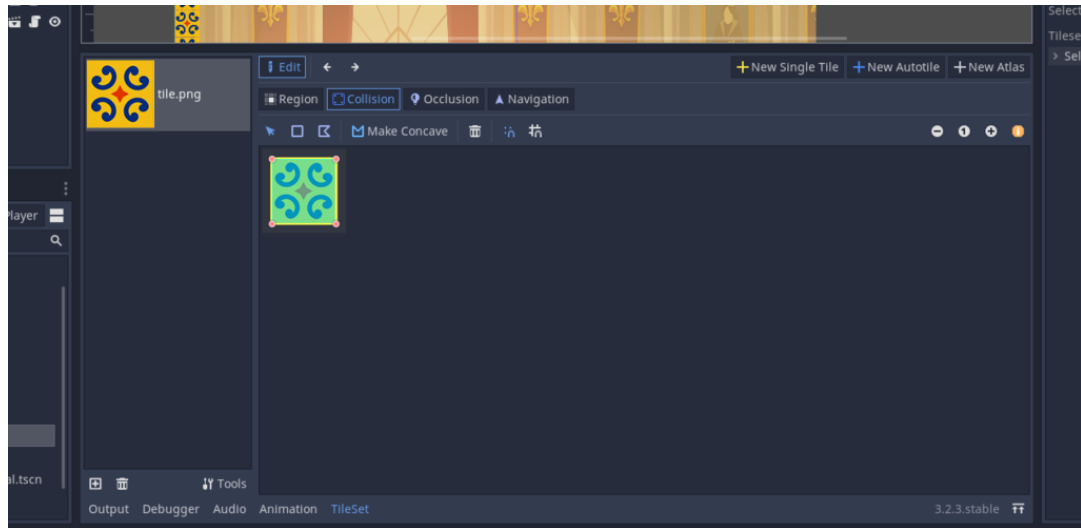
⁴ Structură cu 2 elemente care poate fi utilizată pentru a reprezenta poziții în spațiul 2D sau orice altă pereche de valori numerice.

-> mutarea unui corp de-a lungul Vector2

- var velocity
 - > viteza liniară
 - > tip: Vector2
 - > dată membru a clasei Actor se comportă ca o variabilă globală
 - > valoare default: var velocity = Vector2.ZERO (corpul inițial nu o să se miște)
 - > Vector(300, 0) -> se mută cu 300px/sec pe axa Ox și 0px/sec pe Oy
- var gravity
 - > se aplică și valoarea unei gravitații de-a lungul mișcării
 - > se aplică pe Oy și este o accelerație egală cu 3000.0px
- velocity.y += gravity * delta
 - > mișcarea caracterului este constantă indiferent de performanța computerului
- **export** var gravity
 - > i se poate schimba valoarea din Inspector-> Categoria Script Variables
- var speed = Vector2(300.0, 1000.0)
 - > viteză maximă pe Ox și pe Oy

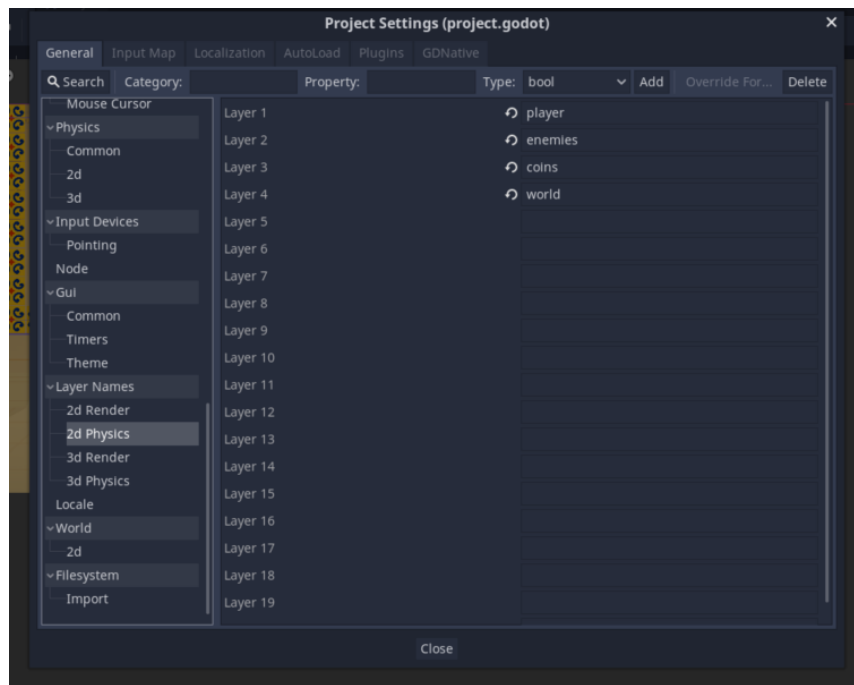
2. Creare decor

- 2D Scene -> Adaug un nou nod -> TileMap -> Inspector -> Categoria Tile Map -> Tile Set -> New Tile Set -> Deschid editorul pentru tile set -> (+) incarc imaginea tile.png de dimensiune 80x80px
- Adăugare coliziune (butonul Collision) pentru a *susține* actorii.



- New Single Tile -> Inspector -> Categoria Snap Options -> Stop x=80, y=80
- Inspector-> TileMap -> Cell -> Size x=80, y=80 ca textură
- Adăugare player în scenă prin drag and drop Player.tscn, amplasat ca, copil al scenei nu al nodului TileMap.
- Creare layere pentru fiecare entitate: player, enemies, coins, world:

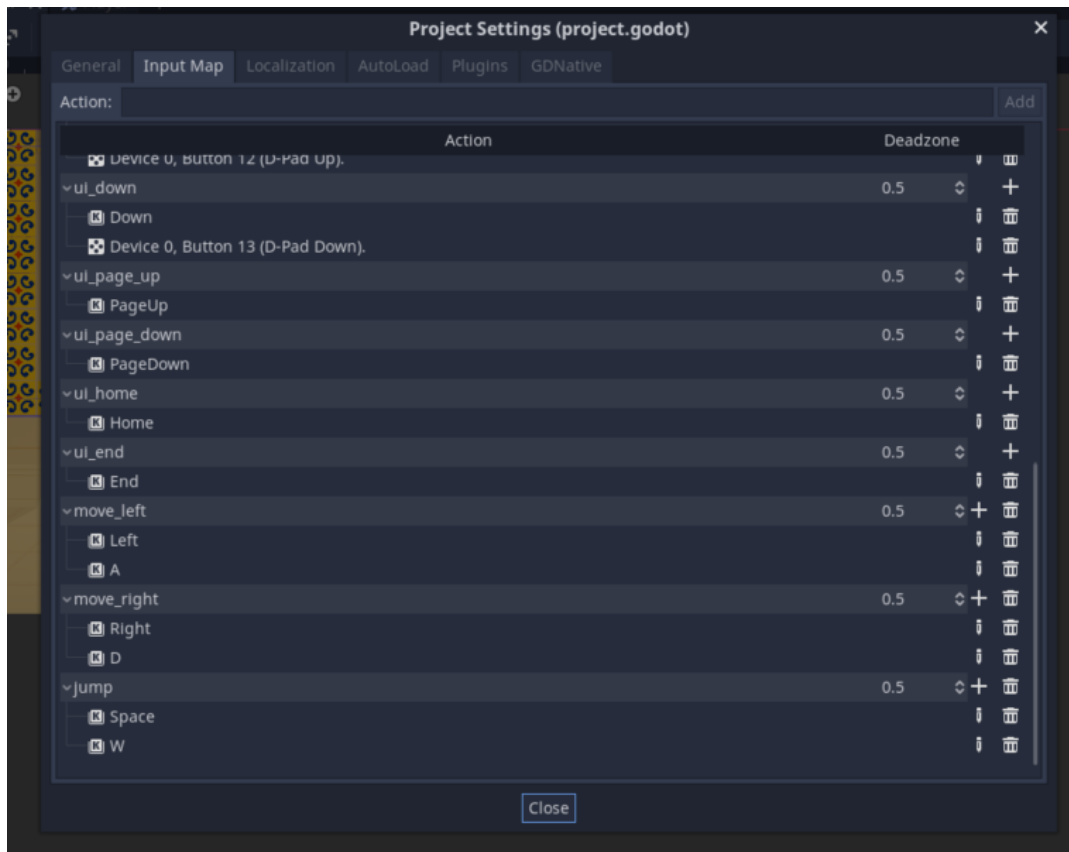
Project-> Project Setting -> Layer Names -> 2D Physics



- Evitare bug-uri pentru decor: Inspector -> Collision -> Layer⁵(selectat doar pentru world) -> Mask deselctez tot)

⁵ Dacă este nevoie de o listă mare și / sau dinamică de excepții este eficient să se folosească sistemul de strat(Layer – de ce se lovește) / mască de coliziune(Mask - ce ignoră).

➤ Personalizare taste de input: Project Settings -> Input Map



3. Scripting: Player.gd

- **Input.get_action_strength("nume_mutare")**
 -> dacă cheia corespunzătoare mutării este apăsată va returna 1, altfel va returna 0, iar dacă nu este apăsată complet va returna o valoare între 0 și 1
 -> `Input.get_action_strength("move_left")`, -1.0 if⁶ `Input.is_action_just_pressed("jump")` and `is_on_floor()` else 0.0
 -> 1.0-caracterul pica de sus pe pamant si -1.0 ca sa sara si se va muta si pe OX si OY
 -> pentru dreapta are valoarea 1 si la stanga este -1 (daca se apasa pe amandoua in acelasi timp in direction.x o sa fie 0)
- func **calculate_move_velocity**(linear_velocity: Vector2,direction: Vector2,speed: Vector2,is_jump_interrupted: bool) -> Vector2
 -> creez abilitatea de a opri căderea
 -> `new_velocity.y += gravity * get_physics_process_delta_time ()` ->mutare în jos cu o funcție predefinită
- `Vector2.UP ⇔ (0,-1)` este o constantă din Vector2
- `is_jump_interrupted := Input.is_action_just_released("jump")` and `_velocity.y < 0.0`

⁶ Operator ternar.

-> la eliberarea cheii pentru salt => o săritură cu o intensitate în funcție de cât de tare se apasă cheia

4. Creare inamic

- Duplicarea scenei jucătorului
- Click and drag în Sprite (Inspector) -> Texture

5. Scripting: Enemy.gd

- Metoda din KinematicBody2D **is_on_wall()**
-> se actualizează după ce se mută actorul
- func **_ready()** -> void:
-> inamicul se mișcă mereu spre player
-> predefinită de Godot este chemată automat de fiecare nod al scenei începând de sus în jos și de la cel mai adânc spre suprafața nivelelor
 - **set_physics_process(false)**
-> inamicul nu va începe să se miște până când nu intră în vizor
- **velocity.x = -speed.x**
-> merge la stânga cu viteza dată
- **velocity.y = move_and_slide(velocity, FLOOR_NORMAL).y**
-> nu se mai oprește când se lovește de un perete
- Optimizare inamic:
-> inamicul nu se mișcă când e departe de player
-> creare nod VisibilityEnabler2D⁷ -> Inspector-> activez Physics Process Parent (când iese din vizor se va opri) și când va trece de player nu se mai întoarce

6. Creare urmărire player (Camera)

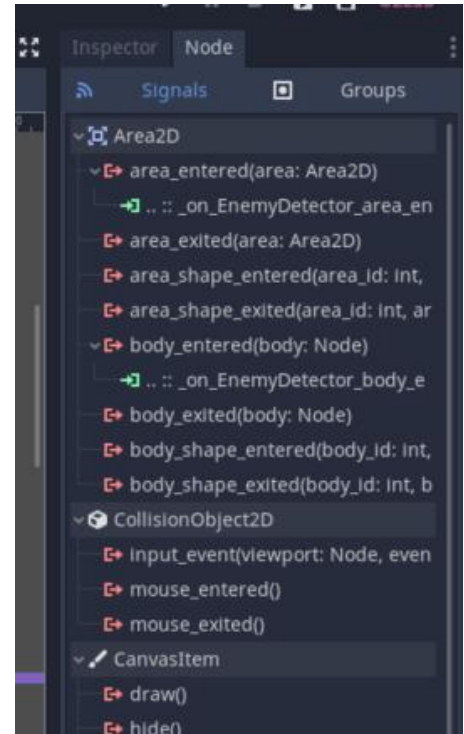
- Player-> Nodul Camera2D⁸ -> Inspector -> Current -> On => camera va urmări playerul

⁷ Va dezactiva RigidBody2D, AnimationPlayer și alte noduri atunci când acestea nu sunt vizibile. Acesta va afecta doar nodurile cu același nod rădăcină ca VisibilityEnabler2D și nodul rădăcină în sine.

⁸ Forțează ecranul (stratul curent) să deruleze după acest nod. Acest lucru face mai ușoară (și mai rapidă) programarea scenelor defilabile decât schimbarea manuală a poziției nodurilor bazate pe CanvasItem.

7. Creare abilitate de a sări pe inamici și de a-i omori

- Lui Enemy i se adaugă nodul Area2D⁹ -> se va numi StompDetector -> Inspector -> Categoria Collision -> Layer (nimic selectat) și Mask (selectare player)
- Lui StompDetector i se adaugă nodul CollisionShape2D
- Feature al Godot: există aceste semnale(signals)
 - > Mod de adăugare: Node -> Dublu Click pe una din funcții -> Creare și editare semnal
 - > Numele va fi alcătuit automat din _on_+ numele nodului + _numele semnalului

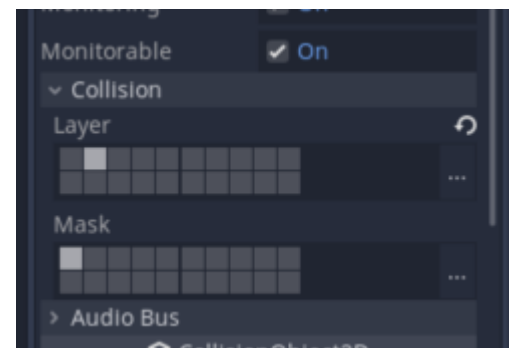


➤ Scripting Enemy.gs

- funcția de semnal func
_on_StompDetector_body_entered (body: PhysicsBody2D) -> void:
 -> inamicul o să fie omorât în momentul în care playerul îi sare pe cap
- if body.global_position.y > **get_node** ("StompDetector").global_position.y:
 -> comparare poziție globală pe Oy a lui stomp detector cu poziția jucătorului care intră în zonă
 -> dacă nu intră pe condiție, se va apela funcția **queue_free()** care va șterge inamicul
- **get_node**("CollisionShape2D").disabled=true
 -> jucătorul nu va fi distrus când sare pe inamic să-l ucidă

8. Inamicul poate ucide jucătorul:

- > în Player se creează un detector ca să-l omoare dacă se întâlnește cu inamicul:
- Area2D -> se va numi EnemyDetector -> Inspector-> not Monitorable -> Layer(dezactivat tot)
- Lui EnemyDetector2D i se adaugă un nou nod de coliziune CollisionShape2D
- Setări pentru ca, coliziunea jucătorului să fie detectată de inamic : (vezi imagine)
- Semnal pentru jucător -> func **_on_EnemyDetector_area_entered**(area: Area2D) -> void:
- **calculate_stomp_velocity**(_velocity, stomp_impulse)
 -> face caracterul să sară mai sus când sare pe capul unui inamic
- out.y = -impulse



⁹ Zona 2D care detectează nodurile CollisionObject2D care se suprapun, intră sau iese. Poate, de asemenea, modifica sau suprascrive parametrul local de fizică (gravitație, amortizare).

-> stomp_impulse îl face să difere față de o săritura normală
 -> aici se pot face modificări în cum să sară (la diferite înălțimi etc.)

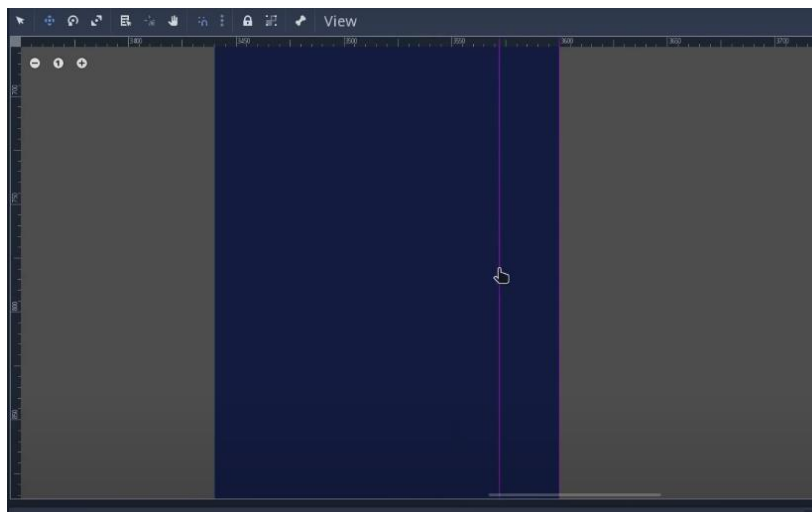
- func **_on_EnemyDetector_body_entered**(body) ->void:
 -> se crează un nou semnal pentru abilitatea jucătorului de a muri

9. Modificări ale camerei ca să se miște mai ușor:

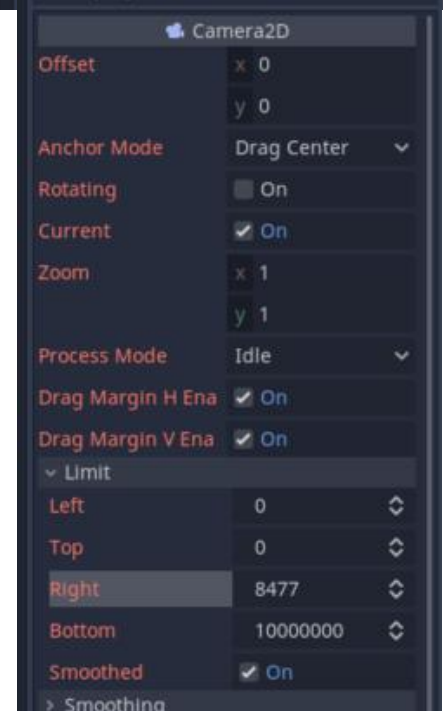
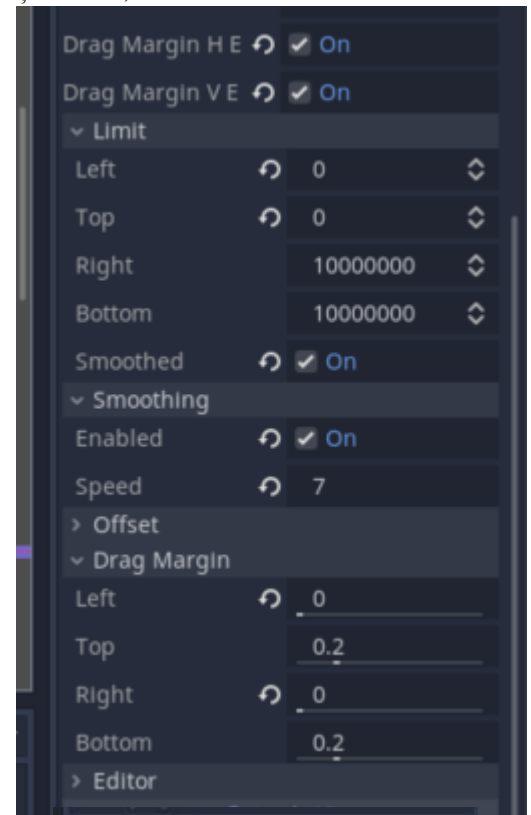
- Camera2D -> Inspector -> Categoria Limit left=0 (începe de pe partea dreaptă a Oy) , top=0 (va fi sub OX) -> Smoothed bifat(nu se oprește abrupt în marginile ecranului)
- Drag Margin Box -> left=0, right=0 -> mereu se centrează pe dreapta și pe stânga după jucător
- Smothing->Enable Smothing -> enable -> se mișcă mereu (cu cât e mai mică valoarea vitezei cu atât se mișcă mai greu)
- Oprește camera la sfârșitul nivelului

->Rigla se mută până la sfârșitul nivelului, iau valoarea la care se termină și schimb valoarea dând click dreapta pe player (din level template medieval) -> Editable Children -> Camera2D -> Inspector -> Categoria Limit -> adăugare valoare în câmpul Right

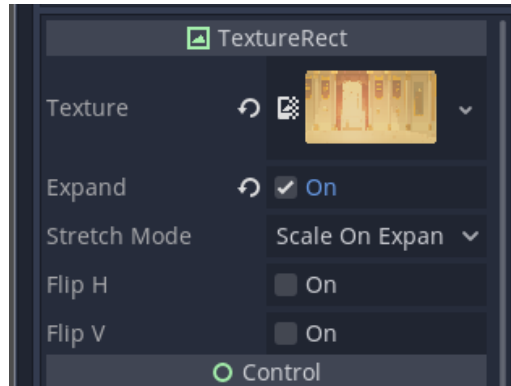
10. Adaugare fundal



- Assets -> click and drag pe tablă -> drag & drop + Alt => meniul -> TextureRect -> fundalul trebuie să fie un copil al nivelului
- Inspector -> Expand -> On
- Layout(din Toolbar) -> full rect -> umple toată ecranul



- Ca imaginea de background să poată sta tot jocul fix în fundal-> nodul CanvasLayer¹⁰ (copil imediat al lumii, iar background-ul devine copil al canvasului)
- Inspector -> Proprietatea Layer Index in CanvasLayer -> setare valoare negativă => backgroundul va fi mereu în fundal



II. Creare inamic care arunca flacari:

1. Creare nod nou inamic

- Duplicare inamic de tipul 1
- Eliminare nod StompDetector (jucatorul nu-l va putea omori, va trebuie sa treaca de el si sa se fereasca de bombe)
- Add Child Node -> Area2D
 - Add Child Node -> CollisionShape2D
 - rol: va declansa automat aruncarea bombelor atunci cand playerul intra in raza acesteia
 - Adaugare semnale:
 - _on_Area2D_body_entered() –rol: declansat cand jucatorul intra in raza de actiune a Area2D
 - _on_Area2D_body_exited()
- Add Child Node -> Timer
 - Adaugare semnal pe Timer: timeout()
- Atasare script : EnemyType2.gd (asemanator cu scriptul inamicului de tipul 1)

¹⁰ Stratul de desen pe pânză. Nodurile CanvasItem care sunt copii direcți sau indirecti ai unui CanvasLayer vor fi desenate în acel strat.(...) Scena 2D implicită se redă cu index 0, deci un CanvasLayer cu index -1 va fi desenat mai jos, iar unul cu index 1 va fi desenat mai sus.

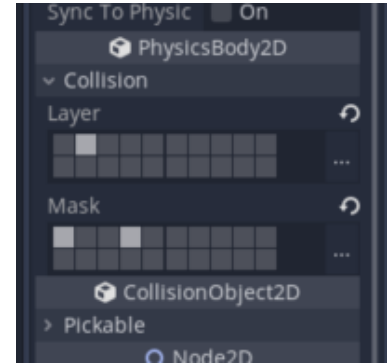

```

1
2 extends "res://src/Actors/Actor.gd"
3
4 # abilitatea de a arunca cu flacari
5 onready var BULLET_SCENE = preload ("res://src/Actors/Bullet.tscn")
6
7 var player = null
8 var move = Vector2.ZERO
9
10
11 func _ready() -> void:
12     set_physics_process(false)
13     # daca ar fi negativa ar porni spre stanga => merge spre jucator
14     # directia normala de mers este spre dreapta de-a lungul axei ox
15     _velocity.x = -speed.x * 1.5
16
17 func _physics_process(delta):
18     _velocity.y += gravity * delta
19
20     if is_on_wall():
21         _velocity.x *= -1.0 #schimbare directie de deplasare
22         $enemy2.scale.x *= -1.0 #rotatie imagine inamic cand isi schimba directia
23         _velocity.y = move_and_slide(_velocity, FLOOR_NORMAL).y
24
25 func _on_Area2D_body_entered(body): # semnal pt Area2D
26     player = body
27
28 func _on_Area2D_body_exited(body): # semnal pt Area2D
29     player = null # nu vrem sa mai fie urmarit
30
31 func fire(): #pentru aruncarea cu bombe
32     var bullet = BULLET_SCENE.instance() #instantiere scena
33     bullet.position = get_global_position() # va incepe de la pozitia aruncatorului
34     bullet.player = player # se duce catre player
35     get_parent().add_child(bullet)
36     $Timer.set_wait_time(0.7) #intervalul la care se arunca bombele
37
38 func _on_Timer_timeout(): # semnal pt Area2D
39     # rol: pentru a controla cat de rapid inamicul aruna cu bombe
40     if player != null:
41         fire()

```

2. Creare nod nou ce va reprezenta o flacara

- KinematicBody2D -> se va numi Bullet
 - Adaugare script: Bullet.gd (va exista in KinematicBody2D)
- Add Child Node -> Sprite (adaugare imagine ca la inamic)
- Add Child Node -> CollisionShape2D (alegere forma ca la inamic)
- Setari de coliziune (ca la inamic)
- Atasare script : Bullet.gd
- Metode specifice Vector2() utilizate:
 - Vector2 move_toward (Vector2 **to**, float **delta**)
Moves the vector toward **to** by the fixed **delta** amount.
 - Vector2 normalized ()
Returns the vector scaled to unit length. Equivalent to $v / v.length()$.



```
1 extends KinematicBody2D
2
3 var move = Vector2.ZERO
4 # va retine directia playerului
5 var player_direction = Vector2.ZERO
6 var player = null
7 #viteza de deplasare a bombelor
8 export var speed = 3
9
10 func _ready():
11     #pozitia playerului - pozitia inamicului
12     player_direction = player.position - global_position
13     # pentru a fi mai orientate catre player
14     player_direction.y -= 50
15 func _physics_process(delta):
16     move = Vector2.ZERO
17
18     move = move.move_toward(player_direction, delta)
19     move = move.normalized() * speed
20     position += move
21
```

III. Creare calaret care alearga pe cal (creare inamic sub forma de tool¹¹ in editor):

1. Creare nod nou inamic

- Duplicare inamic de tipul 1
- Maresc VisibilityEnabler2D pentru ca inamicul sa inceapa sa se miste inainte sa ajunga in vizorul camerei
- Sterg nodul StompDetector pentru ca calaretul sa nu poate fi omorat de catre jucator, ci doar evitat prin saritura.
- Atasare script : Horse2.gd duplicare cod Enemy.gd si adaugarea functiei _draw() care va desena o curba sinusoidala in editor care va reprezenta traiectoria sariturii pe care o va face personajul.

```
1  tool
2  extends "res://src/Actors/Actor.gd"
3  # curba sinusoidala
4  # <0 spre stanga
5  # >0 spre dreapta
6  var sine_move = -400
7  # lungimea unei linii intrerupte
8  var length= 0.0
9
10 var start_position = Vector2() # initializare
11
12 # cat de inalte vor fi curbele
13 export(float) var sine_amplitude = 80 setget _set_sine_amplitude
14 # setget se asigura de modificarea valorii din editor in cod
15 # si apoi rezultatul in editor in timp real
16
17 func _ready() -> void:
18     start_position = position
19
20     set_physics_process(false)
21     if speed.x < 0 : # daca ii dau viteza mai mica
22         $enemy.scale.x *= -1.0
23     _velocity.x = -speed.x
```

¹¹ Tool-ul (instrumentul) este o linie puternică de cod care, atunci când este adăugată în partea de sus a scriptului, îl face să se execute în editor.

```

31 ~ func _physics_process(delta: float) -> void:
32 ~     # functie apelata la fiecare frame
33 ~
34 ~     _velocity.y += gravity * delta
35 ~     if is_on_wall(): # se loveste de un perete
36 ~         sine_move*=-1.0 # isi schimba directia de deplasare(linia)
37 ~         _velocity.x *= -1.0
38 ~         $enemy.scale.x *= -1.0 # rotatie imagine inamic cand isi schimba directia
39 ~         _velocity.y = move_and_slide(_velocity, FLOOR_NORMAL).y
40 ~
41 ~     if Engine.editor_hint:
42 ~         # true daca scriptul inca ruleaza in editor
43 ~         # good practice
44 ~         # previne rularea accidentala a codului
45 ~         # care ar afecta starea scenei in timp ce se afla in editor
46 ~         return
47 ~
48 ~     length += delta
49 ~     # amplific viteza de miscare
50 ~     position.x +=sine_move *delta
51 ~     # calcul amplitudine curba sinusoidala
52 ~     position.y = start_position.y + sin(length*10) * sine_amplitude
53
54 ~ func _set_sine_amplitude(value):
55 ~     sine_amplitude = value
56 ~     update()
57 ~
58 ~ # functie cu care desenez traiectoria in editor
59 ~ func _draw():
60 ~     if not Engine.editor_hint:
61 ~         return
62 ~
63 ~     # vector de puncte
64 ~     var points_array = PoolVector2Array()
65 ~     var number_points=100 # numarul de puncte de pe traiectorie
66
67 ~     for i in range(number_points):
68 ~         # 0.02 masura unui pas
69 ~         var l= i * 0.02 #lungimea liniei intrerupte
70 ~         # coordonate punct (x, y)
71 ~         var point = Vector2(sine_move * l, sin(l * 10) *sine_amplitude)
72 ~         points_array.append(point)
73 ~     # functie care va va desena linii intrerupte
74 ~     # primeste vectorul de puncte
75 ~     # culoarea liniilor
76 ~     draw_multiline(points_array, Color('#ffffff'), 2.0)

```

IV.Creare Coin

Pentru ca jocul să aibă un impact mai mare asupra utilizatorilor, pe parcursul tuturor nivelelor se vor găsi banuti, care în funcție de nivelul la care ne aflăm are o diferită formă.

Nivelul 1/2/3: Diamante

Nivelul 4/5/6: Floricele

Nivelul 7/8/9: Stelute

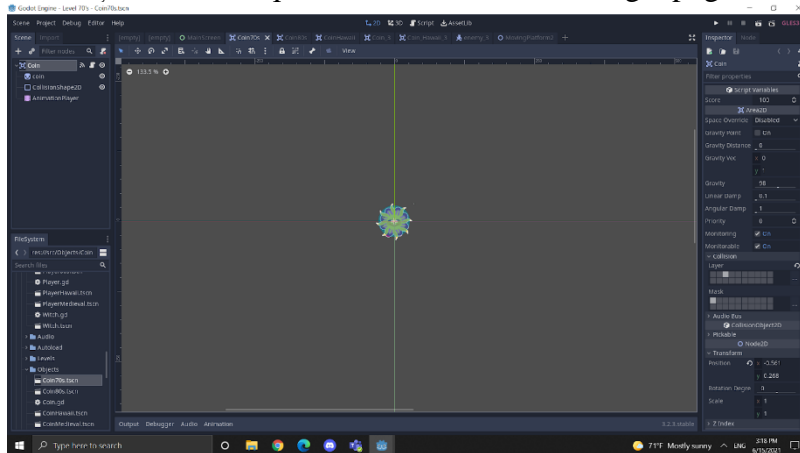
Nivelul 10: Ananasi

Nivelul 11: Ananasi

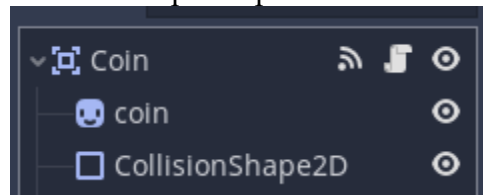
Nivelul 12: Pesti

Pentru crearea lor am făcut următorii pași:

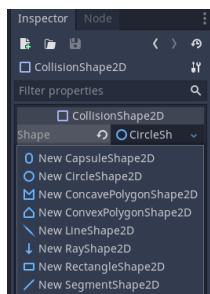
1. Am creat o nouă scenă:
Scene -> New scene -> Custom Node-> Area2D (Coin)
2. În funcție de nivelul pe care l-am creat, am adăugat png-urile dorite prin drag&drop.



3. La fel ca playerul și coin-urile vor avea nevoie de un CollisionShape2D.
click dreapta pe "Coin" -> New Node -> CollisionShape2D

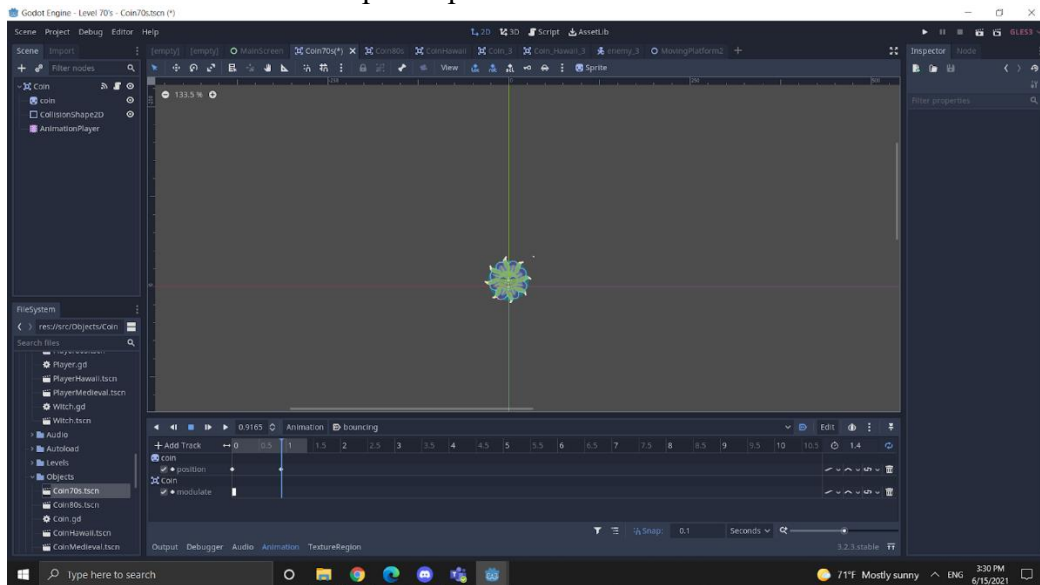


Pentru ca sa acoperim o cat mai mare parte din coin, de la opțiunea "shape" alegem forma pe care o dorim

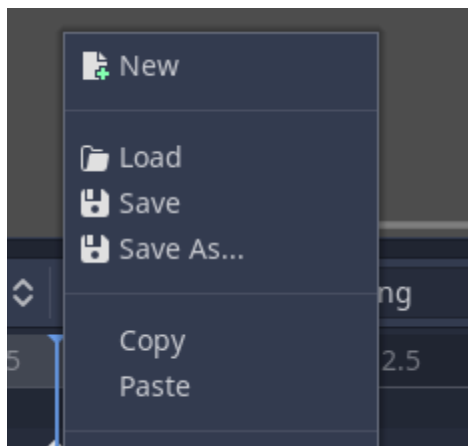


Pentru a ne asigura ca banutul nostru dispere treptat atunci cand am fost capturat am făcut următoarele setări:

click dreapta pe “Coin” -> New Node -> AnimationPlayer



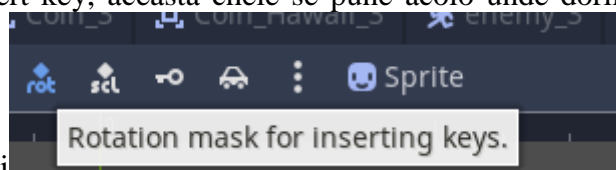
În partea de jos a paginii vom da click pe “Animation” și după aceea vom alege opțiunea “new”, astfel vom crea o animație.



Vom avea doua astfel de animatii:

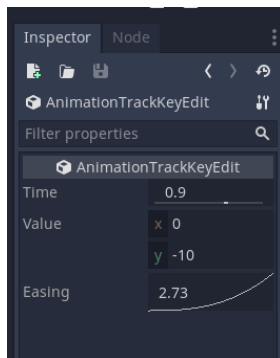
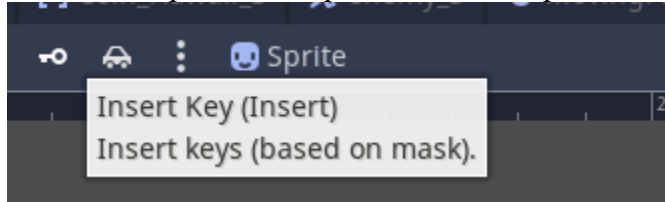
a. bouncing

- vom folosi căsuța din dreapta butonului de “ Add track” unde vom seta durata de timp in care dorim sa se realizeze animatia
- în partea de sus al ecranului avem butonul specific rotației, pe care il vom selecta
- langa acest buton se afla cel de insert key, aceasta cheie se pune acolo unde dorim sa

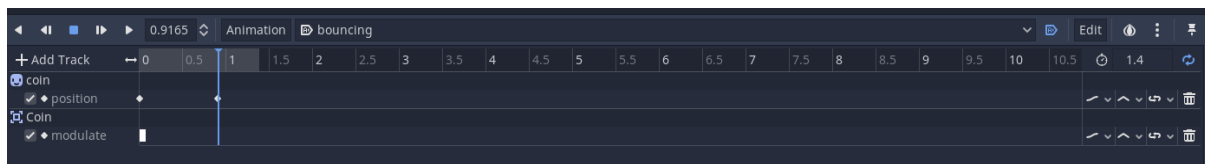


schimbam animatia specifica coinului

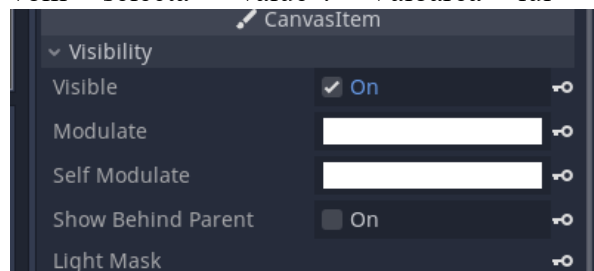
- Uitându-ne în secțiunea animației, observăm că a apărut un punct, specific unei key. Dacă dăm click pe el, în partea din dreapta a ecranului vedem următoarele opțiuni

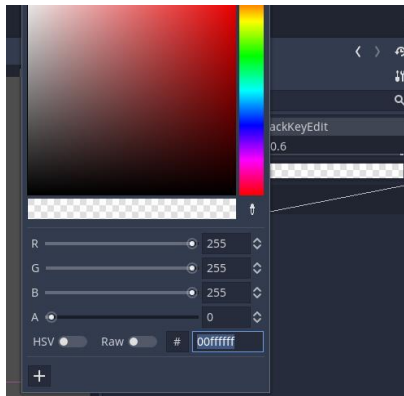


- Pentru primul punct vom avea mereu x și y vor avea valoarea 0. Însă pentru următoarele key vom seta noi valoarea dorită. În cazul de fata, x= 0, y= -10



- fade-out: aceasta animație este folosită pentru dispariția dreaptă a coinului
- La fel ca la animația anterioară, vom adăuga cheie acolo unde dorim să se întâmple modificări asupra acestei animații. Pentru că modificarea pe care dorim să o facem aici referă la mișcare, cheia pe care o dorim să o adăugăm nu mai este aceeași cu cea de sus. Mergem la “coin” și în partea din dreapta la visibility, vom adăuga cheia de la “Modulate”
 - După ce am adăugat două cheie, mergem pe cea din dreapta, iar în partea din dreapta vom selecta “value”. Valoarea lui “A” o vom duce la minim.





La fel ca playerul și coinurile noastre au nevoie de partea de script. Adăugăm un nou script și scriem codul necesar.

```

1  extends Area2D
2
3  onready var anim_player : AnimationPlayer = get_node("AnimationPlayer")
4  # devine o referinta pentru nodul nostru din scene tree
5
6  export var score: = 100
7
8  func _on_body_entered(body: Node) -> void:
9      >| picked()
10     >| anim_player.play("fade_out")
11     >|
12     >|
13  func picked() -> void:
14     >| PlayerData.score += score
15     >|
16

```

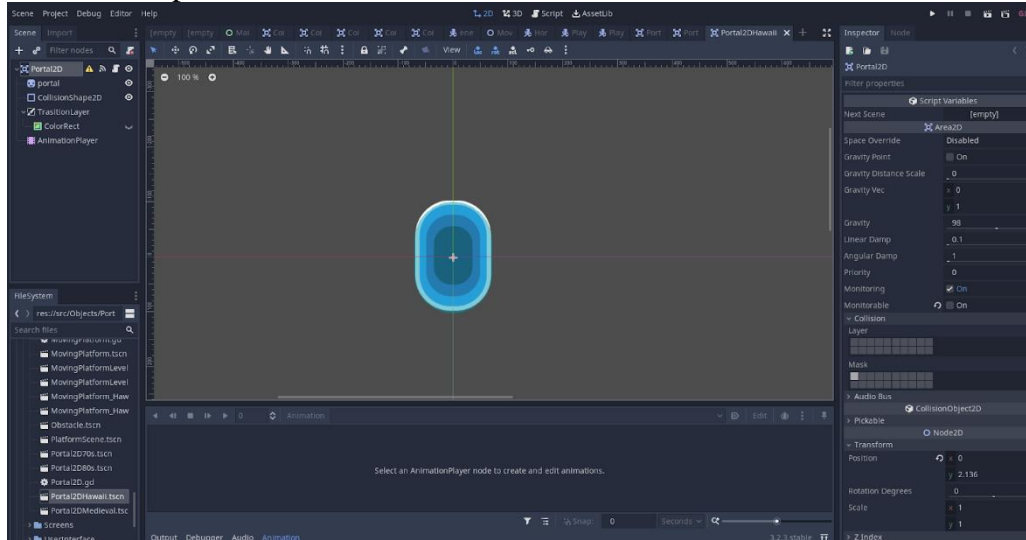
V. Creare Portal

Pentru ca jucătorul să poată avansa de la un nivel la altu, pe parcursul tuturor nivelelor se vor găsi portale.

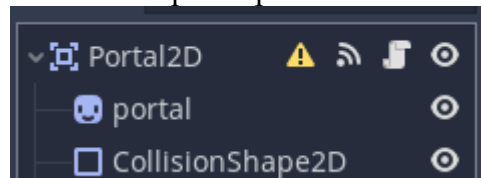
Pentru crearea lor am făcut următorii pași:

1. Am creat o nouă scena:
Scene -> New scene -> Custom Node-> Area2D (Coin)
2. În funcție de nivelul pe care l-am creat, am adăugat png-urile dorite prin drag&drop.

3. La fel ca playerul și coin-urile vor avea nevoie de un CollisionShape2D.

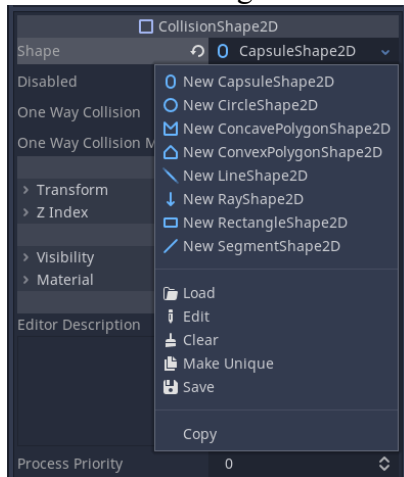


click dreapta pe "Coin" -> New Node -> CollisionShape2D



Pentru ca sa acoperim o cat mai mare parte din coin, de la opțiunea "shape" alegem forma pe care o dorim

4. Vom adăuga un Transaction Layer, ca un nod nou, urmand sa adaugam un "ColorRect".

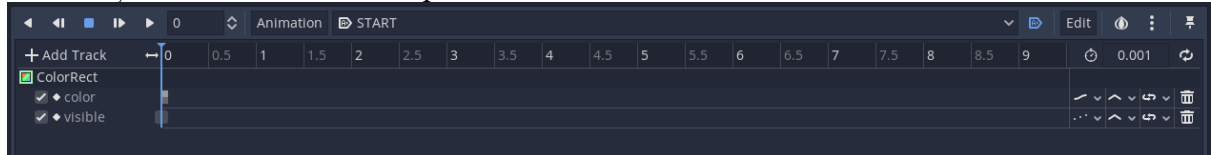


Mergem la "ColorRect" și în partea din dreapta la visibility, vom selecta



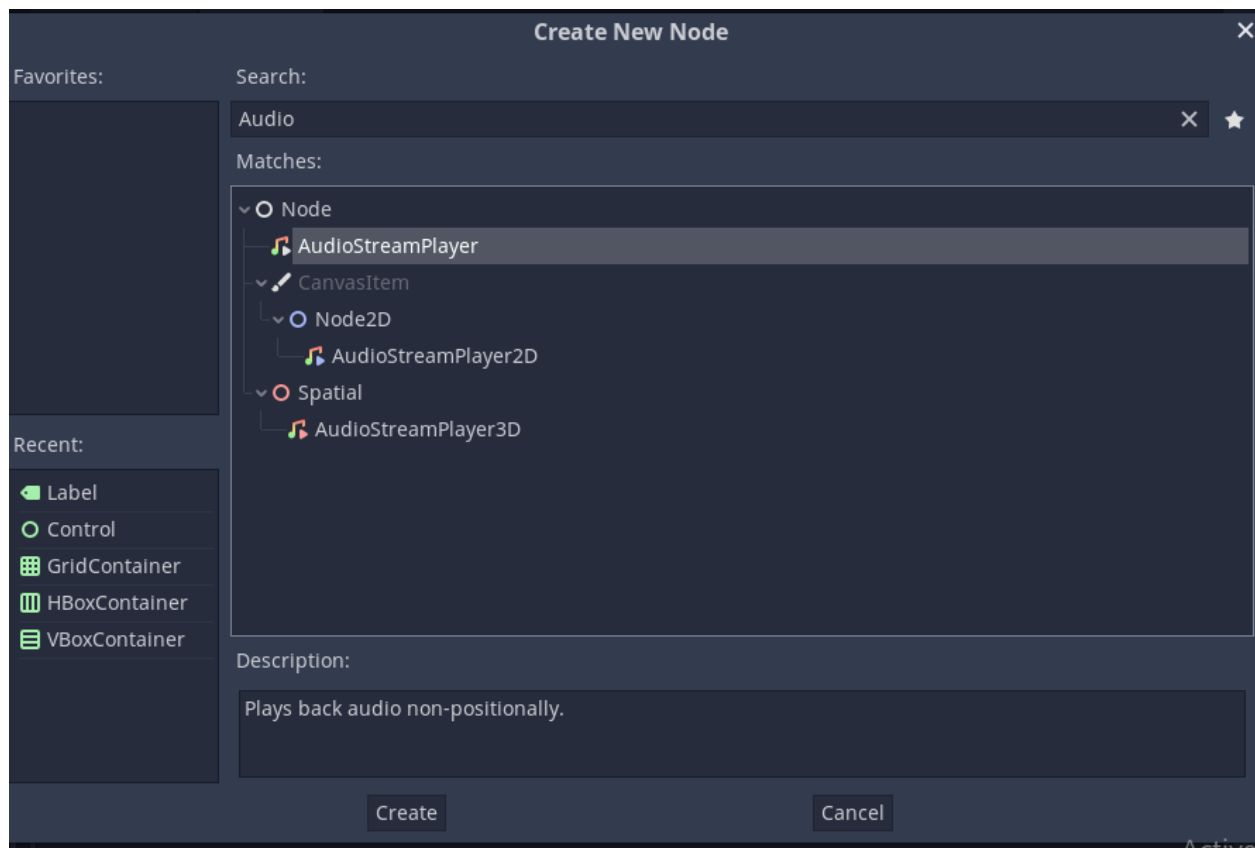
"Modulate" și vom duce "A"ul la valoarea minima.

5. Deoarece scopul portalului este acela de a trece de la un nivel la altu, vom adăuga, la fel ca la coin, o animație care face acest lucru posibil.



VI. Adaugare muzica

Pentru a adauga muzica, se va selecta nodul AudioStreamPlayer.



Sunetul dorit va trebui sa fie de tip .ogg, Godot nerecunoscand fisierele de tip .mp3.

VII. Testare

În Godot, mergem la AssetLib, vom descărca, iar apoi vom instala **GUT** - Godot Unit Testing(ne asigurăm ca avem toate căsuțele bifate) .

Mergem la ProjectsSettings, Plugins și vom activa GUT.

Cream o nouă scenă pe care o denumim TestRunner, iar apoi adugam nodul Gut pe care il redenumim corespunzător.

Cream un folder test, iar altul în interiorul lui numit unit, unde vom crea un scripturile pentru testare.

În Inspector ne asigurăm ca este bifata Run On Load. Pentru Directory1, vom atașa calea către folderul unit, creat anterior.

În scriptul **test_player.gd**, preluam în variabila player scripul Player.gd folosit, apoi ne creăm o funcție care va testa dacă PlayerData.deaths se va mari cu o unitate atunci cand Player-ul moare. Apelăm metoda die() din Player, apoi cu ajutorul funcției assert_eq([],[],[]) din Gut testăm dacă valoare obtinuta este egala cu cea așteptată.

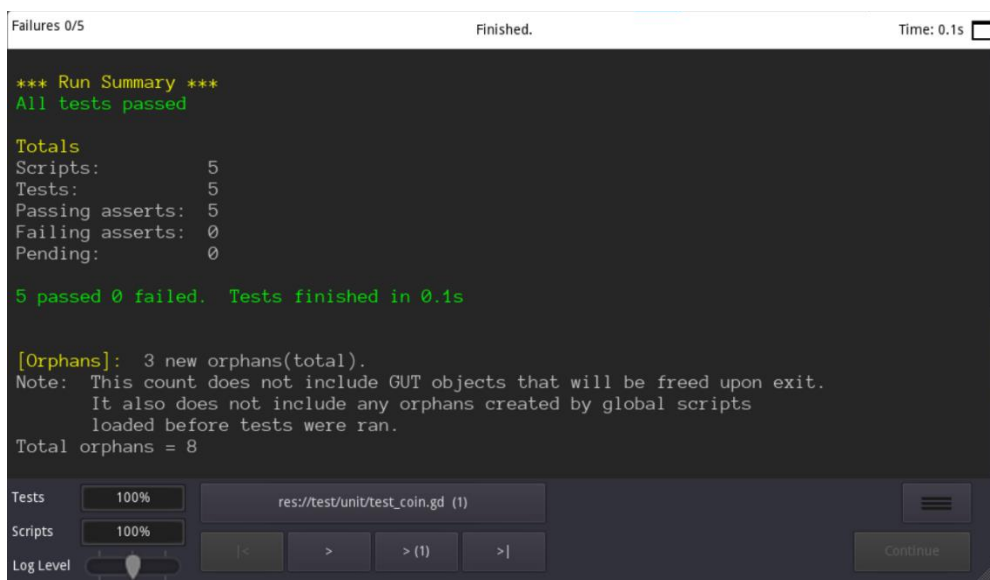
În scriptul **test_coin.gd**, preluam în variabila coin scripul Coin.gd folosit, apoi ne creăm o funcție care va testa dacă PlayerData.score se va mari cu o unitate atunci cand Player-ul colectioneaza un coin. Apelăm metoda picked() din Coin, apoi cu ajutorul funcției assert_eq([],[],[]) din Gut testăm.

În scriptul **test_enemy.gd**, preluam în variabila enemy scripul Enemy.gd folosit, apoi ne creăm o funcție care va testa dacă PlayerData.score se va mari cu o unitate atunci cand Player-ul ucide un Inamic. Apelăm metoda die() din Enemy, apoi cu ajutorul funcției assert_eq([],[],[]) din Gut testăm dacă valoare obtinuta este egala cu cea așteptată.

În scriptul **test_nodes_free.gd**, preluam în variabila autoFree scripul autofree.gd din Gut, apoi ne creăm o funcție care va testa dacă se eliberează după ștergerea unui nod. Cream un nod nou, il adaugam(copil al celui de mai sus), apoi apelăm metoda free_all(). Cu ajutorul funcției assert_freed([],[]) din Gut testăm dacă nodul adaugat a fost șters.

În scriptul **test_emitted_signal.gd**, preluam în variabila signalWatcher scripul signal_watcher.gd din Gut, apoi ne creăm o funcție care va testa dacă semnalul a fost emis. Cream o clasa nouă SignalObject, pentru semnale fără parametrii, care va emite semnalul, apoi apelăm metoda get_emit_count pentru signalWatcher si verificam daca a fost emis.

Dăm play și vom vedea ca a trecut toate testele(print-screen mai jos).



The screenshot shows the Godot test runner interface. At the top, it says "Failures 0/5", "Finished.", and "Time: 0.1s". The main area displays a "Run Summary" indicating that all tests passed. Below this, a "Totals" section shows 5 scripts, 5 tests, 5 passing asserts, 0 failing asserts, and 0 pending tests. A green line indicates "5 passed 0 failed. Tests finished in 0.1s". At the bottom, there is a section for "[Orphans]" showing 3 new orphans and a note about GUT objects. The bottom bar shows "Tests" and "Scripts" both at 100%, a file path "res://test/unit/test_coin.gd (1)", and a "Continue" button.

```
Failures 0/5 Finished. Time: 0.1s

*** Run Summary ***
All tests passed

Totals
Scripts:      5
Tests:        5
Passing asserts: 5
Failing asserts: 0
Pending:      0

5 passed 0 failed. Tests finished in 0.1s

[Orphans]: 3 new orphans(total).
Note: This count does not include GUT objects that will be freed upon exit.
It also does not include any orphans created by global scripts
loaded before tests were ran.
Total orphans = 8

Tests 100% res://test/unit/test_coin.gd (1)
Scripts 100%
Log Level < > > (1) > | Continue
```