**Documentatie Proiect**

**Cuprins:**

[**I.Creare nivel**](#Nivel)

[**II.Creare inamic care arunca flacari**](#inamic)

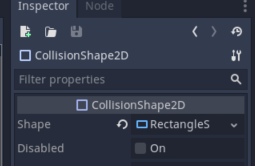
[**III.Creare calaret care alearga pe cal (creare inamic)**](#inamic2)

[**IV.Creare Coin**](#coin)

[**V.Creare Portal**](#portal)

[**VI. Adaugare Muzica**](#muzica)

[**VII.Testare**](#testare)

**I.Creare nivel**

1. Creare player

* Other node-> KinematicBody2D[[1]](#footnote-1) -> se va numi Player + adăugare detector de coliziune (CollisionShape2D)
* Forma detectorului: Inspector -> Shape -> Rectangle
* Adăugare imaginea(Sprite[[2]](#footnote-2)) playerului cu drag and drop player.png

Atașarea celor 2 scripturi:

1. Player.gd (clasa jucătorului): Add -> Language GDScript, Inherits Actor, Empty Template
2. 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**(delta[[3]](#footnote-3): 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: Vector2[[4]](#footnote-4) **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)

-> 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

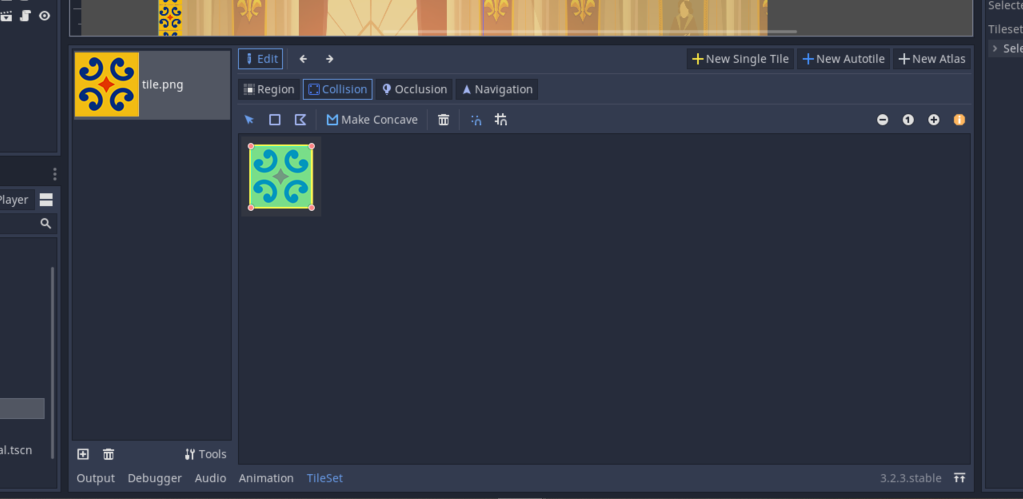
* **export** var gravity

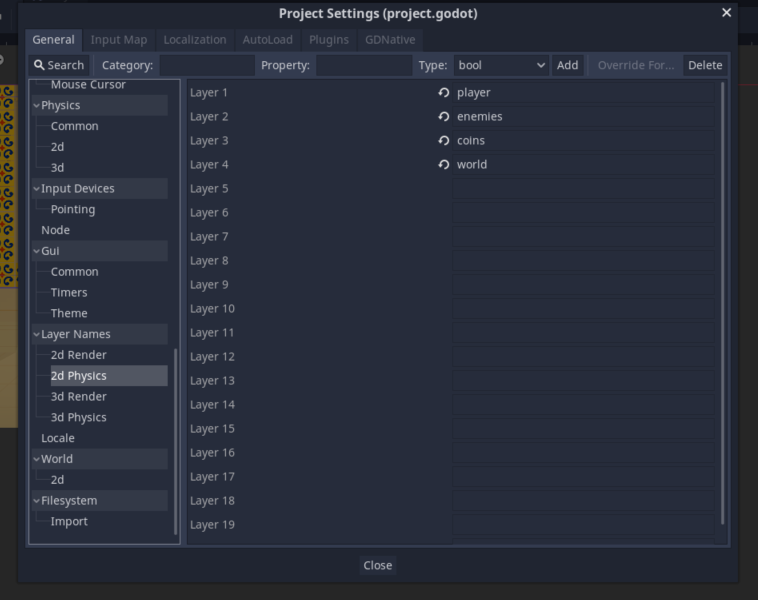
-> i se poate schimba valoarea din Inspector-> Categoria Script Variabiles

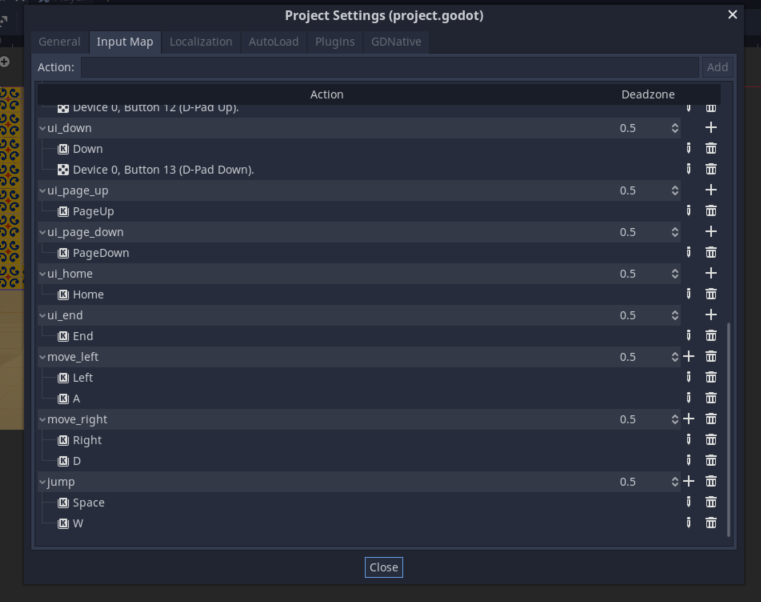
* var speed =Vector2(300.0, 1000.0)

-> viteză maximă pe Ox și pe Oy

1. 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 -> Layer[[5]](#footnote-5)(selectat doar pentru world) -> Mask deselctez tot)
* Personalizare taste de input: Project Settings -> Input Map

1. 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 if[[6]](#footnote-6) 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

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

1. Creare inamic

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

1. 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

* volcity.y= **move\_and\_slide**(\_velocity, FLOOR\_NORMAL).y

-> nu se mai oprește când se lovește de un perete

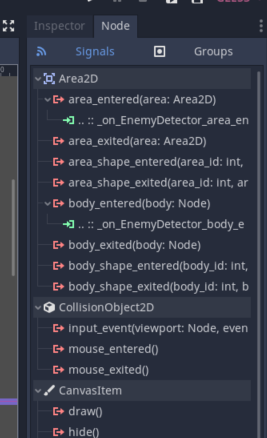
* Optimizare inamic:

-> inamicul nu se mișcă cand e departe de player

-> creare nod VisibilityEnabler2D[[7]](#footnote-7) -> Inspector-> activez Physics Process Parent (când iese din vizor se va opri) și când va trece de player nu se mai întoarce

1. Creare urmărire player (Camera)

* Player-> Nodul Camera2D[[8]](#footnote-8) -> Inspector -> Current -> On => camera va urmări playerul

1. Creare abilitate de a sări pe inamici și de a-i omorî

* Lui Enemy i se adaugă nodul Area2D[[9]](#footnote-9) -> 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
* functia de semnal func **\_on\_StompDetector\_body\_entered** (body: PhysicsBody2D) -> void:

-> inamicul o să fie omorât în momentul în care playerul îi sare pe cap

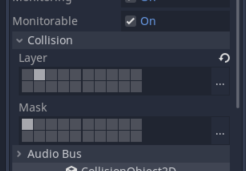
* if body.global\_position.y > **get\_node** ("StompDetector"). global\_position.y:

-> 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



1. Inamicul poate ucide jucătorul:

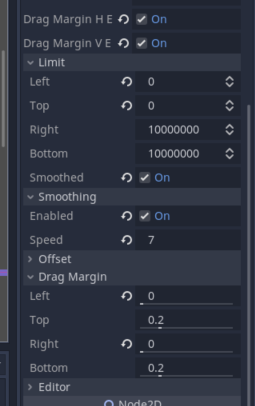
-> în Player se creează un detector ca să-l omoare dacă se întâlnește cu inamicul:

* 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

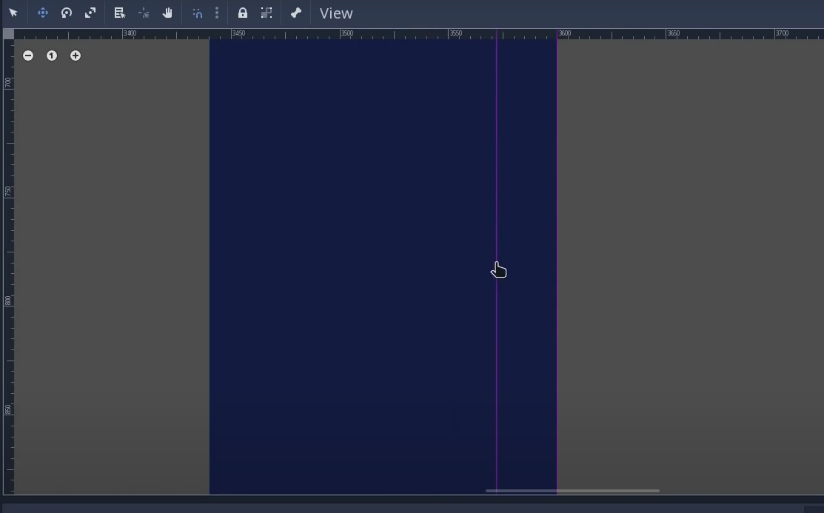
-> stomp\_impulse îl face să difere față de o săritura normală  
-> aici se pot face modificări în cum să sară (la diferite înălțimi etc.)

* func **\_on\_EnemyDetector\_body\_entered**(body) ->void:

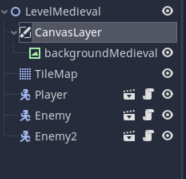
-> se crează un nou semnal pentru abilitatea jucătorului de a muri

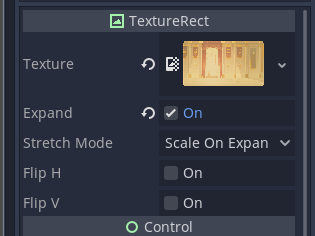
1. Modificări ale camerei ca să se miște mai usor:

* 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)
* Oprire cameră la sfarșitul nivelului

->Rigla se mută pana 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 campul Right

1. Adaugare fundal

* Assets -> click and drag pe tablă -> drag & drop + Alt => meniu -> TextureRect -> fundalul trebuie să fie un copil al nivelului
* Inspector -> Expand -> On
* Layout(din Toolbar) -> full rect -> umple toat *ecranul*
* Ca imaginea de background să poată sta tot jocul fix in fundal-> nodul CanvasLayer[[10]](#footnote-10) (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

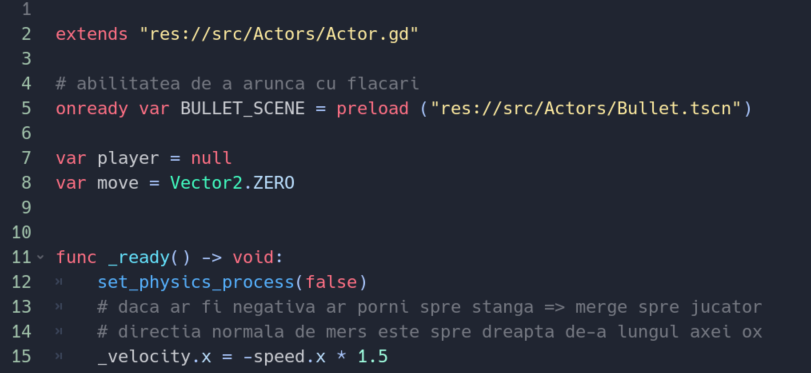
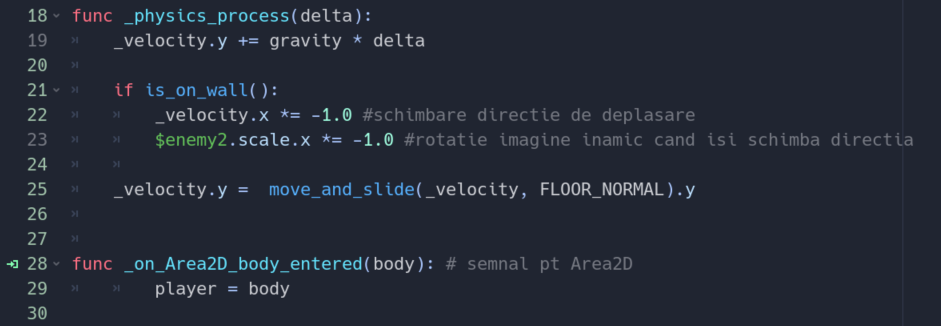
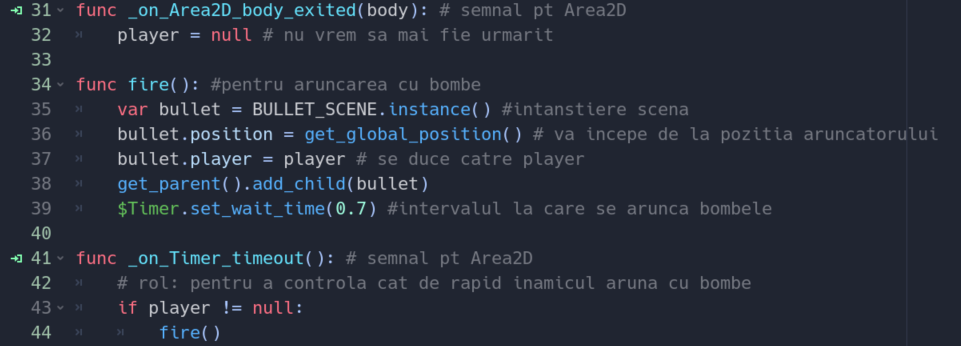
- rol: va declansa automat aruncarea bombelor atunci cand playerul intra in raza acesteia

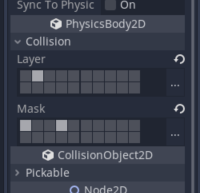
* Adaugare semnale:

-\_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)

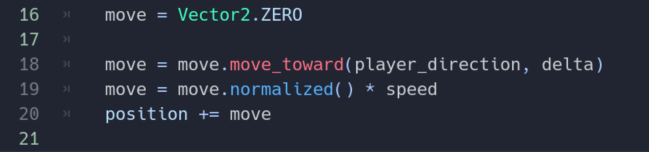


2. Creare nod nou ce va reprezenta o flacara

* KinematicBody2D -> se va numi Bullet
* Adaugare script: Bullet.gd (va exitinde 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** )

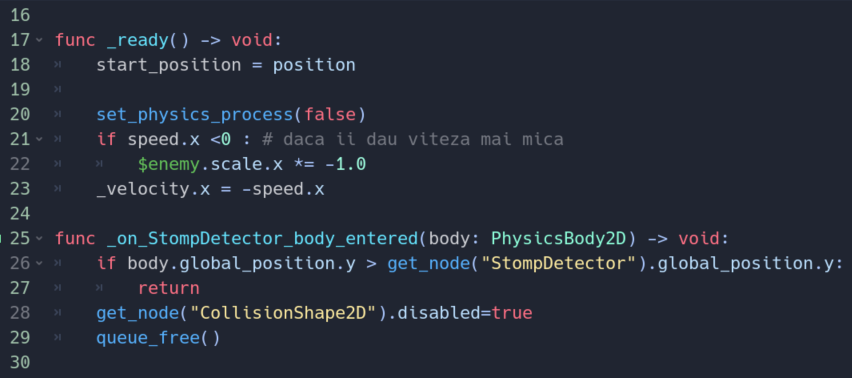
Moves the vector toward **to** by the fixed **delta** amount.

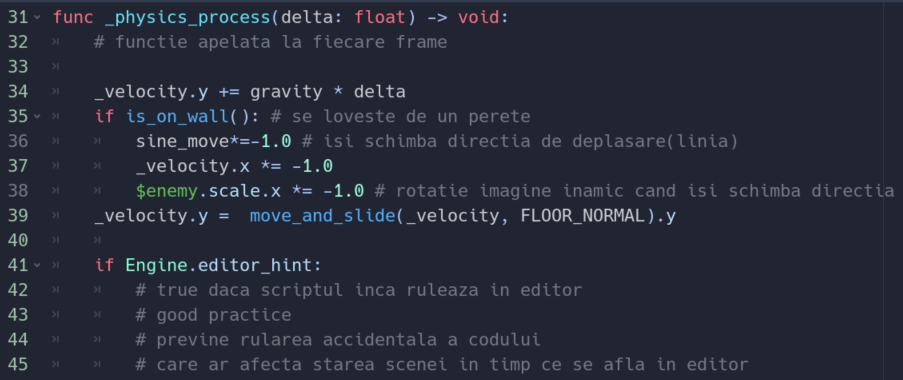
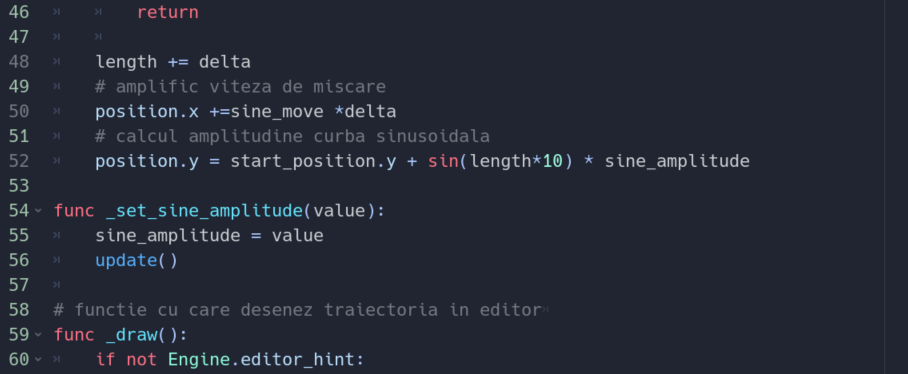
* + Vector2 normalized ( )

Returns the vector scaled to unit length. Equivalent to v / v.length().

**III.Creare calaret care alearga pe cal (creare inamic sub forma de tool[[11]](#footnote-11) in editor):**

1. Creare nod nou inamic

* Duplicare inamic de tipul 1
* Maresc VisibiltyEnabler2D 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.



**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 forma.

Nivelul 1/2/3: Diamante

Nivelul 4/5/6: Floricele

Nivelul 7/8/9: Stelute

Nivelul 10: Ananasi

Nivelul 11: Ananasi

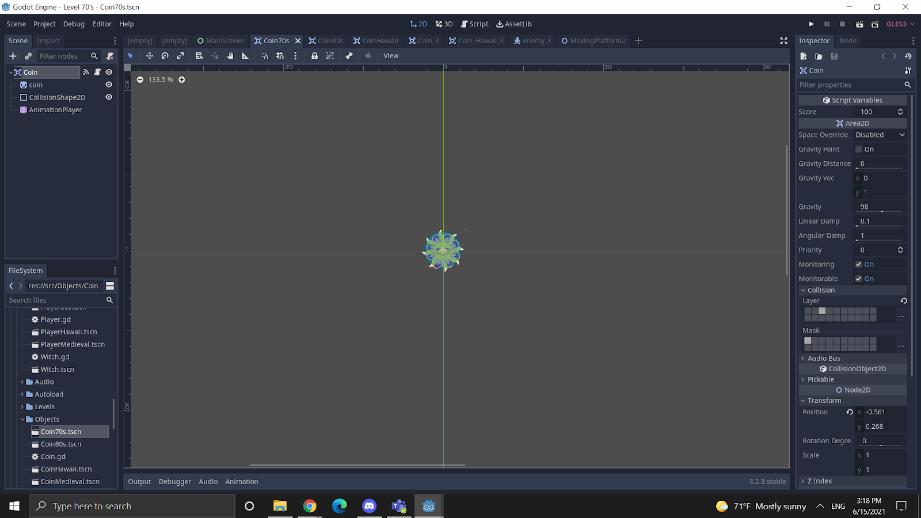
Nivelul 12: Pesti

Pentru crearea lor am făcut următorii pași:

1. Am creat o nouă scena:

Scene -> New scene -> Custom Node-> Area2D (Coin)

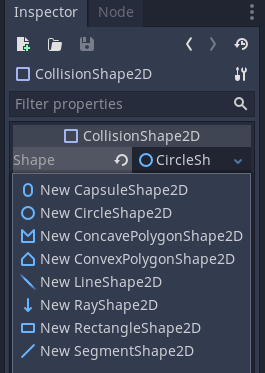
1. În funcție de nivelul pe care l-am creat, am adaugat png-urile dorite prin drag&drop.

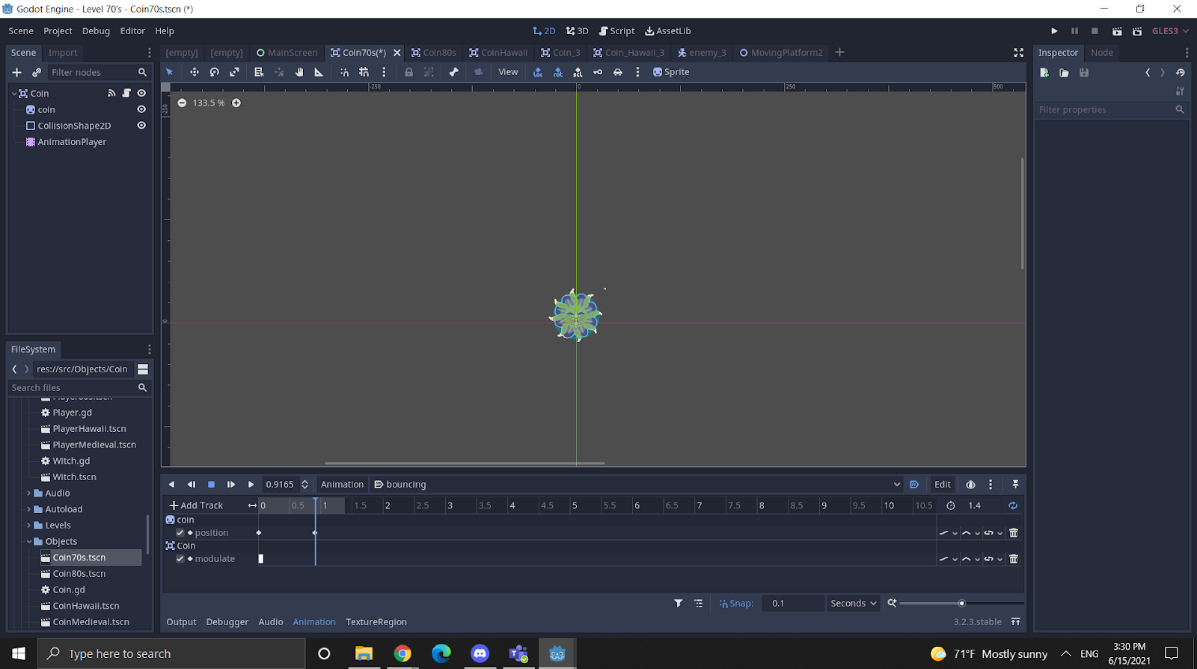


1. La fel ca playerul și coin-urile vor avea nevoie de un CollisionShape2D.

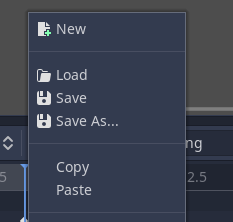
click dreapta pe “Coin” -> New Node -> CollisionShape2D

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 dispare treptat atunci cand am fost capturat am făcut următoarele setări:

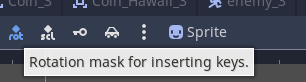
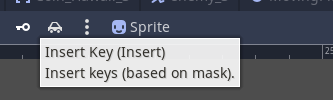
click dreapta pe “Coin” -> New Node -> AnimationPlayer

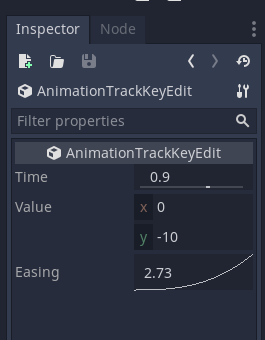
În partea de jos a paginii vom da click pe “Animation” și după aceea vom alege opțiunea “new”, astfel vom creea o animație.



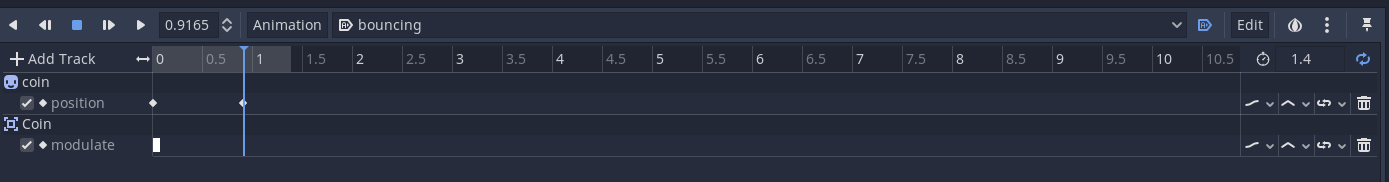
Vom avea doua astfel de animatii:

1. bouncing

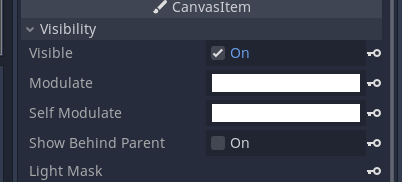
* vom folosi căsuța din dreapta butonului de “ Add track”  unde vom seta durata de timp in care dorim sa se realizeze animatia
* în partea de sus al ecranului avem butonul specific rotației, pe care il vom selecta
* langa acest buton se afla cel de insert key, aceasta cheie se pune acolo unde dorim sa schimbam animatia specifica coinului
* Uitandu-ne în secțiunea animației, observam ca a apărut un punct, specific unei key. Dacă dăm click pe el, în partea din dreapta a ecranului vedem următoarele opțiuni

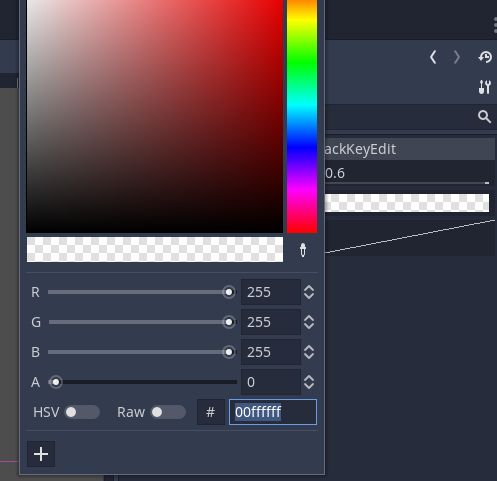


* Pentru primul punct vom avea mereu x și y vor avea valoarea 0. Însă pentru următoarele key vom seta noi valoarea dorită. În cazul de fata, x= 0, y= -10

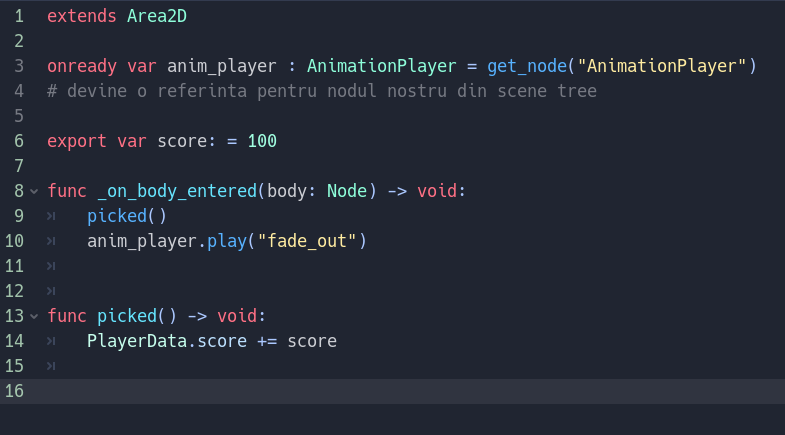


1. fade-out: aceasta animație este folosită pentru dispariția dreaptă a coinului

* La fel ca la animatia anterioara, vom adăuga cheiei acolo unde dorim sa se intample modificări asupra acestei animatii. Pentru ca modificarea pe care dorim sa o facem aici referă la mișcare, cheia pe care o dorim sa 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”
* Dupa ce am adugat doua cheiei, mergem pe cea din dreapta, iar in 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.



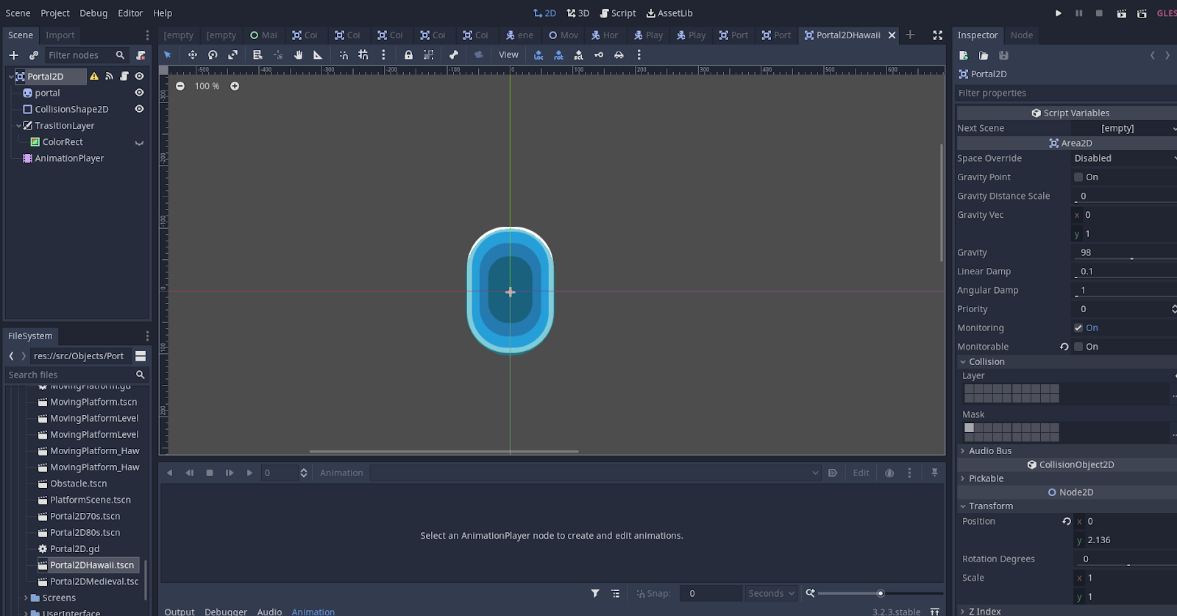
**V. Creare Portal**

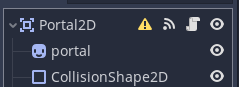
Pentru ca jucătorul să poată avansa de la un nivel la altu, pe parcursul tuturor nivelelor se vor găsi portale.

Pentru crearea lor am făcut următorii pași:

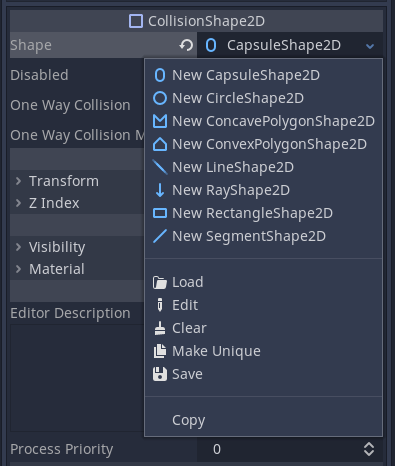
1. Am creat o nouă scena:

Scene -> New scene -> Custom Node-> Area2D (Coin)

1. În funcție de nivelul pe care l-am creat, am adaugat png-urile dorite prin drag&drop.
2. La fel ca playerul și coin-urile vor avea nevoie de un CollisionShape2D. 

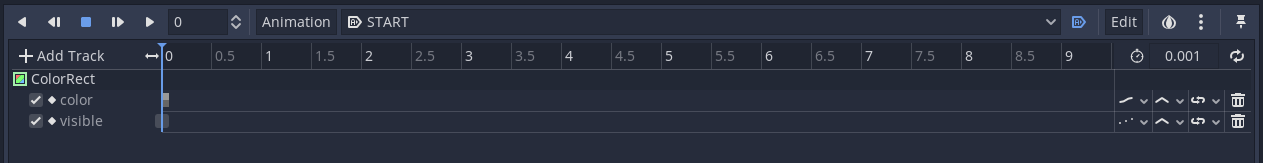
click dreapta pe “Coin” -> New Node -> CollisionShape2D

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

1. Vom adăuga un Transaction Layer, ca un nod nou, urmand sa adaugam un “ColorRect”.

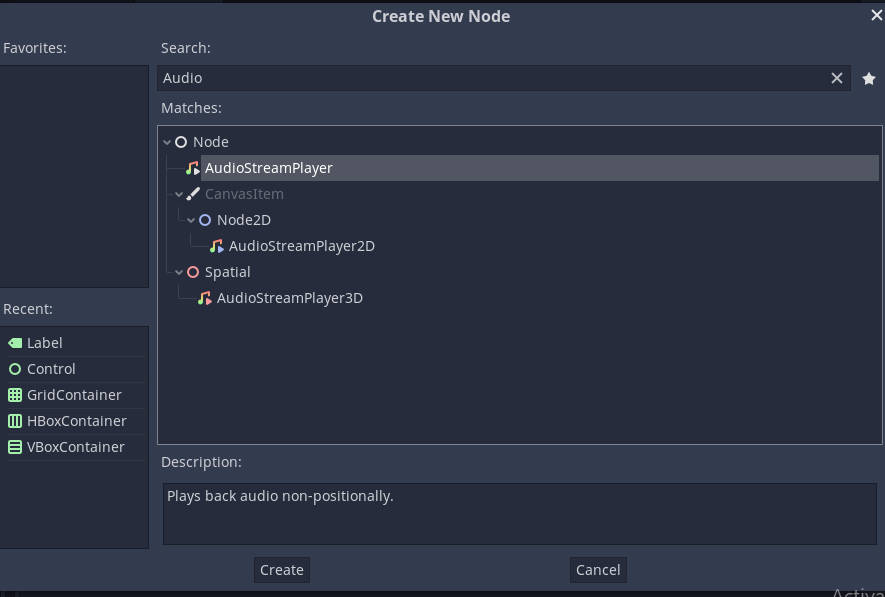
Mergem la “ColorRect” și în partea din dreapta la visibility, vom selecta “Modulate” și vom duce “A”ul la valoarea minima. 

1. 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.

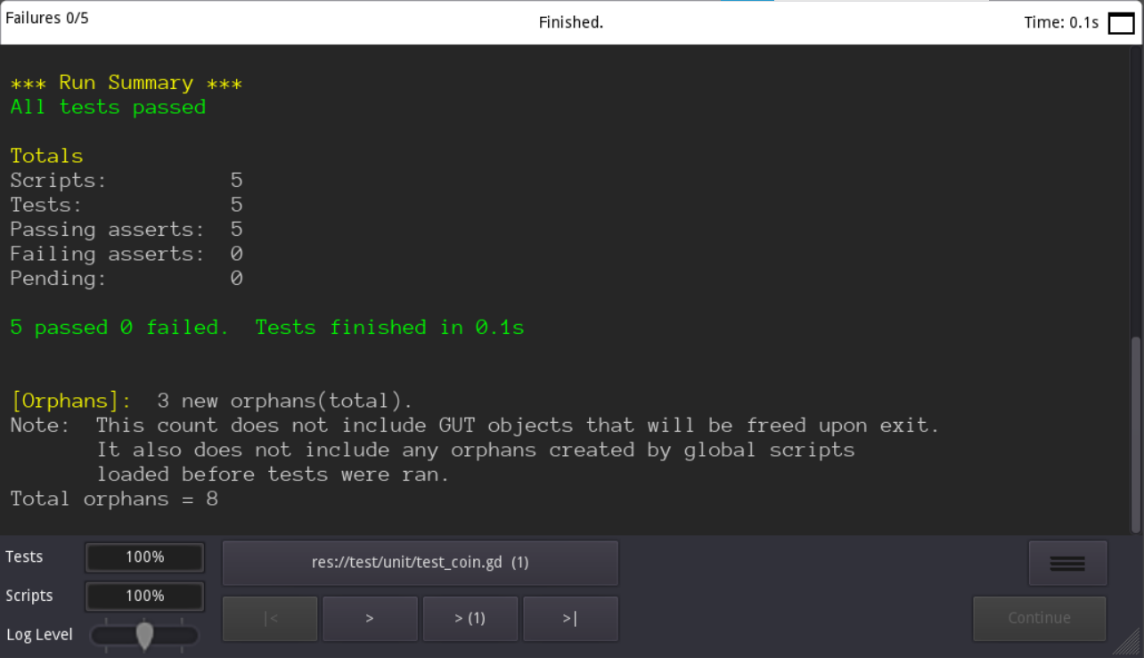
În scriptul **test\_coin.gd**, preluam în variabila coin scripul Coin.gd folosit, apoi ne creăm o funcție care va testa dacă PlayerData.score se va mari cu o unitate atunci cand Player-ul colectioneaza un coin. Apelăm metoda picked() din Coin, apoi cu ajutorul funcției assert\_eq([],[],[]) din Gut testăm.

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).



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. [↑](#footnote-ref-1)
2. Un nod care afișează o textură 2D. [↑](#footnote-ref-2)
3. Valoare dată de Godot care reprezintă timpul scurs de la ultimul frame. [↑](#footnote-ref-3)
4. Structură cu 2 elemente care poate fi utilizată pentru a reprezenta poziții în spațiul 2D sau orice altă pereche de valori numerice. [↑](#footnote-ref-4)
5. Dacă este nevoie de o listă mare și / sau dinamică de excepții este eficient să se foloseasca sistemul de strat(Layer – de ce se lovește) / mască de coliziune(Mask - ce ignoră). [↑](#footnote-ref-5)
6. Operator ternar. [↑](#footnote-ref-6)
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. [↑](#footnote-ref-7)
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. [↑](#footnote-ref-8)
9. Zona 2D care detectează nodurile CollisionObject2D care se suprapun, intră sau iese. Poate, de asemenea, modifica sau suprascrie parametrii locali de fizică (gravitație, amortizare). [↑](#footnote-ref-9)
10. Stratul de desen pe pânză. Nodurile CanvasItem care sunt copii direcți sau indirecți 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. [↑](#footnote-ref-10)
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. [↑](#footnote-ref-11)