IZRADA ARKADNE 3D VIDEOIGRICE

Radoš, Ivona

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:704096

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Repository of University Department of Professional Studies

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Ivona Radoš

Z A V R Š N I R A D

Izrada arkadne 3D videoigrice

Split, rujan 2021.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Objektno programiranje

Z A V R Š N I R A D

Kandidat: Ivona Radoš

Naslov rada: Izrada arkadne 3D videoigrice

Mentor: Ljiljana Despalatović, viši pred.

Split, rujan 2021.

Sadržaj

Sažetak ... 1
1 Uvod ... 3
2 Godot ... 4
2.1 Godot Editor ... 5
2.2 Čvorovi i scene ... 7
2.3 Instanciranje ... 8
2.4 Signali ... 10
2.5 SilentWolf plug-in ... 13
3 Implementacija igre ... 14
3.1 Scene u 2D formatu ... 14
3.1.1 Control ... 14
3.1.2 YouWin ... 17
3.1.3 SubmitScore ... 19
3.1.4 NewScore ... 21
3.1.5 GameOver ... 24
3.2 Scene u 3D formatu ... 24
3.2.1 Izrada poligona, gridMape i kamere ... 25
3.2.2 Implementacija lika ... 31
3.2.3 Implementacija neprijatelja ... 36
3.2.4 Implementacija novčića ... 42
3.2.5 Implementacija života ... 46
3.2.6 Control ... 50
4 Zaključak ... 59

Sažetak

Cilj ovog završnog rada je izrada funkcionalne trodimenzionalne videoigre, koristeći Godot pogonski alat (engl. game engine), namijenjenu za jednu osobu u kojoj korisnik upravlja glavnim likom videoigre te pokušava svladati prepreke kako bi glavni lik igre ostao živ određeno vrijeme.

Za izradu PacMan videoigre koristio se Godot pogonski alat i programski jezik GDScript (najmjenjen samo za Godot), dosta sličan Pythonu.

Videoigra je arkadni tip igre u kojem igrač mora hodajući kroz labirint skupiti određeni broj novčića prije nego što mu istekne vrijeme. U isto vrijeme mora izbjeći neprijatelje, gdje u suprotnom gubi jedan život. Isto tako na određenim nivoima postoje životi koje igrač može skupiti kako bi imao veće šanse za pobjedu. Igrica se sastoji od 5 nivoa gdje će se na kraju, ako igrač uspješno završi igricu, zbrojiti svi bodovi i svrstati na ljestvicu sa konačnim rezultatima. Broj bodova ovisi o vremenu u kojem smo prešli svaki nivo i o broju života koji su nam ostali.

Ključne riječi: Godot, GdScript, videoigra

Summary

Developing an arcade 3D videogame

The goal of this final paper is to create a functional three-dimensional game for one person, in which the player controls the main object and tries to overcome barriers so that the main character stays alive certain amount of time.

For building the game PacMan, the Godot game engine and programing language GDScript (used only in Godot) were used. It is a programing language similar to Python.

It is an arcade type of game where the player is walking through a maze and has to collect a certain amount of coins before the time runs out. At the same time, he has to avoid enemies, where he will lose lives. He can also collect lives at certain levels so that the player can have a bigger chance of winning. The game consists of 5 levels where if the player finishes the game successfully, his final score will then be saved on the scoreboard. The score is determined based on the remaining time in which the player completed every level and number of lives.

Keywords: Godot, GDScript, videogame

1 Uvod

U radu je obrađena izrada 3D igre u Godotu. Motivacija za rad je uvod u granu informacijskih tehnologija (IT) koja se bavi izradom igara pomoću modernog pogonskog alata za izradu igara (engl. game engine). S mnoštvo sustava koji omogućavaju bolju sliku i vizualno upravljanje kôdom, stvaranje videoigre je postalo jednostavnije i naprednije.

Videoigra se sastoji od različitih područja stvaranja (programiranje, grafički dizajn, animacije, zvuk itd.), tako da je najčešće slučaj da grupa ljudi sa svim potrebnim sposobnostima radi na stvaranju videoigre.

U prvom poglavlju govorit će se o Godot okruženju, na koji način funkcionira i neke osnovne funkcije koje nam Godot nudi. Također će se moći primijetiti koliko je zapravo jednostavan pogonski alat i zašto je igrica tako lagana za shvatiti.

U drugom poglavlju će se objasniti sami način izrade igre. Način na koji je igrica strukturirana, koje scene sve imaju, za što koja služi i njihove skripte. Nadalje, što sami kôd obavlja i koja je zadaća koje scene. Također će se detaljno objasniti implementacija lika i ostalih bitnih objekata u igrici koji su svi međusobno povezani kroz različite scene. Isto tako su navedena rješenja koja su izradile druge osobe, a koja su korištena u ovom radu. Na samom kraju je naveden zaključak.

2 Godot

Godot pogonski alat služi za stvaranje 2D i 3D igara u jedinstvenom okruženju. Sadrži opsežni paket alata, zahvaljujući kojima se korisnik može koncentrirati na stvaranje igara bez nepotrebnih stvari. Igre se mogu izvesti na mnogo različitih platformi, uključujući glavne stolna računala (Linux, macOS, Windows), kao i mobilne (Android, iOS) i web (HTML5) platforme.

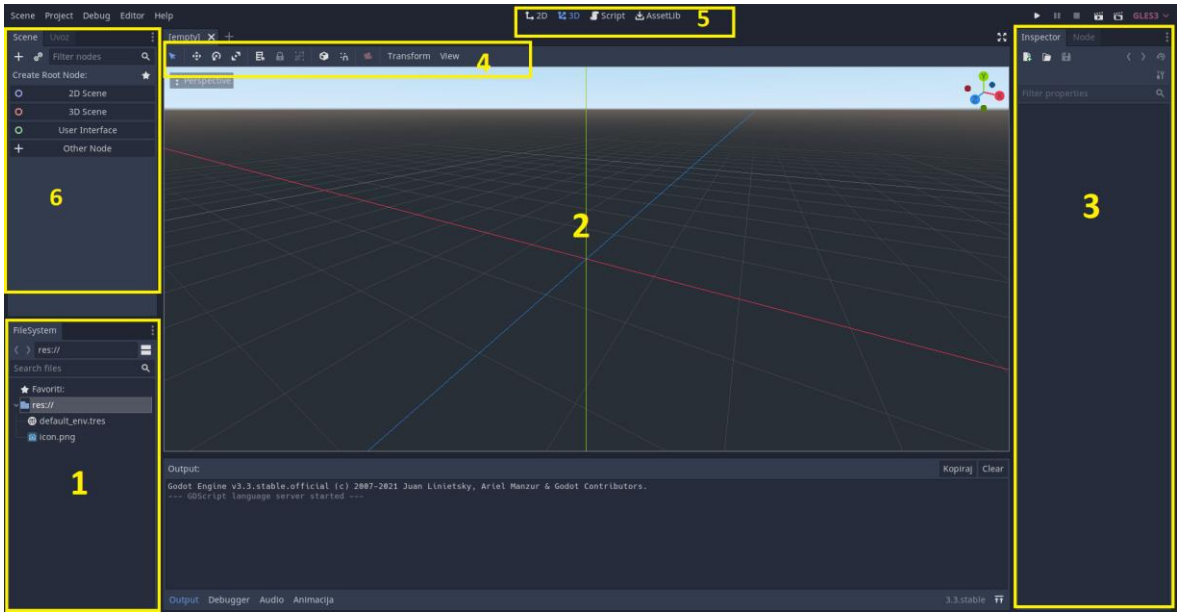
Godot je potpuno slobodan pogonski alat koji potpada pod dozvolu za MIT. Igre pripadaju onima koji su ih stvorili, uključujući sve retke kôda pogonskog alata. Razvoj Godota potpuno je neovisan i vođen zajednicom, prilagodljiv je korisnicima i na taj način omogućava jednostavnije korištenje.

GDScript je glavni jezik Godota. U usporedbi s drugim jezicima vrlo je jednostavan i prilagodljiv što je velika prednost, također je korisnicima jezika kao što su Python, Lua i Squirrel jako pristupačan. Isto tako je omogućeno jednostavno automatsko dovršavanje kôda za čvorove, signale i ostale elemente scene koji se uređuje. U njega su ugrađeni vektorski tipovi (poput vektora, transformacija itd.), šta olakšava primjenu linearne algebre u Godotu. Njegova dinamičnost olakšava optimizaciju dijelova koda u C ++-u (putem GDNative-a) kada su potrebne veće performanse, sve bez ponovnog sastavljanja kôda.

Godot trenutno podržava četiri programska jezika i jako je fleksibilan. Međutim, „glavni“ jezici su GdScript i VisualScript. Razlog odabira ova dva jezika je njihova neprimjetna integracija s Godotom i njegovim uređivačem skripti, dok C# i C++ treba uređivati u zasebnom IDE-u.

2.1 Godot Editor

Godot Editor je korisničko sučelje koje ima opcije stvaranja dvodimenzionalne i trodimenzionalne igre gdje se prilikom pokretanja Godot projekta automatski otvori trodimenzionalni svijet u kojeg se mogu dodavati objekti (čvorovi): svjetlost, atmosfera, vizualno prikazane objekte itd. (engl. nodes). Unutar njega uređuje se sama okolina tj. svijet u kojem se igra. Igra je prikazana kroz više scena, čvorova i mapa. Također se scena može sastojati od više scena pa se na taj način može od manjih dijelova sastaviti jedna veća cjelina ili čvor od više čvorova. Korisničko sučelje Godota je prikazano na slici 1, ispod koje je opis pojedinog polja editora. Gornji izbornik, radni prostor i botuni za pokretanje projekta prikazuju se na vrhu slijeva udesno.



Slika 1: Godot Editor korisničko sučelje

Prvo polje označava FileSystem gdje se može upravljati datotekama od projekta i resursima, pregledavati što se nalazi u kojoj datoteci, dodavati nove mape itd.. Drugo polje pokazuje izgled trenutne aktivne scene, tu stvaramo izgled igrice gdje je omogućeno jednostavno mijenjanje svojstva svakog oblika koji se nalazi u sceni (veličina, boja, oblik…). Omogućeno je dodavanje već postojećih scena gdje se lako pozicioniraju na željeno mjesto

i dodaju potrebne animacije. Najbitniji je dio editora jer se pomoću njega stvaraju svi vizualni aspekti igrice i određuje se način na koji će se igrica prikazati. Treće polje označava dio zvan Inspector koji omogućuje maloprije spomenuto uređivanje svojstava scene. Na četvrtom polju je alatna traka na kojoj se nalaze alati za premještanje, skaliranje ili zaključavanje objekata scene. Mijenja se pri prelasku na različite radne prostore i na slici 2 se može vidjeti izgled trake za 2D i 3D svijet.

Slika 2: 2D i 3D alatna traka

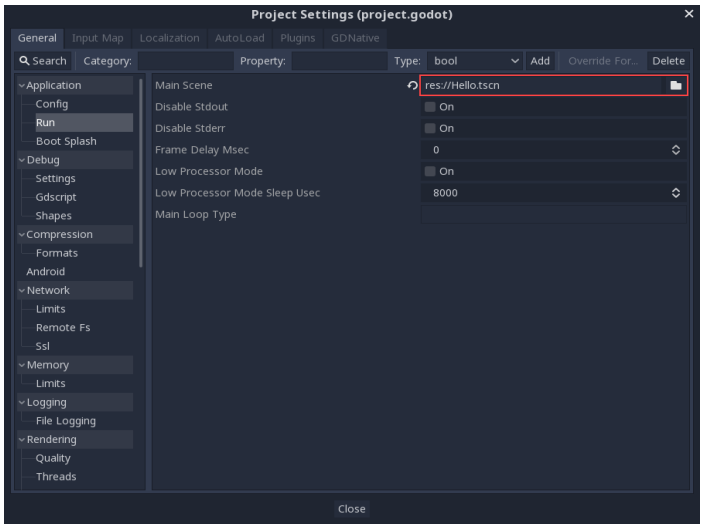
Polje pet prikazuje četiri botuna za promjenu radne površine: 2D, 3D, Script i AssetLib. 2D radni prostor upotrebljava se za sve vrste igara. Uz 2D igre, 2D radni prostor je mjesto na kojem se grade sučelja. U 3D radnom prostoru može se raditi sa gridom, svjetlima i razinama dizajna za 3D igre. Skripta je radni prostor za pisanje kôda gdje svaki objekt ili čvor može imati svoju skriptu sa svojim kôdom ili više objekta može biti povezano sa istom skriptom u slučaju da se naprave duplikati nekih objekata ili se povežu sa opcijom za povezivanje skripte na objekt koja se nalazi u Inspectoru.

Na šestom polju su prikazane početne opcije prilikom stvaranja nove scene gdje "2D Scene" dodaje Node2D čvor, "3D Scene" dodaje prostorni čvor, "User Interface" dodaje glavni čvor i "Other Node", što omogućuje odabir bilo kojeg čvora (dakle ekvivalentno je pritisku na "Add Child". Tu se također prikazuje trenutno aktivna scena zajedno sa njezinim scenama u slučaju da postoje.

2.2 Čvorovi i scene

Čvorovi su osnovni dio za stvaranje igre. Može izvoditi velik broj posebnih funkcija i svaki čvor uvijek ima sljedeća svojstva (attribute): ime, značajke za uređivanje, može se proširiti i drugim funkcijama, također se može dodati drugom čvoru kao njegovo dijete. Posebno važna osobina koju ima je nasljeđivanje. Kako čvorovi mogu imati i druge čvorove kao djecu ta ista djeca postanu dio glavnom čvora. Na primjer, čvor jabuka koja je u sceni stavljena na čvor stablo i dijete je tog istog čvora stablo će ostati na istom mjestu na stablu prilikom premještanja stabla na drugu poziciju jer je dio stabla. Takav raspored čvorova nazivamo "stablom". U Godotu je sposobnost grupiranja čvorova u stabla moćan alat za organizaciju projekta igre. Budući da različiti čvorovi mogu raditi različite funkcije, kombinirajući ih možemo stvoriti složenije ponašanje i na kraju cijelu igru.

Prizor se sastoji od skupine čvorova raspoređenih hijerarhijski (u strukturi stabla). Uz to, scena uvijek ima jedan korijenski čvor i njegova se instanca može stvoriti. Projekt može sadržavati nekoliko scena, ali da bi igra mogla započeti, jedna od scena mora biti označena kao takozvana glavna scena koja se učitava svaki put kad se započne projekt. Na slici 3 je prikazan način dodavanje određene scene (u ovom slučaju scena Hello.tscn) kao glavnu scenu, koji se dobije odabirom opcije Projekt -> Postavke projekta.

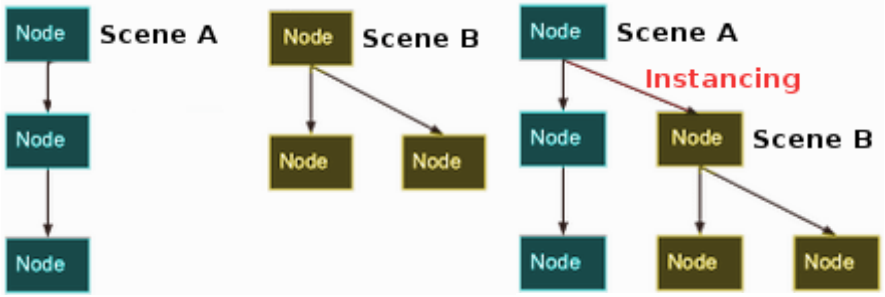


Slika 3: Određivanje glave scene

U osnovi, Godot Editor je urednik scene. Ima puno alata za uređivanje 2D i 3D scena i korisničkih sučelja, ali sam uređivač temelji se na konceptu uređivanja scene i čvorova koji čine scenu.

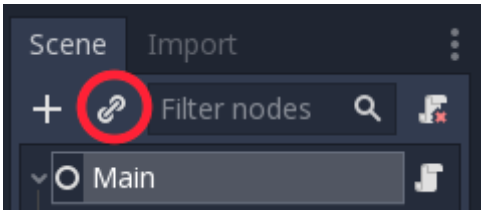
2.3 Instanciranje

Prilikom stvaranja scena dodaje se nastavak .tscn, također se može stvoriti bilo koji broj scena. Jednom kad je scena spremljena, može se umetnuti (instalirati, također: instancirati) u drugu scenu kao da je to bilo koji drugi čvor. Na donjoj slici 4, scena B dodana je sceni A kao primjer. To postignemo tako da otvorimo scenu A i izaberemo njezin korijenski čvor.



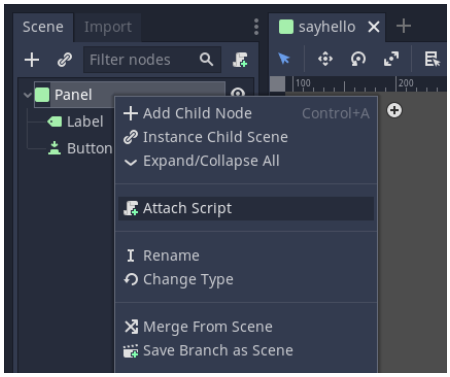
Slika 4: Instanciranje

Pritiskom na botun prikazanog na slici 5 odabire se file tj. Scena B i tako scena B djeluje sada u sceni A. Na primjer da je scena B lik igrice i njega kao scenu ubacujemo u glavnu scenu koja predstavlja level tj. poligon za taj nivo u kojem će lik djelovati.



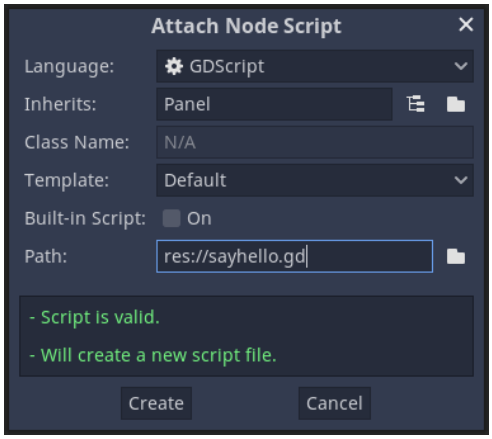
Slika 5: Postupak instanciranja

Također se može dodati skripta određenoj sceni ili čvoru, kako bi se mogle obavljati razne funkcije, ovisno o kôdu u skripti. Odabere se čvor na koji se želi dodati skripta i desnim klikom se otvore opcije za taj čvor, jedna od njih je da se spoji skripta na taj čvor (slika 6).



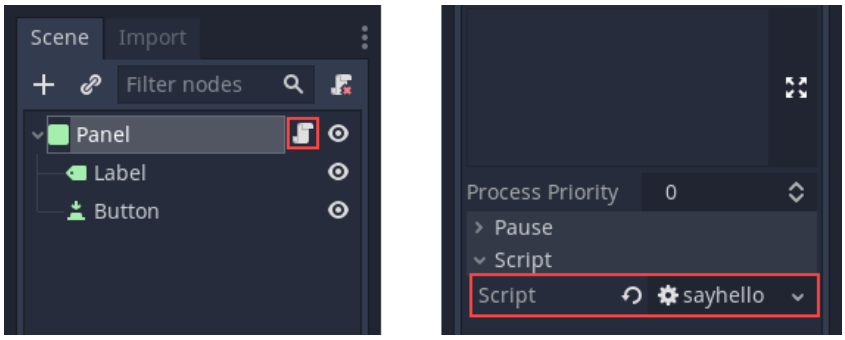
Slika 6: Dodavanje skripte

Pritom se otvara se dijaloški okvir za stvaranje skripte (slika 7). Ovaj dijaloški okvir omogućuje postavljanje jezika skripte, naziva klase i drugih relevantnih opcija. U GDScriptu datoteka sama predstavlja klasu, pa se polje naziva klase ne može uređivati. Čvor na koji se dodaje skripta je u ovom slučaju Panel, pa će se polje Nasljeđivanje automatski popuniti u "Panel" (engl. Inherits).



Slika 7: Dijaloški okvir za stvaranje skripte

Zatim se stvara skripta i dodaje se u čvor. To se može vidjeti kao ikona "Otvori skriptu" pored čvora na kartici Scene na slici 8 (lijevo) i također u svojstvima skripte u Inspektoru (desno).



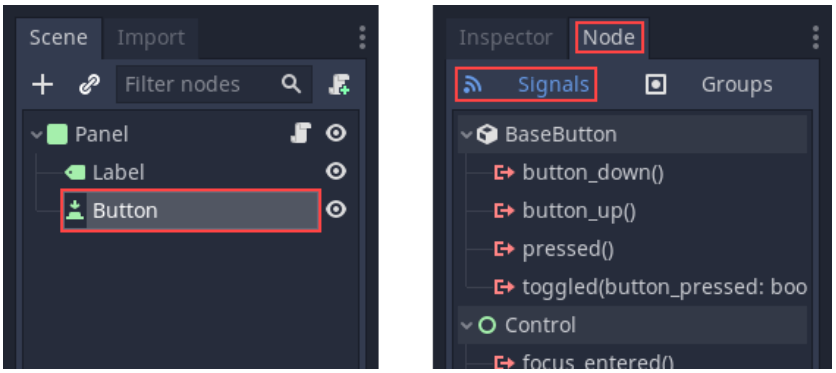
Slika 8: Oznake za skriptu

Funkcija `_ready` se poziva kada čvor i svi njegovi potomci uđu u aktivnu scenu.

2.4 Signali

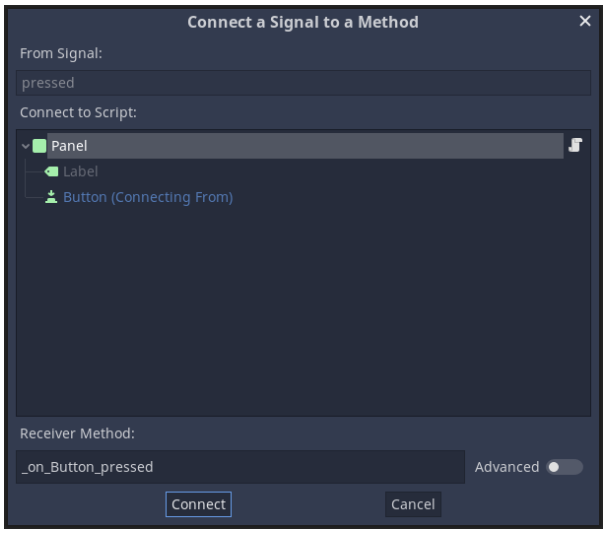
Signali omogućuju čvoru da pošalje poruku koju drugi čvorovi čuju i na koju mogu odgovoriti. Na primjer, umjesto da neprestano provjerava je li botun pritisnut, on može jednostavno poslati signal kada se pritisne, gdje će oni čvorovi koji su povezani sa tim botunom (čvorom) primiti taj isti signal i pokrenut će se funkcija koja je nastala prilikom stvaranja tog signala. Signali dovode do bolje organizacije i održavanja koda. Na jednostavan način se javlja funkciji da je došlo do promjene tj. da je korisnik pokrenuo neki zahtjev i funkcija može obaviti taj zahtjev primanjem tog signala.

Kada se stvori neki signal u isto vrijeme se stvori konekcija između ta dva čvora i nastaje funkcija koja se pokreće u trenutku kada se signal emitira. Preko primjera sa botunom u čvoru Panel će se objasniti način na koji se koriste signali. Odabere se čvor botuna na stablu scene (na slici 9, lijevo), a zatim odaberite karticu pokraj Inspectora, "Čvor". Također provjerite je li odabrano "Signali" (desno).



Slika 9: Stvaranje signala

Zatim, ako se odabere "pressed()" u odjeljku "BaseButton" i klikne botun "Poveži" u donjem desnom kutu, otvorit će se dijaloški okvir za stvaranje veze (slika 10).



Slika 10: Dijaloški okvir za stvaranje signala

Na vrhu dijaloškog okvira nalazi se popis čvorova u sceni s plavim nazivom čvora slanja. Ovdje se odabere čvor "Panel". Na dnu dijaloškog okvira nalazi se naziv metode koja će se stvoriti. Prema zadanim postavkama, naziv metode sadržavat će ime čvora izdavanja (u ovom slučaju, "Gumb"), što će rezultirati `_[EmitterNode]_[signal_name]`, gdje

će se u tu istu funkciju `_on_button_pressed` kasnije napisati kôd koji opisuje šta se radi kada se stisne na taj botun.

Također se mogu stvoriti vlastiti signali u Godotu na način da se deklarira u kôdu sa `signal my_signal` gdje je ključan naziv Signal koji određuje vrstu varijable, u ovom slučaju signal. Jednom stvoreni, signali se pojavljuju u inspektoru i mogu se povezati na isti način kao i ugrađeni signali čvora. Razlika je u tome što se funkcija koja je povezana na taj signal može pozvati samo kada se upotrijebi naredba/funkcija `emit_signal ("my_signal").`

```
extends Node2D

signal my_signal


func _ready():
emit_signal("my_signal")
```

Ispis 1: Primjer kreiranja signala

Mnoge ugrađene vrste Godot čvorova pružaju signale koje se mogu koristiti za otkrivanje događaja. Na primjer, Area2D koji predstavlja novčić emitira signal `body_entered()` svaki put kada tijelo igrača (čvor tipa Physicsbody) uđe u oblik novčića tj. u njegov prostor, omogućujući snimanje trenutka kada igrač podiže novčić i tako emitira signal kako bi se pozvala funkcija koja povećava broj skupljenih novčića.

2.5 SilentWolf plug-in

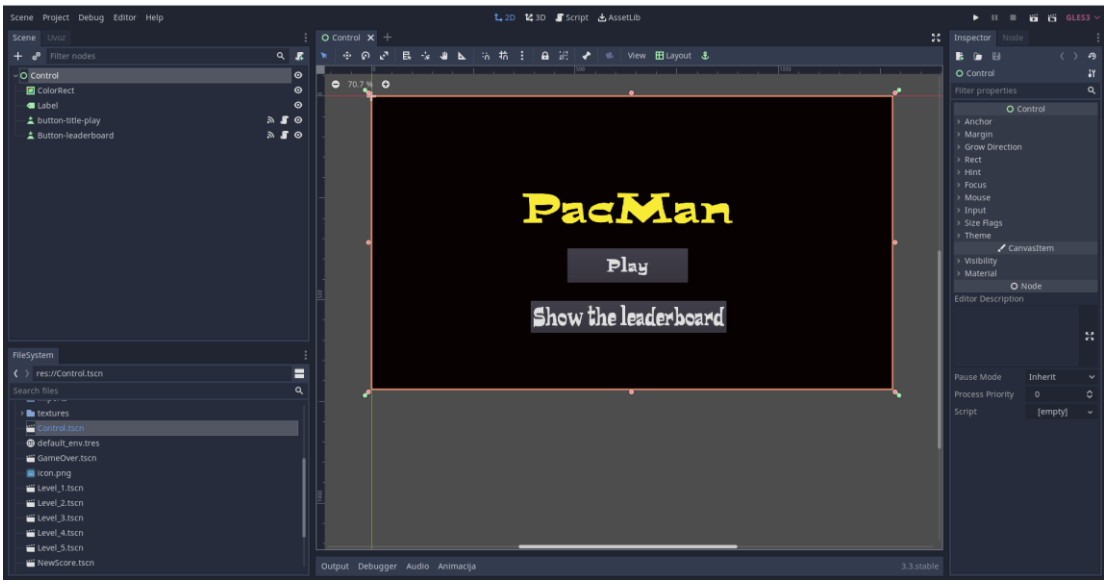
SilentWolf je plugin i kolekcija backend poslužitelja za Godot koji je ovdje poslužio kao spremanje rezultata na ljestvicu sa bodovima i dohvaćanje istih. Prilikom završetka igre igrač je dobio opciju spremanja rezultata na bodovnu listu, ali je za to potrebno spremanje svih rezultata od drugih igrača kako bi se odredilo tko se nalazi na kojem mjestu na ljestvici. SilentWolf pruža jednostavan način dodavanja gotove ljestvice za bodove, koja sprema sve prijašnje igrače i njihove bodove bez korištenja baze podataka. Također ima opciju prikazivanja same ljestvice sa svim igračima ili u ovoj igrici top 10 igrača.

3 Implementacija igre

3.1 Scene u 2D formatu

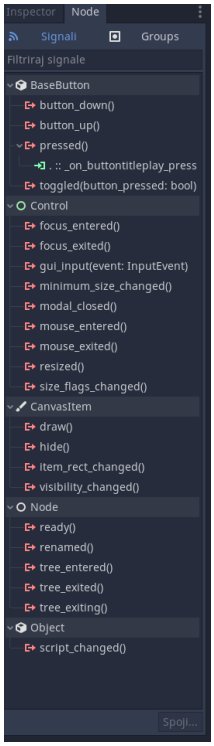
Igrica sadrži više scena gdje se u ovom poglavlju opisuju scene u 2D formatu, koje služe kao početne scene za pokretanje igrice, spremanje bodova na kraju igrice, prikazivanje igraču kada uspješno završi igricu ili u slučaju da izgubi i ostalih scena koje služe za kretanje kroz igricu. Igrica također sadrži scene u 3D formatu koje će se detaljnije opisati u sljedećem poglavlju. One su poredane po nivoima gdje je izgled svake scene drugačiji, ali su čvorovi, funkcije i skripte iste.

3.1.1 Control



Slika 11: Scena Control

Glavni čvor se zove `Control` (slika 11), sadrži čvor `ColorRect` za podešavanje boje pozadine, `Label` koji nam služi za prikazivanje običnog teksta (u ovom slučaju PacMan) gdje ima svojstva dodavanja novog fonta, promjena boje i stila. Sadrži još dva dodatna čvora a to su dva botuna. `Button-title-play` nam služi za pokretanje same igrice gdje na desnoj strani ima ikonu skripte, što znači da sadrži kôd pomoću kojega smo mu dali funkciju pokretanja igrice. Da bi botun uopće mogao djelovati kada stisnemo na njega, treba spojiti signal na njega.



Slika 12: Prikaz spojenog signala pressed()

Korišten je signal `pressed()` gdje se na slici 12 može vidjeti kako je povezan sa botunom `button-title-play`. Nakon povezivanja signala sa botunom dobije se funkcija `_on_buttontitleplay_pressed`, kao što se vidi u kôdu ispod, koja se izvršava kada stisnemo na botun Play.

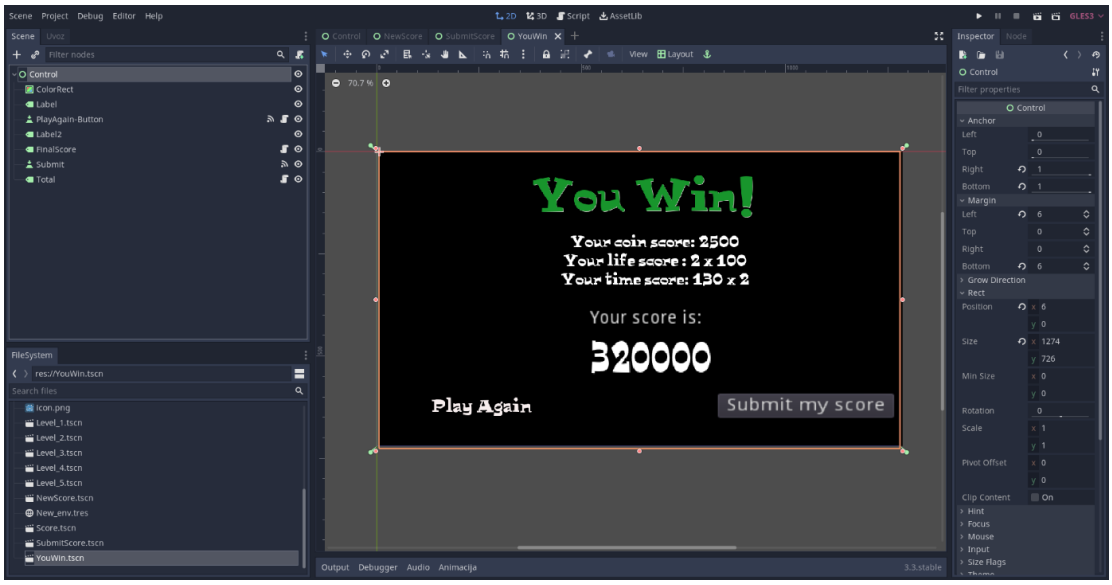
```
extends Button

func _ready():
	pass

func _on_buttontitleplay_pressed():
	Score.score = 0
	Score.candy = 2
	Score.coins = 0
	Score.time_left = 0
	Seconds.seconds = 60
	Score.player_name = 0
	get_tree().change_scene("res://Level_1.tscn")
```

Ispis 2: Kôd čvora button-title-play

Funkcija će prvo promijeniti vrijednosti dole navedenih varijabli kako bi prilikom svakog novog pokretanja igre bile uvijek nove vrijednosti za novu igru, jer igrač mora početi ispočetka. Isto tako u zadnjoj liniji kôda se nalazi naredba koja traži čvor i pokreće drugu scenu, u ovom slučaju scena `Level_1`, gdje igrač započinje prvi nivo sa početnim vrijednostima. Botun leaderboard radi istu stvar samo što mijenja na scenu `Leaderborad.tscn`

3.1.2 YouWin

U `YouWin` sceni imamo četiri čvora tipa `Label` (slika 13), od kojih su dva obična sa tekstom (`Label` i `Label2`), a druga dva imaju skriptu gdje im kôd prikazuje konačne bodove i sveukupne bodove (`FinalScore` i `Total`).



Slika 13: Scena YouWin

Kôd za `FinalScore`, ispis broj 3, gdje se uzima vrijednost iz globalne skripte Score koja sadrži vrijednost score. U toj varijabli se nalaze sveukupni bodovi koji su se tijekom igrice povećavali i koji su na kraju rezultirali konačnim bodovima igrača.

```
extends Label


func _ready():
	text = String(Score.score)
```

Ispis 3: Kôd za FinalScore

Isto je i u kôdu za `Total`, gdje se dohvaća iz globalne skripte Score vrijednost preostalih života (candy) i sveukupnog vremena što je igraču preostalo tijekom igrice kako bi se izračunao sveukupni broj bodova.

Globalna Score skripta izgleda ovako:

```
extends Spatial


var score = 0
var candy = 2
var coins = 0
var time_left = 0
var player_name = 0
```

Ispis 4: Kôd globalne skripte Score

Scena `YouWin` sadrži isto dva botuna (`PlayAgain` i `Submit`), gdje se u skripti od botuna `PlayAgain` nalaze obadvije konekcije na ta dva botuna (ispis 5).

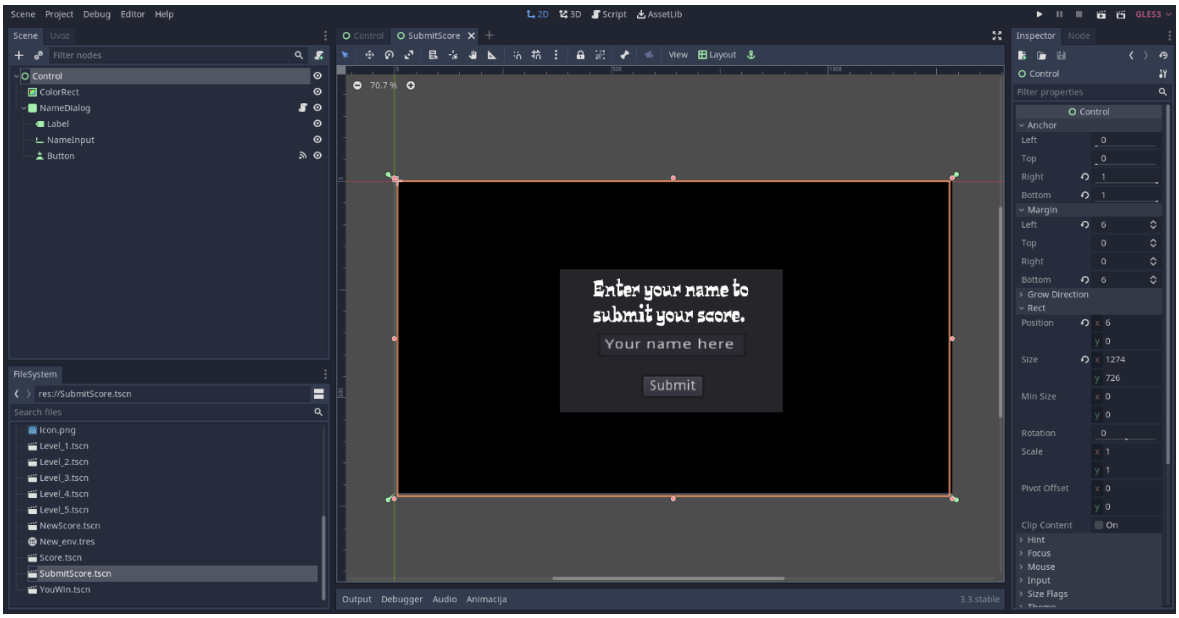
Kada se stisne botun za `PlayAgain` mijenja se scena na scenu `Control` koja je početna scena za igricu, a za `Submit` se prelazi na scenu `SubmitScore`.

```
extends Button


func _ready():
	pass


func _on_PlayAgainButton_pressed():
	get_tree().change_scene("res://Control.tscn")



func _on_Submit_pressed():
	get_tree().change_scene("res://SubmitScore.tscn")
```

Ispis 5: Kôd za PlayAgain i Submit botune

3.1.3 SubmitScore

Glavna značajka kod scene `SubimScore` je čvor `NameDialog`, tipa Panel, koji sadrži obični čvor `Label` sa tekstom unutra, botun za `Submit` koji je povezan sa signalom `pressed()` i sadrži `NameInput` (na slici je polje gdje piše „Your name here“) gdje igrač upisuje svoje ime koje se sprema kada se klikne na botun `Sumbit` (slika 14).



Slika 14: Scena SubmitScore

`NameDialog` ima skriptu preko koje može pristupiti svim njegovim čvorovima (u ovom slučaju su bitni `NameInput` i `Button`), na taj način se u skripti stvara funkcija `on_Button_pressed`.

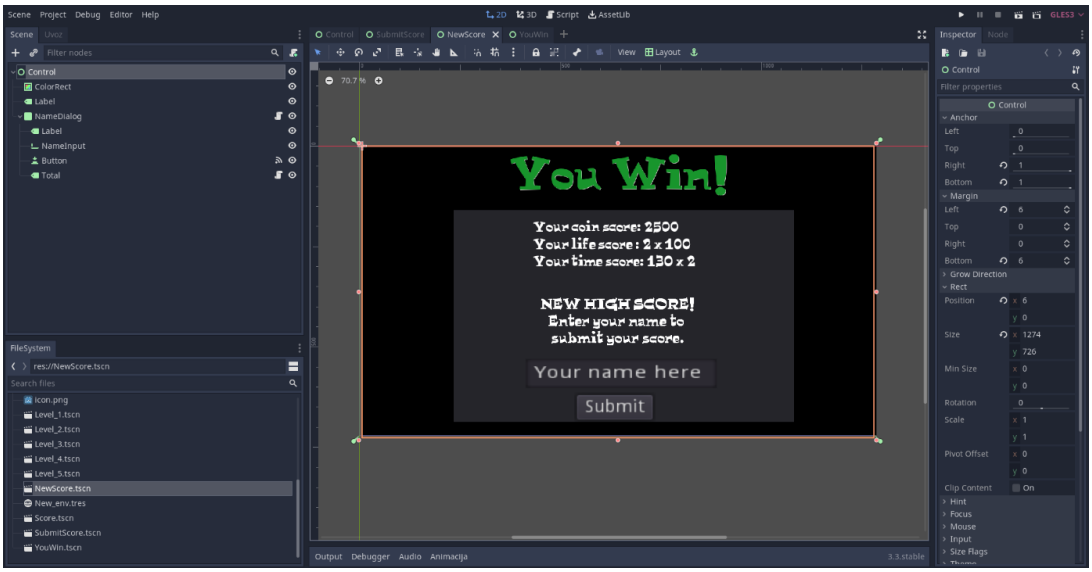
Kada se klikne na `Submit` prvo se sprema ime igrača koje je unio u `NameInput` u varijablu `player_name` koja se nalazi u globalnoj skripti Score, zatim se preko SilentWolf plugin-a šalju bodovi i ime igrača kako bi se stavio na ljestvicu sa rezultatima ostalih igrača (Leaderboard). Nakon toga u zadnjoj liniji kôda mijenjamo na scenu `Control`, koja je početna scena igrice gdje se može birati između nove igre ili pregled ljestvice sa rezultatima (ispis 6).

```
extends Panel


func _ready():
	pass


func _on_Button_pressed():
	var play_name = $"NameInput".text
	Score.player_name = play_name
	SilentWolf.Scores.persist_score(Score.player_name, Score.score)
	SilentWolf.Scores.get_high_scores()
	get_tree().change_scene("res://Control.tscn")
```

Ispis 6: Skripta čvora NameDialog

3.1.4 NewScore

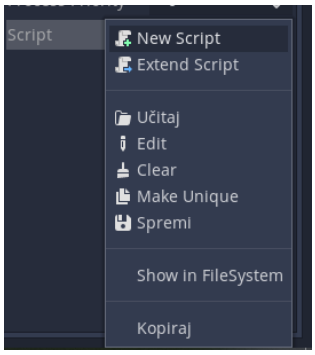
Scena `NewScore` je dosta slična sceni `YouWin`, razlika je ta što će se na temelju konačnih bodova odrediti na koju scenu se prebacuje. Ako se ne postigne novi najbolji rezultat onda se prebacuje na scenu `YouWin` gdje se igraču pokazuju njegovi bodovi i daje mu se opcija igranja ponovno bez da objavi bodove ili ako hoće ipak objaviti bez obzira što nije najbolji rezultat.



Slika 15: Scena NewScore

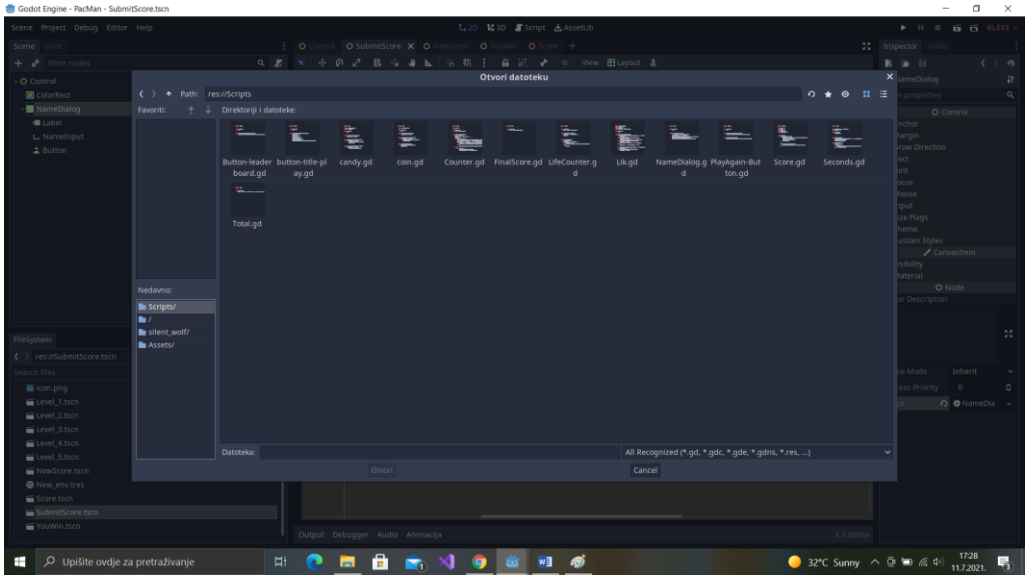
U slučaju da se postigne novi najbolji rezultat onda se prebacuje na scenu `NewScore` koja ima iste čvorove kao i `YouWin` scena, samo je drugačije postavljena (kao što se može vidjeti na slici 15).

`Total` je obični label u kojem pišemo konačne bodove od igrača koje dohvaćamo iz globalne skripte Score (`Score.score` i `Score.candy` za živote). `NameDialog` ima istu skriptu kao i `NameDialog` u `SubmitScore`, to je zapravo jedna skripta koju obadvije scene koriste jer radi istu stvar. Znači da scena `SubmitScore` i `NewScore` korisite istu skriptu, a to postignemo sa opcijom za dodavanje skripte za neki čvor. Ta opcija se nalazi na desnoj strani u Inspectoru. Na dnu se nalazi opcija za dodavanje skripte gdje klikom na nju se otvaraju nove opcije (slika 16), gdje se nudi opcija za stvaranje nove skripte ili učitavanje već postojeće što u ovakvom slučaju dobro dođe.



Slika 16: Dodavanje skripte

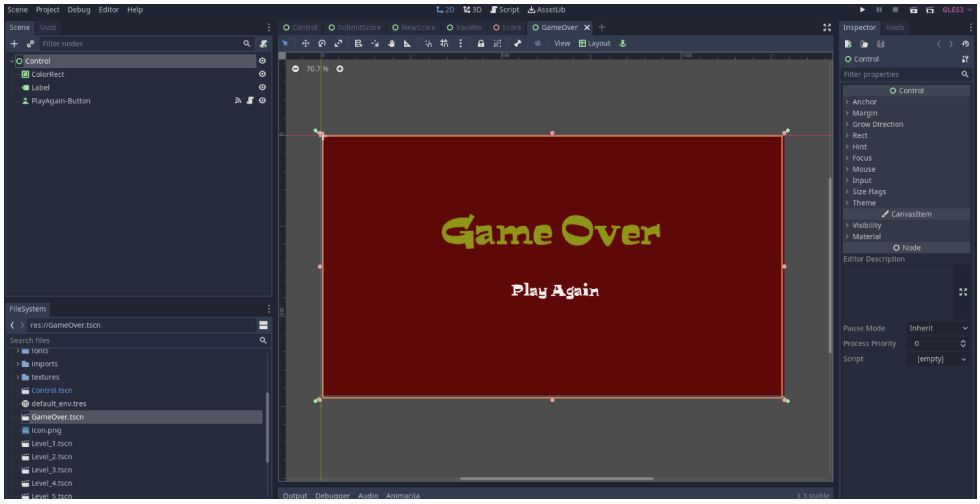
U slučaju da se odabere učitavanje već postojeće skripte otvori se novi prozor sa direktorijem i datotekama gdje se izabere tražena skripta. Na slici 17 se mogu vidjeti sve skripte od ove igrice koje su spremljene u datoteku naziva Scripts radi lakše preglednosti.



Slika 17: Mapa sa svim skriptama

3.1.5 GameOver

Scena `GameOver` se pojavljuje kada igrač izgubi (istekne mu vrijeme ili je izgubio sve živote). Prikazana je na slici 18 i sadrži samo jedan `PlayAgain` botun gdje se stiskom na njega automatski prebaci na početnu tj. `Control` scenu.



Slika 18: Scena GameOver

3.2 Scene u 3D formatu

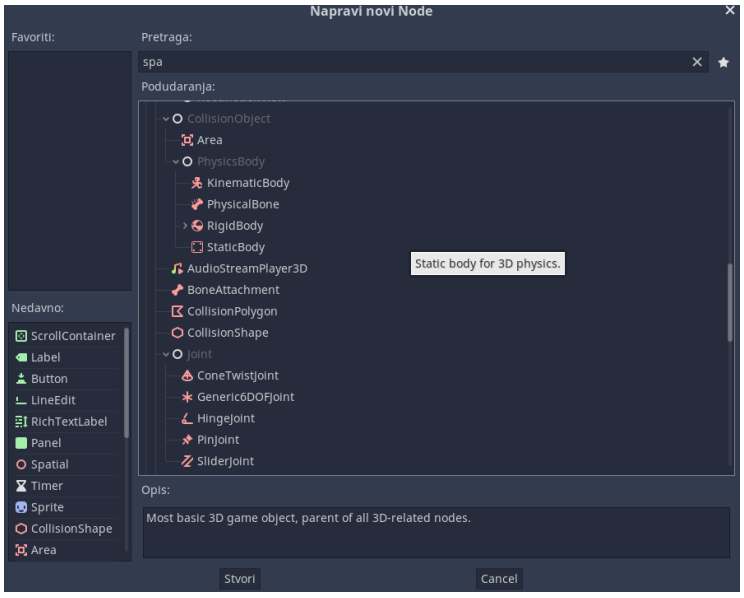
Igrica sadrži pet nivoa (slika 19), gdje je svaki nivo svoja zasebna scena sa drugačijim izgledom, ali su im funkcije i skripte iste. To je postignuto na način da se nakon izrade prvog nivoa on samo duplicirao određen broj puta tj. onoliko puta koliko bi imali nivoa i samo promijenili ime duplikata/scena u `Level2`, `Level3` itd. Što znači da se sve skripte iz prvog nivoa dijele sa ostalim nivoima jer su sve scene `Level1`, `Level2`, `Level3` itd. povezani, tako da u slučaju da se u nekoj skripti promjeni kod automatski se mijenja i na svim ostalim nivoima jer je to zapravo jedna skripta koju dijele sve ostale scene nivoa.

Slika 19: Broj razina igrice i njihove scene

Iako su scene nivoa povezane skriptama i dalje se može mijenjati njihov 3D izgled. Svaki nivo se razlikuje po izgledu poligona, broju neprijatelja, broju života dostupnim za prikupljanje i po načinu na koji su novčići poredani. Glavni čvor svakog nivoa je tipa Spatial, koji je u Godotu glavni roditelj svih ostalih 3D čvorova.

3.2.1 Izrada poligona, gridMape i kamere

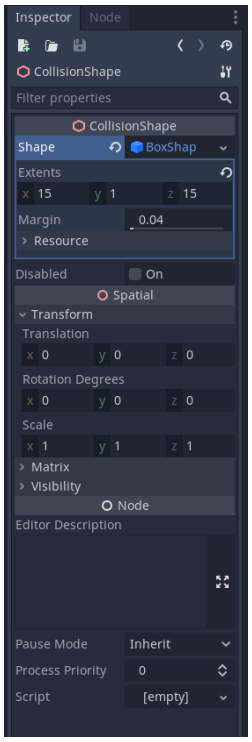
Za izradu tla se koristio čvor tipa `StaticBody` jer je upravo on objekt koji će nam biti u 3D sceni gdje neće padati pod utjecajem gravitacije, ali će se sudarati sa drugim objektima u sceni.



Slika 20: Neki od čvorova koji postoje

Slika 21: Floor

Nalazi se pod `PhysicsBody` i u igrici je preimenovano u `Floor`, kao što se vidi na slici 21. Pošto se treba odrediti oblik i same dimenzije poligona, potrebno je dodati čvor kojem će objekt `Floor` biti roditelj. Tu se koristi objekt zvan `CollisionShape` pomoću kojega se može odrediti kojeg će biti oblika i kolike su dimenzije. Na slici 22 na strani Inspectora se mogu vidjeti svojstva koja se mogu mijenjati prema potrebi. Njegova glavna uloga je da drugi objekti znaju kada dođu u kontakt s njim ili u ovom slučaju da ostali objekti ne padaju pod utjecajem gravitacije kada se nalaze na tom objektu, nego da se kreću po njemu jer ima ulogu poda.



Slika 22: Svojstva poda

Za ovu igricu je za oblik odabrana opcija `BoxShape` (kao što se vidi na slici 22), dimenzije su 15x15 i koordinate su (0,0,0), ali pošto se ovaj objekt ne može vidjeti treba se dodati još jedan objekt tj. čvor koji se zove `MeshInstance`. Koristi se na način da se preko njega umetne željeni uzorak preko kojega onda možemo vidjeti kako zapravo izgleda tlo.

Tada će čvor `Floor` imati dvoje djece (slika 23) preko kojih se definira veličina i dimenzija samog objekta koji će se sudarati sa drugim objektima i izgled tj. uzorak preko kojeg se vidi kako će pod zapravo izgledati.

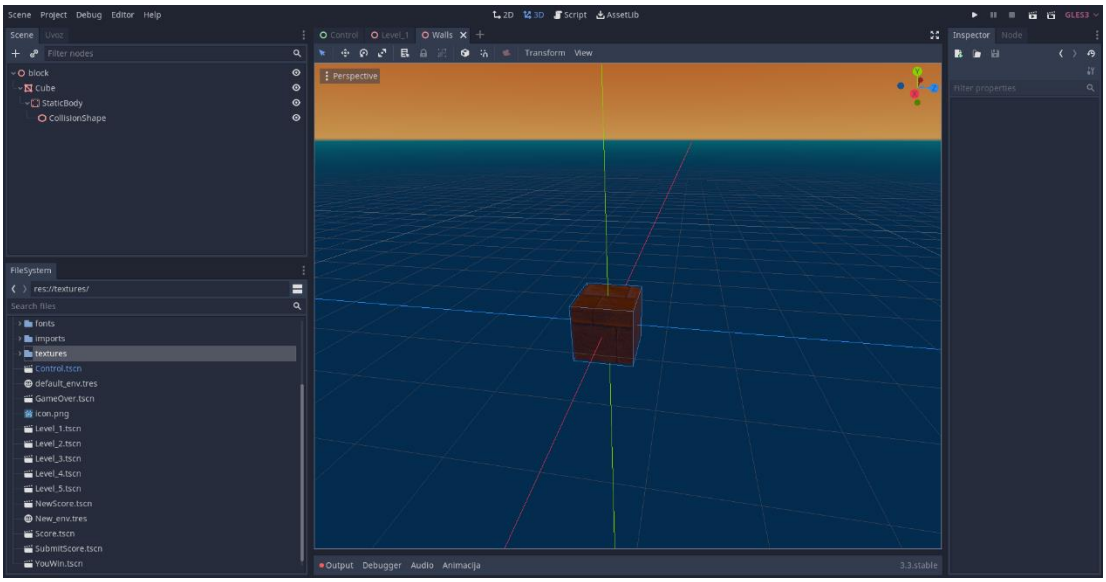
Slika 23: Čvor Floor

U datoteci pod nazivom „Textures“ su spremljeni uzorci koji su se koristili u igrici radi poboljšanja izgleda i koji su skinuti sa interneta sa besplatne stranice. `Mesh` služi samo za vizualne potrebe dok je `CollisonShape` potreban kao statičko tijelo, stoga moraju biti jednakih dimenzija i oblika.

Na slici 24 se vidi opcija materijal gdje se jednostavno klikne i povuče uzorak iz datoteke Textures i ubaci u opciju materijal, nakon koje nam se ispuni cijeli poligon sa tim odabranim uzorkom.

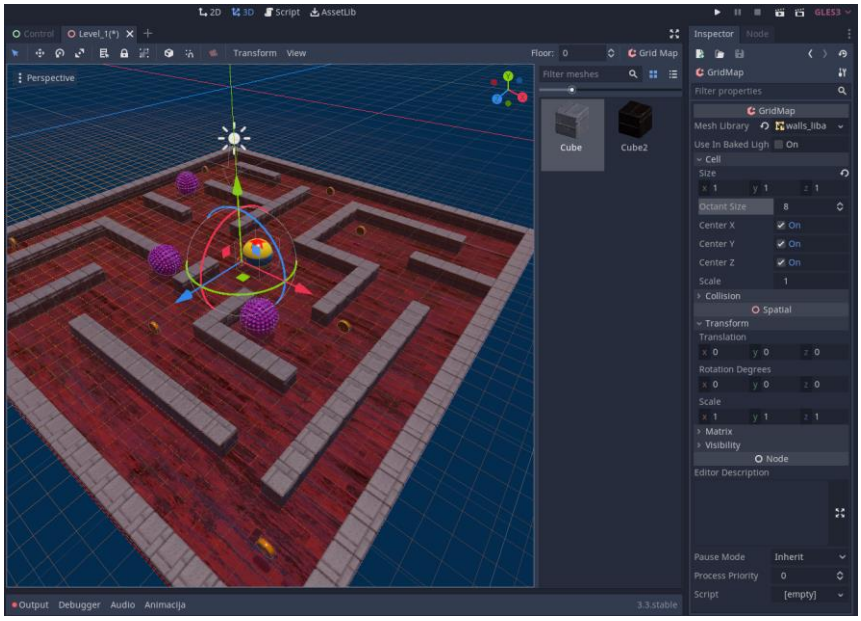
Slika 24: Dodavanje uzorka podu

Kako bi se lakše dodali zidovi na podu koristio se objekt `GridMap` preko kojeg se može umetnuti više `Mesh` instanci s kojima se mogu lakše postavljati zidovi. U ovom slučaju se prvo treba napraviti blok tj. zid koji će imati svojstvo statičnog tijela i objekt `CollisonShape` pomoću kojeg će se igrač i ostali hodajući objekti moći zaustaviti da ne idu dalje. Njegov izgled se može vidjeti na slici 25.



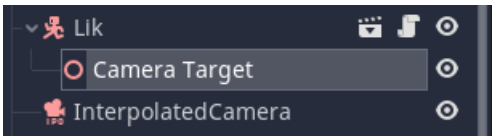
Slika 25: Izgled zida

Zatim se u GridMapu na desnoj strani u Inspectoru pod Mesh Libary doda taj zid gdje se nakon može jednostavno stavljati po podu blok po blok (slika 26). Pomoću GridMape se može namjestiti veličina po kojoj želimo da nam se poligon pokaže, u ovom slučaju 1x1 jer je i sami zid tj. blok veličine 1x1x1, pa je zbog toga olakšan način postavljanja zidova.



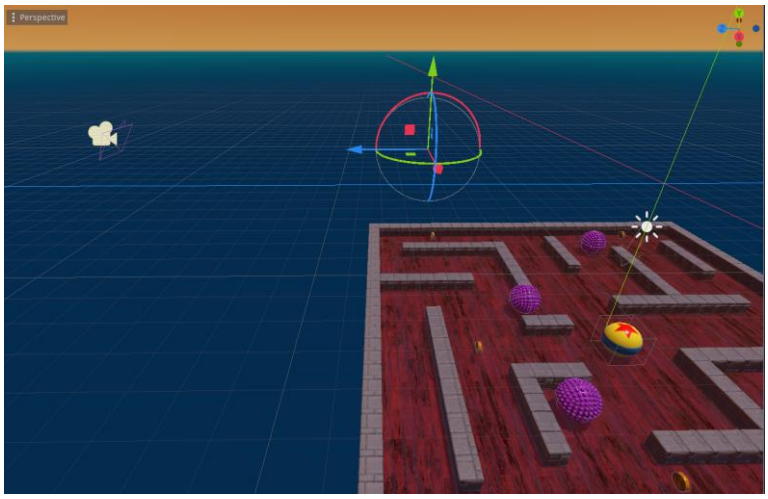
Slika 26: Postavljanje zida po poligonu

Još jedno svojstvo dodano u igricu je mogućnost praćenja lika dok se kreće, drugim riječima omogućeno je praćenje lika kamerom dok se kreće labirintom. To se postiže na način da se stvori još jedan čvor tipa Spatial (`CameraTarget` kao što se vidi na slici 27), koji će biti nevidljiv ali će i dalje biti dio lika jer je njegovo dijete pa će ga pratiti kako se lik bude micao. Samo je razlika što će čvor `CameraTarget` biti pozicioniran dalje od lika kako kamera ne bi došla skroz do igrača nego će zapravo doći do one točke gdje smo smjestili čvor `CameraTarget`.



Slika 27: Čvor Camera Tager kao dijete čvora Lik

Zatim se dodaje čvor za kameru tipa `InterpolatedCamera` čije je svojstvo da prati određenu metu ali ne toliko oštro i sa lakšim tranzicijama dok se igrač kreće. Na taj način se stvara iluzija da čvor `InterpolatedCamera` prati lika dok se kreće, ali zapravo prati metu `CameraTarget` koja je pozicionirana dalje od samog igrača i time se sprječava da se kamera ne približi skroz do igrača. To se može bolje vidjeti na slici 28, gdje ova ikona na lijevoj strani predstavlja čvor `InterpolatedCamera`, koja se može micati kroz prostor. Zatim ovaj krug desno od kamere predstavlja nevidljivi čvor `CameraTarget` koji kamera zapravo prati i lik tj. Igrač koji se nalazi na tlu u obliku žute lopte sa par detalja na njemu.

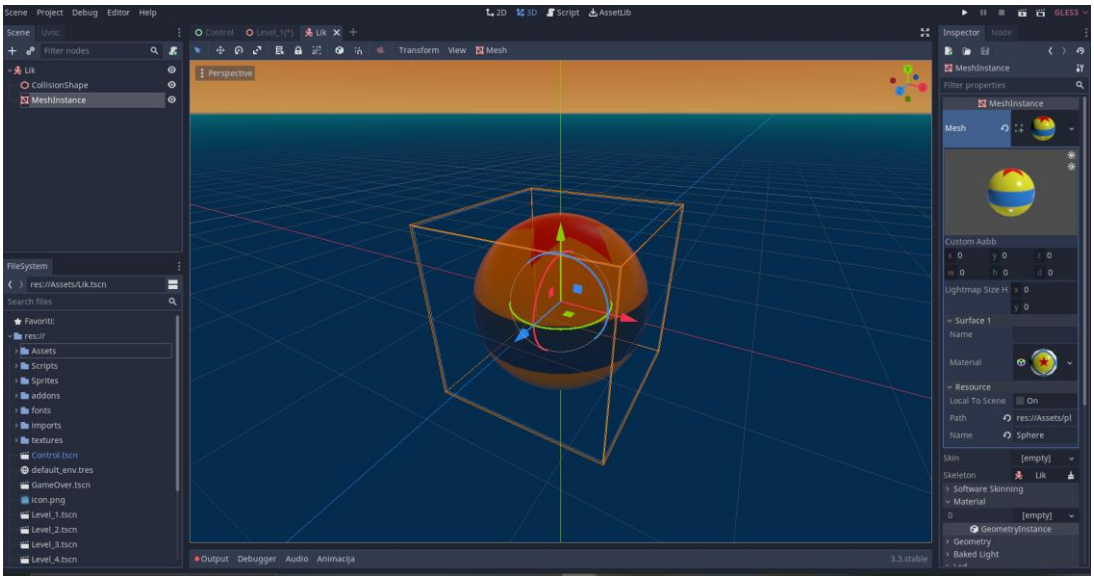


Slika 28: Prikaz praćenja lika kamerom

3.2.2 Implementacija lika

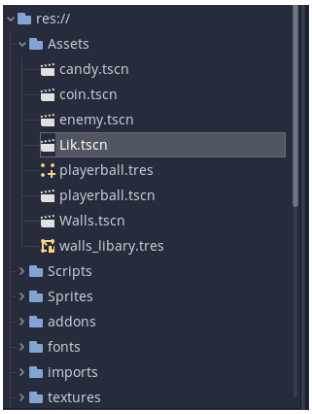
Za stvaranje lika se koristio glavni čvor tipa `KinematicBody`, koji omogućava kretanje tijela kroz prostor i zahtjeva dodavanje još jednog čvora kao njegovo dijete tipa `CollisonShape` pomoću kojega će znati kada se sudari sa drugim objektima. Za čvor `MeshInstance` se za materijal koristio gotov uzorak kojeg smo samo importali i spremili u datoteku Imports.

Iako je lik okruglog oblika za `CollisionShape` je stavljen oblik kvadrata ali u njegovim dimenzijama tako da se neće primijetiti razlika dok se lik bude kretao i nailazio na druge objekte (slika 29).



Slika 29: Izgled lika i postupak njegovog stvaranja

Igrač je stvoren kao zasebna scena spremljena u datoteku Assets, kao što se vidi na slici 30, gdje se nalaze sve ostale scene koje predstavljaju neki objekt (novčići, životi, zidovi..) ili lik koji se kreće (neprijatelj). Njih se ubacuje u glavnu scenu `Level` na način da djeluju unutar te scene, ali su i dalje zasebno svoji objekti.



Slika 30: Izgled Assets datoteke sa njezinim scenama

Nakon što je igrač dodan u glavnu scenu dodaje se skripta u kojoj će se definirati njegovo ponašanje tj. Kretanje i funkcije. Ta skripta izgleda ovako:

```
extends KinematicBody

const SPEED = 6

const ROTSPEED = 6

var velocity = Vector3(0,0,0)


func _ready():

	pass


func _physics_process(delta):

	if Input.is_action_pressed("ui_right") and
Input.is_action_pressed("ui_left"):

		velocity.x = 0

	elif Input.is_action_pressed("ui_right"):

		velocity.x = SPEED

		$MeshInstance.rotate_z(deg2rad(-ROTSPEED))

	elif Input.is_action_pressed("ui_left"):

		velocity.x = -SPEED

		$MeshInstance.rotate_z(deg2rad(ROTSPEED))

	else:

		velocity.x = lerp(velocity.x, 0, 0.1)
```

Ispis 7: Kôd Lika (1. dio)

```
if Input.is_action_pressed("ui_up") and
Input.is_action_pressed("ui_down"):
              velocity.z = 0
       elif Input.is_action_pressed("ui_up"):
              velocity.z = -SPEED
              $MeshInstance.rotate_x(deg2rad(-ROTSPEED))
       elif Input.is_action_pressed("ui_down"):
              velocity.z = SPEED
              $MeshInstance.rotate_x(deg2rad(ROTSPEED))
       else:
              velocity.z = lerp(velocity.z, 0, 0.1)
       move_and_slide(velocity)
```

Ispis 7: Kôd Lika (2. dio)

Funkcija `_physics_process(delta)` je ugrađena funkcija Godota koja se vrti konstantno (kroz 60 okvira u sekundi) provjeravajući da li se dogodila neka promjena. U ovom slučaju provjerava da li je neki botun, koje se unaprijed odredilo, na tipkovnici stisnuto kako bi se lik mogao upravo u tom smjeru kretati.

Pomoću gotove funkcije `Input.is_action_pressed()` se može pratiti kada je koja tipka pritisnuta na način da se unutar zagrada napiše naziv te tipke. Gdje se u ovom

slučaju gledaju samo tipke za lijevo, desno, gore i dolje. U slučaju da se pritisne lijeva ili desna tipka tj. Da se igrač kreće lijevo ili desno, vrijednost x-a u Vector3 će se povećati ili smanjiti za 6 ili -6, ovisno o smjeru kretanja. Određeno je da je vrijednost konstante SPEED jednaka šest gdje će se za toliko igrač po x osi kretati. Isto vrijedi i za tipke gore i dolje samo što se tada igrač ne kreče po osi-x nego po osi-z.

Najbitnije je da se na kraju pomoću funkcije `move_and_slide`, koja je ugrađena funkcija za čvor `KinematicBody`, igrač zapravo kreće. Kao parametar se traži varijabla tipka Vector3 kako bi znalo u kojem smjeru se kreće.

U slučaju da se ne dira ni jedan botun igrač tada se pomoću `else` naredbe poziva naredba `lerp`, koja govori da se igrač postepeno zaustavlja ili u ovom slučaju da se od 6 do 0 mijenja svaki put za deset posto i time se igrač polako zaustavlja umjesto da odmah stane.

Isto tako se prilikom kretanja igrača u određenom smjeru može vidjeti kako se lopta okreće upravo u tom smjeru, to se postiglo sa funkcijom `rotate`. Ako se radi o kretanju u lijevo ili desno onda se koristi funkcija `rotate_x` i unutra brzina kojom želimo da se lopta vrti, a ako se radi o kretanju prema dole ili gore onda se koristi funkcija `rotate_z` . Pošto je bitno da izgleda kao da se lopta okreće u ovom slučaju će se koristiti Mesh same lopte jer upravo ona čini sami izgled lopte tj. uzorak. Tako da će njemu pristupiti pomoću naredbe `$MeshInstance`, jer znak $ nam omogućuje da pristupimo svim čvorovima tog roditelja (u ovom slučaju `Lik`) koji sadrži dva čvora, `CollisionShape` i `MeshInstance`.

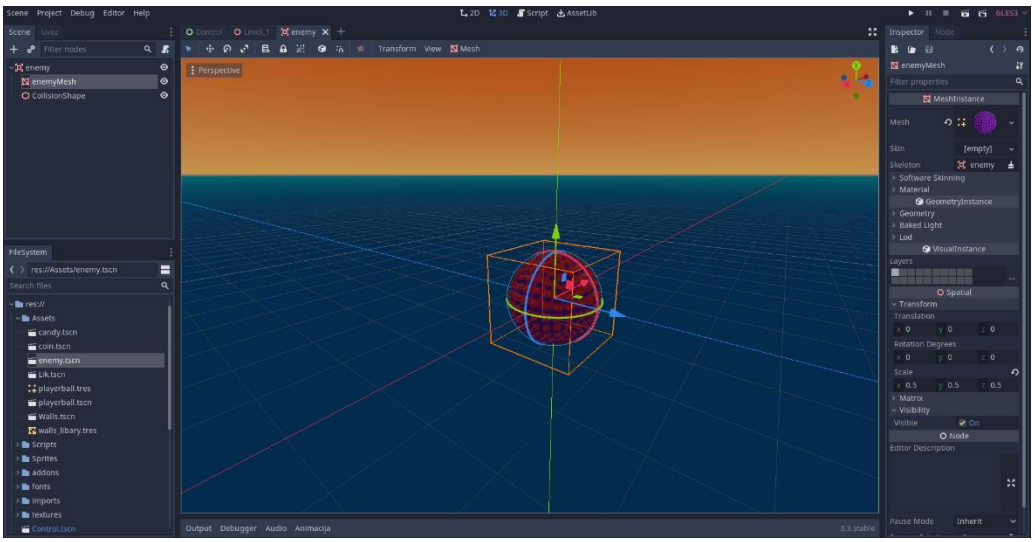
Pošto funkcija `rotate` koristi radijane, pomoću funkcije `deg2rad` će se stupnjevi koje upišemo kao parametre pretvoriti u radijane koje će onda funkcija `rotate` prihvatiti.

Na taj način smo postigli da nam se lik kreće prilikom stiskanja određenog botuna i daje izgled samog kretanja u određenom smjeru.

3.2.3 Implementacija neprijatelja

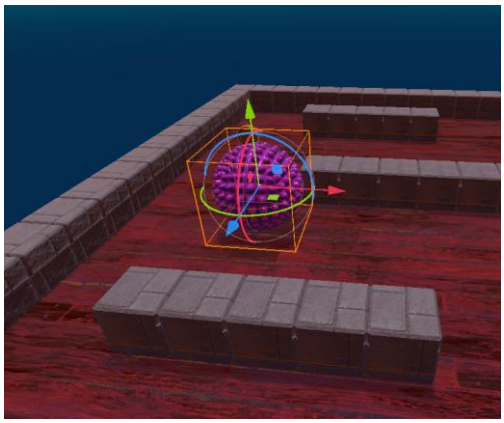
Za stvaranje neprijatelja se koristio tip objekta zvan `Area` kao glavni čvor. Njegove karakteristike su što nema svojstvo sudaranja sa ostalim fizičkim tijelima nego bi samo prošao kroz ta tijela. Upravo to i odgovara za ovu igricu jer je cilj da kôd samo javi kada igrač dođe u kontakt sa neprijateljem kako bi se smanjio broj života ili u slučaju da je to bio zadnji da prebaci na scenu `GameOver`.

Objekt `Area` zadovoljava taj kriterij jer kako i sam naziv govori, to je objekt koji se kreće kroz prostor. Također mora imati dodatne objekte tj. djecu tipa `Mesh` i `CollisionShape`, gdje je dan oblik `SphereShape` i `CollisonShape` kako bi se moglo detektirat kada neprijatelj dođe u kontakt sa igračem gdje će se to do kraja realizirati u kôdu.



Slika 30: Izgled neprijatelja i svojstva na desnoj strani

Kada se `enemy` spremi kao scena u datoteke Scenes i doda u glavnu scenu `Level`, treba pozicionirati pravilo objekt kako bi izgledalo kao da je na tlu. To se može jednostavnim klikom na objekt gdje će se oko njega pojaviti opcije za micanje po x,y i z osi, također i za rotiranje u kojem smjeru da bude okrenuto i sl. (slika 31).



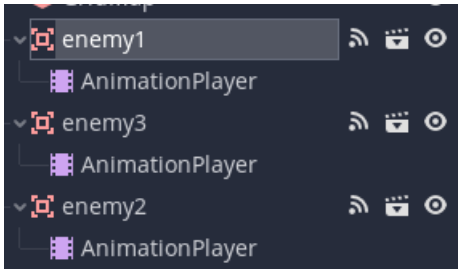
Slika 31: Opcija za micanje neprijatelja kroz poligon

Gdje je crvena strjelica za os x, zelena za os y i plava za os z ili se može promijeniti na način da se unese x y i z vrijednosti u Inspectoru, prozor na desnoj strani gdje se nalaze sve karakteristike i svojstva samog objekta. Na slici 32 se to može vidjeti pod opcijom transform gdje pišu koordinate tog objekta.

Slika 32: Detaljniji prikaz Inspectora za neprijatelja

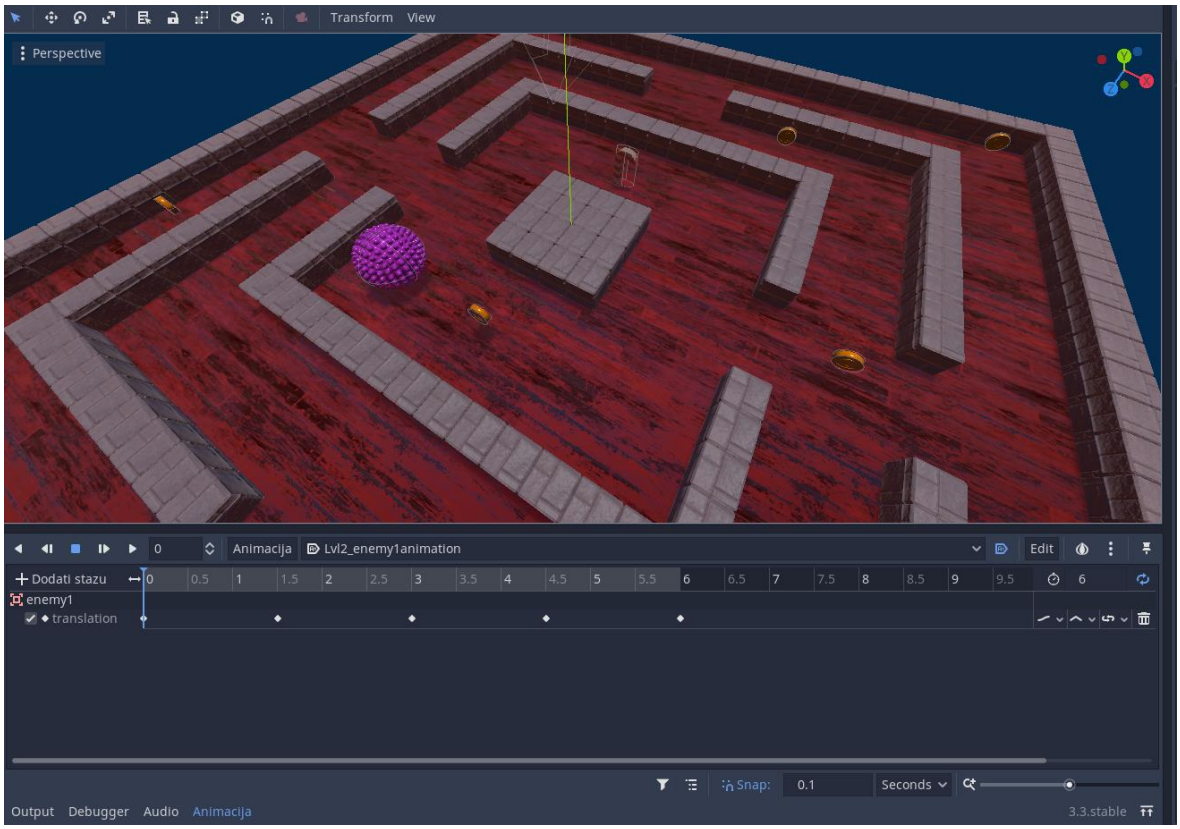
Isto tako pošto se objekt tj. neprijatelj mora kretati određenim putem koristi se novi objekt koji će biti dijete objekta `enemy`. Objekt se zove `AnimationPlayer` i pruža mogućnost da se odredi način na koji će se objekt kretati po poligonu i koliko dugo.

Kako će biti više neprijatelja u glavnoj sceni, objekt `enemy` će se duplicirati određeni broj puta (koliko nam je za svaki nivo potrebno zlikovaca) i svaki će imati drugačije kretanje po poligonu. Stoga svaki objekt `enemy` mora imati drugačije postavljen objekt `AnimationPlayer` i baš zbog toga se on ne smije dodati u orginalnu scenu `Enemy` jer će onda svaki imati ista ta svojstva i isti način kretanja. Da bi se uspješno svaki objekt `enemy` u glavnoj sceni `Level` kretao na svoj određeni način, svaki enemy mora imati svoj objekt `AnimationPlayer` gdje će se za svaki odrediti drugačija ruta kretanja. (slika 33)



Slika 33: Duplicirani neprijatelji

`AnimationPlayer` ima svoju traku koja se pojavi na dnu zaslona koja na početku bude prazna jer se treba kreirati nova animacija. Gledajući sliku 34, to se može ostvariti klikom na botun Animacija gdje će izbaciti više izbora ali u ovom slučaju će biti potrebno novo kreiranje animacije tako da će se izabrati opcija Novo nakon koje upisujemo naziv animacije.



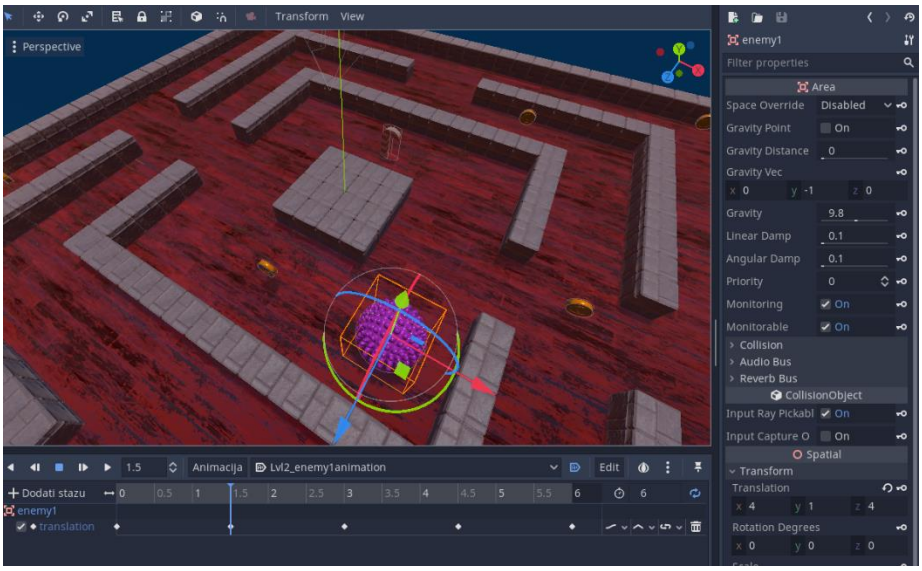
Slika 34: Stvaranje animacije neprijatelja broj 1

Za primjer je uzeta animacija neprijatelja broj 1, na drugom nivou gdje se animacija zove `Lvl2_enemy1animation`. `AnimationPlayer` funkcionira na način da ima punu kontrolu nad svojim objektom (u ovom slučaju `enemy`), gdje će se u Inspectoru pojaviti ikone ključa kod svake opcije neprijatelja (slika 35).

Slika 35: Svojstva objekta

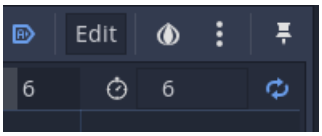
One označavaju kada želimo zaključati određenu poziciju ili karakteristiku u tom trenutku, gdje je u ovom slučaju najbitnija opcija Transform koja označava koordinate gdje se objekt nalazi u tom trenutku.

Gledajući na sliku 35, za početak se treba odredi koliko će kretanje trajati. U ovom slučaju 6 sekundi i početna pozicija objekta `enemy` je (-4, 1, 4) gdje se pritisne ključ kod opcije transform kako bi u početnoj sekundi objekt započeo u toj poziciji. Na slici se može vidjeti da su te točke/koordinate koje se zaključaju označene u donjoj traci kao mali kvadrati kako bi se znalo na koliko mjesta i pozicija se objekt kreće. Zatim se odredi na koju će se stranu objekt dalje kretati i koliko će mu trebati vremena do te točke, gdje je objekt nakon sekunde i pol stigao na poziciju sa novim koordinatama. Na slici 36 se vidi nova pozicija sa novim koordinatama (4,1,4), što znači da će objekt `enemy` sa početne pozicije (-4,1,4), za sekundu i pola stići na novu poziciju (4,1,4).



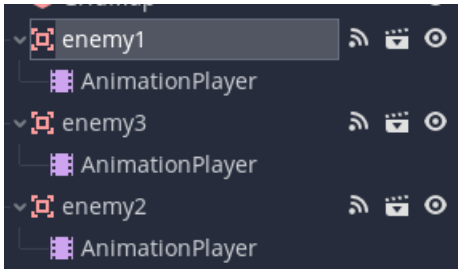
Slika 36: Druga pozicija objekta (4,1,4)

Na taj način će se i za sve ostale točke odrediti. Bitno je da se označi da objekt krene odmah na početku scene kada se pokrene (plava ikona na slici 37, gornji lijevi kut) i opcija za ponavljanje animacije (donji desni kut), kako bi objekt stalno bio u pokretu dok se igrač kreće kroz labirint.



Slika 37: Opcija automatskog kretanja i ponavljanja animacije

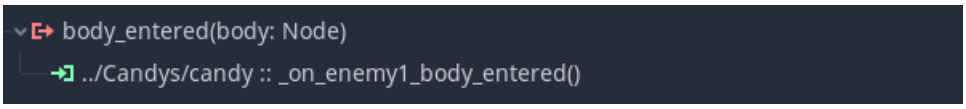
Objekt `enemy` isto tako ima jednu konekciju tj. Signal sa još jednim objektom. To se može vidjeti po ovoj ikoni, na slici broj 38, u obliku signala.



Slika 38: Konekcija sa objektom enemy

Radi se o signalu `body_entered()` koji je spojen sa objektom `candy` (što u igrici predstavlja živote igrača tj. Broj preostalih pokušaja), na slici 39, a funkcija koja je stvorena prilikom spajanja ta dva objekta se nalazi u skripti od objekta `candy` i zove se `_on_enemy1_body_entered(body)`.

Ideja je da kada igrač dođe u kontakt sa neprijateljem, gdje se automatski izvršava funkcija `_on_enemy1_body_entered(body)` i provjerava da je body zapravo igrač a ne neki drugi objekt, šalje se poseban signal pod nazivom `candyLost` koji se nalazi u skripti objekta `candy` (jer je taj signal ručno napisan u kôdu za potrebe ove igrice) i dalje se obavljaju funkcije gdje se smanjuje broj života za jedan i u isto vrijeme se to pokazuje na zaslonu dok igrač još igra. Sama funkcija se nalazi u skripti objekta `candy`, koji će se kasnije detaljnije objasniti.



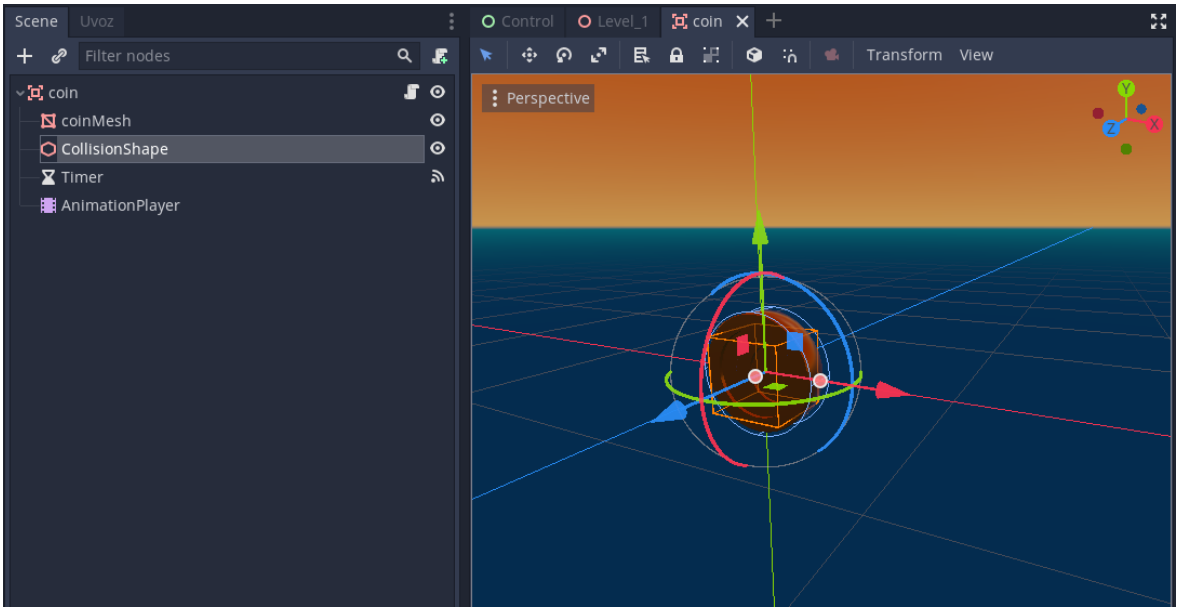
Slika 39: Signal između objekta enemy i candy

3.2.4 Implementacija novčića

Stvaranje novčića je na isti princip kao i igrač i neprijatelj. Stvoren kao objekt `Area` gdje je dan oblik cilindra za `CollisionShape`, a sami izgled tj. uzorak je importan u Mesh instancu sa interneta. Objekt je tipa `Area` jer je potrebno samo da se može detektirati kada igrač uspostavi kontakt sa određenim novčićem. Također je spremljen kao svoja zasebna scena pa je dodan u glavnu scenu `Level` gdje je dupliciran više puta (ovisno koliko treba novčića za koji nivo).

Kod novčića je isto dodan objekt `AnimationPlayer`, samo što je tu razlika što će animacija za svaki novčić biti ista jer je dio glavne scene novčića (slika 40). Njegova animacija se sastoji od poskakivanja novčića u zrak nakon što ga igrač pokupi gdje se aktivira objekt `Timer` i kada on istekne (nakon zadanog vremena) novčić nestane. Ovo ne

bi moglo biti moguće bez kôda, kao ni samo brojanje skupljenih novčića i prelaženje na sljedeći nivo nakon što se skupio određen broj.



Slika 40: Čvorovi objekta coin

Skripta je dodana kao dio glavne scene novčića kako bi se svaki novčić na isti način ponašao. U kôdu se nalaze tri funkcije i jedan signal koji se ručno stvorio. Taj signal se automatski pojavi zajedno kod ostalih gotovih signala na desnoj strani u Node prozoru.

```
extends Area

signal coinCollected

func _ready():
	pass

func _physics_process(delta):
	rotate_y(deg2rad(3))



func _on_coin_body_entered(body):
	if body.name == "Lik":
		$AnimationPlayer.play("bounce")
		$Timer.start()



func _on_Timer_timeout():
	emit_signal("coinCollected")
	queue_free()
```

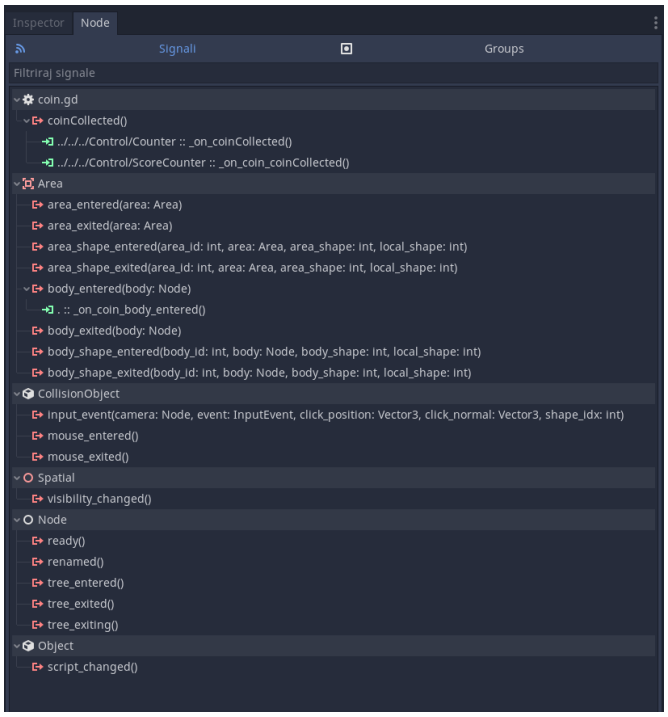
Ispis 8: Kôd objekta coin

Signali, kao što je prije spomenuto, se mogu slati od jednog objekta na drugi kako bi se obavijestili da se nešto dogodilo bez da se stalno provjerava da li je došlo do neke promjene ili sl. U ovoj igrici nam služe za javljanje kada je igrač došao u kontakt sa neprijateljem, novčićem i dodatnim životom. Svaki je drugačije implementiran, ali je način

na koji rade i svrha uvijek ista. Također je signal spojen sa svim novčićima na poligonu kako bi se za svaki emitirao pravilno signal kada se pokupi od strane igrača.

Kod novčića, koji je objekt tipa `Area`, ima već gotove signale (slika 41). Gdje se ovdje koristio signal `body_entered (body:Node)` koji se emitira onda kada dođe u kontakt sa nekim fizičkim tijelom, gdje je upravo to potrebno. Taj signal se spaja sa novčićem i stvara se nova funkcija naziva `_on_coin_body_entered(body)`, s tim da body može biti bilo koji fizički objekt pa se na početku odmah provjerava sa `if` naredbom da li je igrač taj koji je pokupio novčić (u suprotnom se ništa ne radi, novčić i dalje ostaje na istom mjestu bez promjena). Nakon čega se pokreće objekt `AnimationPlayer` kojeg smo dohvatili sa znakom $, gdje se pušta animacija kada novčić skače u zrak u smislu da je pokupljen.

Zatim se pokreće objekt `Timer`, koji je namješten da traje 0.4 sekunde radi lakšeg praćenja kada se pokupi i pokreće funkciju `_on_Timer_timeout` gdje se emitira signal `coinCollected` koji je, kako je maloprije navedeno, ručno napisan u kôdu. U slici ispod se vidi da je upravo taj signal spojen na dva dodatna objekta koji se nalaze u čvoru `Control` koji će se kasnije detaljnije opisati. Objekti `Counter` i `ScoreCounter` su djeca objekta tj. čvora `Control`. Glavni razlog ovoga je što emitiranjem tog signala, prilikom skupljanja novčića, u ta ostala dva objekta se automatski izvršavaju funkcije kreirane u tim objektima, kako bi se pomoću koda u tim funkcijama povećao broj skupljenih novčića i samih bodova gdje ih oboje prikazujemo na zaslonu kroz igricu. Zaslon je definiran kroz maloprije spomenut objekt `Control` koji nam predstavlja korisničko sučelje (prikazani bodovi, vrijeme, životi i broj novčića), koji će se kasnije detaljnije opisati. A te funkcije su kreirane kada se signal `coinCollected` povezao sa objektima `Counter` i `ScoreCounter`, isto kao i na slici iznad gdje se kreirala funkcija `_on_body_entered`() kada se povezao signal sa objektom novčić.

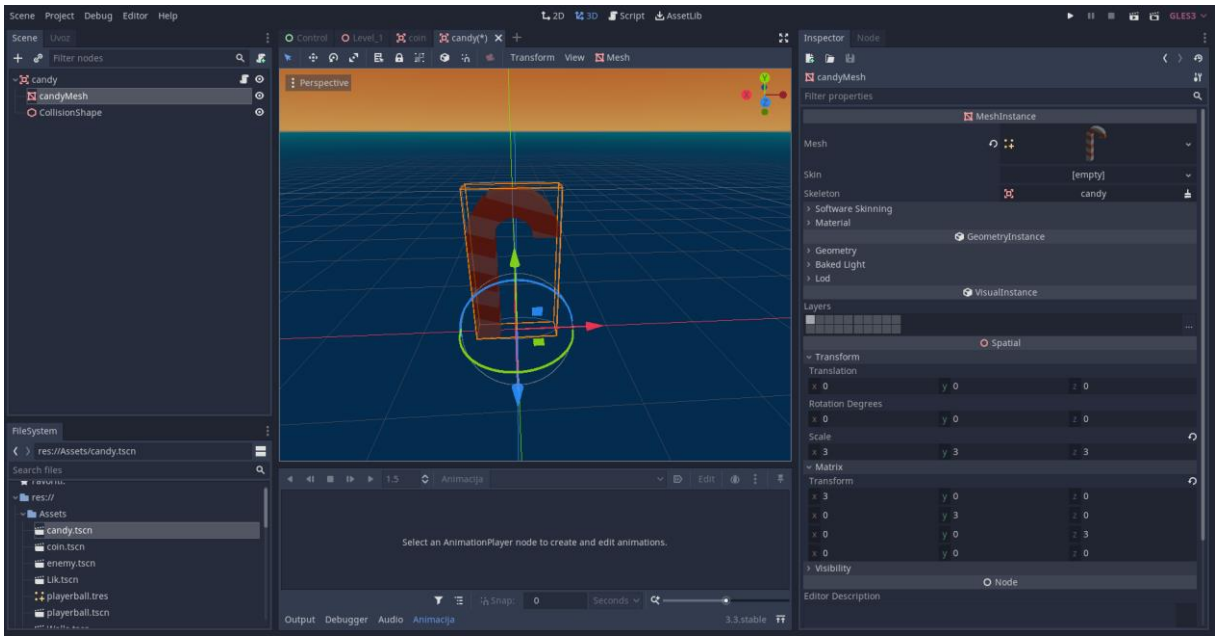


Slika 41: Signali objekta novčić

Funkcija `_physics_process(delta)` daje novčiću dodatnu animaciju na način da se vrti oko svoje osi radi vizualne estetike. Ovdje se mogu vidjeti i neka dodatna svojstva koje Godot pruža i razne vrste animacija.

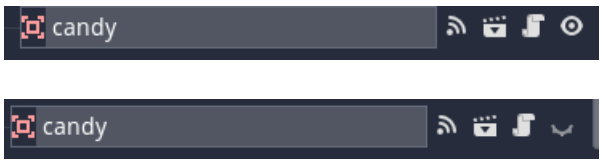
3.2.5 Implementacija života

Izrada života ili u igrici kako se zove, candya, se također koristio tip objekta `Area` kao i za novčić. Gotovi oblik candya je importan sa interneta, gdje je stavljen u `MeshInstance`, a za `CollisionShape` se koristio također oblik cilindra jer najbolje obuhvati oblik objeka `Candy`.



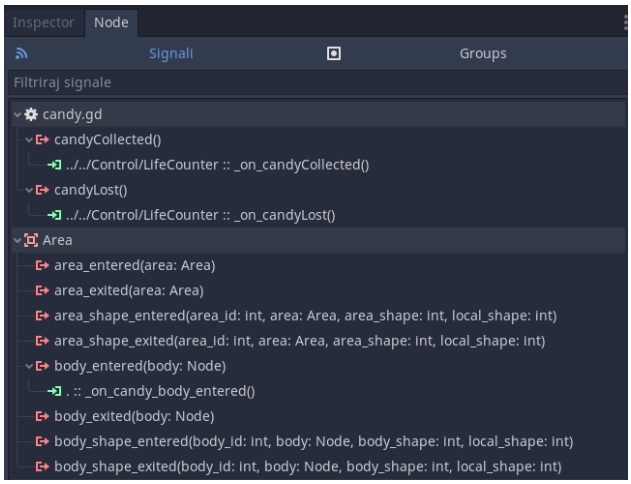
Slika 42: Stvaranje i izgled objekta candy

Na isti način se spremio kao i objekt novčić (slika 42), kao scena, i zatim se ubacio u glavnu scenu `Level` ali bez duplikata. Na svakom nivou je znači po jedan život, ali da bi se igrica malo otežala nije na svakom nivou dostupan život. Pomoću opcije da se objekt sakrije na zaslonu se može izbjeći mogućnost skupljanja života na svakom nivou. Na slici 43 se vidi kada je objekt vidljiv a kada ne, pomoću male ikone u obliku oka se može uključiti i isključiti.



Slika 43: Uključivanje vidljivosti objekta candy

Objekt `candy` ima tri konekcije tj. signala (slika 44), gdje su dva od njih ručno pisani signali u kôdu. Ispis 9 pokazuje kôd objekta `candy`. Signali `candyCollected` i `candyLost` su dodani u kôd i spojeni sa objektom `LifeCounter` koji je dijete objekta `Control`.



Slika 45: Signali čvora candy

Također se može vidjet da je `candy` povezan sa signalom `body_entered` gdje se vidi funkcija na slici iznad, koja se aktivira kada objekt `candy` dođe u kontakt sa drugim objektom tj. `If` naredba provjerava da li se radi o igraču (u suprotnom se ništa ne mijenja i nastavlja se dalje igrica) i ako je istina onda se emitira signal `candyCollected` koji će „aktivirati“ funkciju u skripti objekta `LifeCounter` i izvršavati njezin kôd, koji će se detaljnije objasniti kasnije.

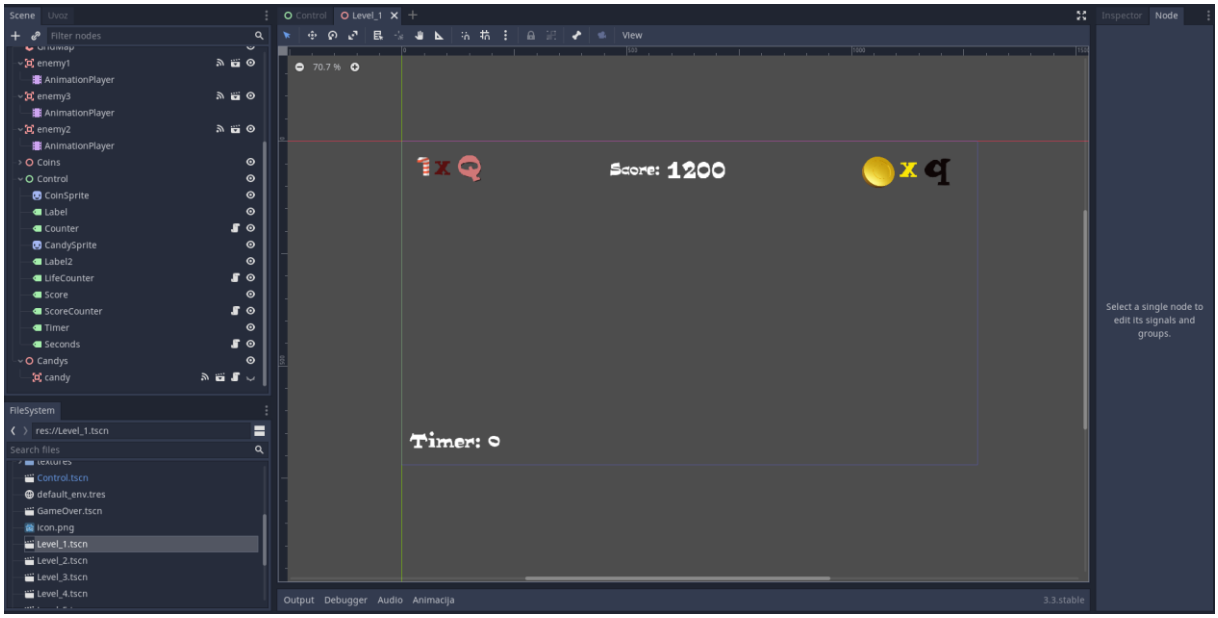
Isto tako je ranije je u poglavlju o objektu `enemy` spomenuta konekcija između objekta `enemy` i `candy`, gdje je ideja da kada igrač dođe u kontakt sa neprijateljem da se funkcija aktivira u skripti objekta `candy` kako bi se emitirao signal `candyLost` i u skripti objekta `LifeCounter`, sa kojom je signal `candyLost` povezan, izvela funkcija gdje se smanjuje broj života ili čak prekida igra u slučaju da je izgubljen zadnji život. Detaljnije o tome će biti u sljedećem poglavlju.

```
extends Area


signal candyCollected
signal candyLost


func _ready():
	pass


func _physics_process(delta):
	 rotate_y(deg2rad(3))



func _on_candy_body_entered(body):
	if body.name == "Lik":
		emit_signal("candyCollected")
		queue_free()



func _on_enemy1_body_entered(body):
	if body.name == "Lik":
		emit_signal("candyLost")
```

Ispis 9: Kôd objekta candy

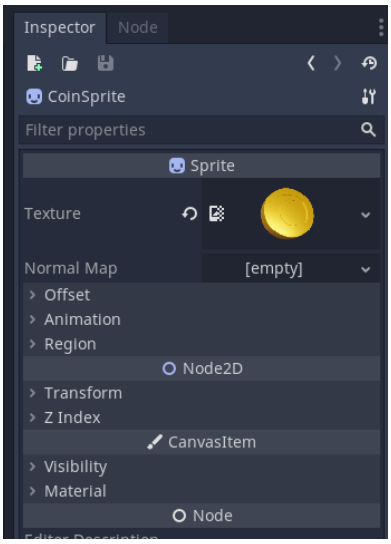
3.2.6 Control

`Control` služi za prikaz bodova, broj skupljenih novčića, broj života i prikaz preostalog vremena kroz igru i sam objekt/čvor se zove `Control` (slika 46). U 2D je formatu i tu se nalazi većina kôda i većina globalnih skripti kojima se pristupa kroz cijeli glavni čvor `Level`.



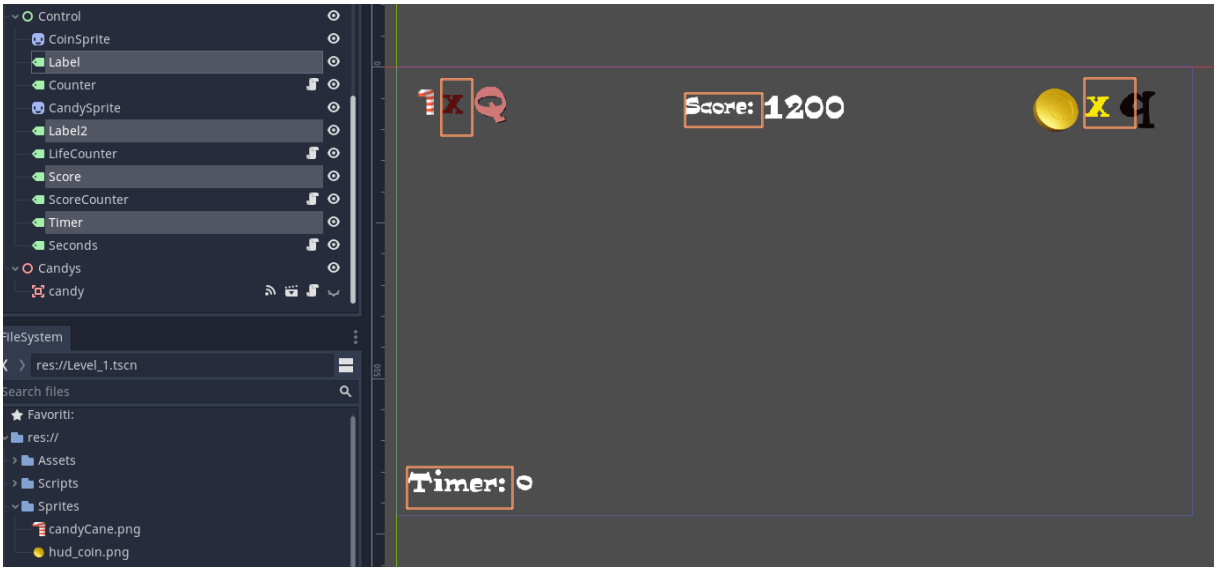
Slika 46: Izgled čvora Control

Sastoji se od običnih objekata `Label` i dva Spritea. Čvor `Sprite` služi za obični prikaz neke slike/ikone, u ovom slučaju `CoinSprite` za sliku novčića i `CandySprite` za sliku slatkiša koji se također dobiju u 2D formatu. Spremljeni su u datoteci Sprites i za korištenje se samo povuče ikona iz datoteke u desnu traku u Insprector prozor pod opciju textures (slika 47).

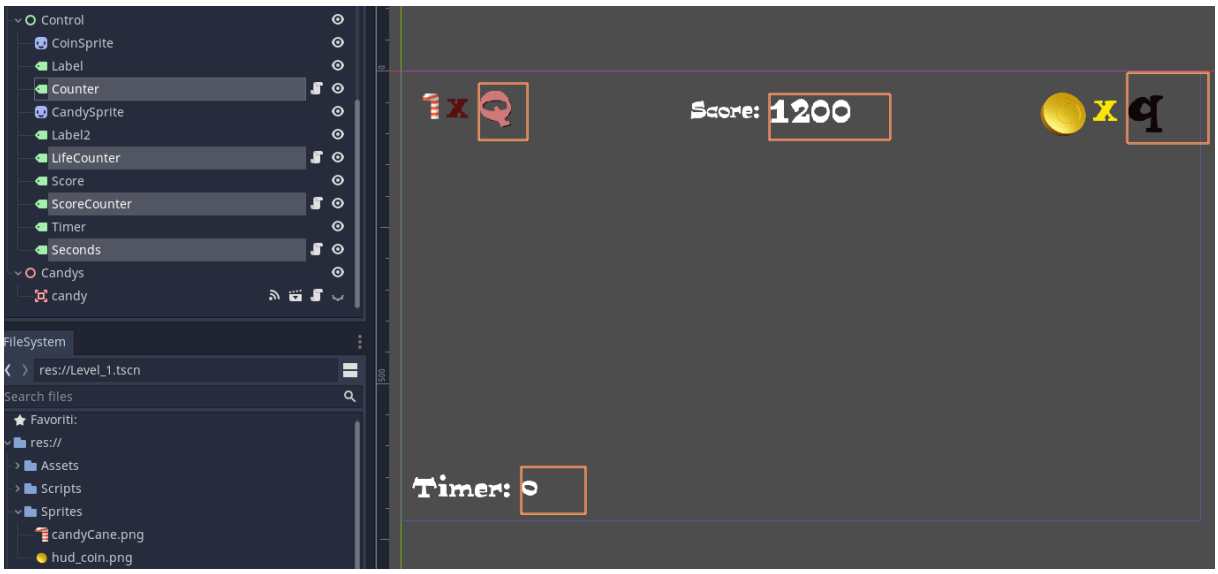


Slika 47: Stvaranje sprita

Čvorovi `Label`, `Label2`, `Score` i `Timer` služe kao obični tekstualni objekti (slika 48), dok čvorovi `Counter`, `LifeCounter`, `ScoreCounter` i `Seconds` (slika 49) isto služe za prikaz teksta, ali svaki od njih ima svoju skriptu gdje im kod obavlja bitne funkcije kao što su povećavanje i smanjivanje života prilikom skupljanja ili sudaranja sa neprijateljem, brojanje novčića i kada se prelazi na drugi nivo. Također kada je igra gotova, koji je konačni broj bodova, na koju scenu će se ići na temelju tih bodova itd.

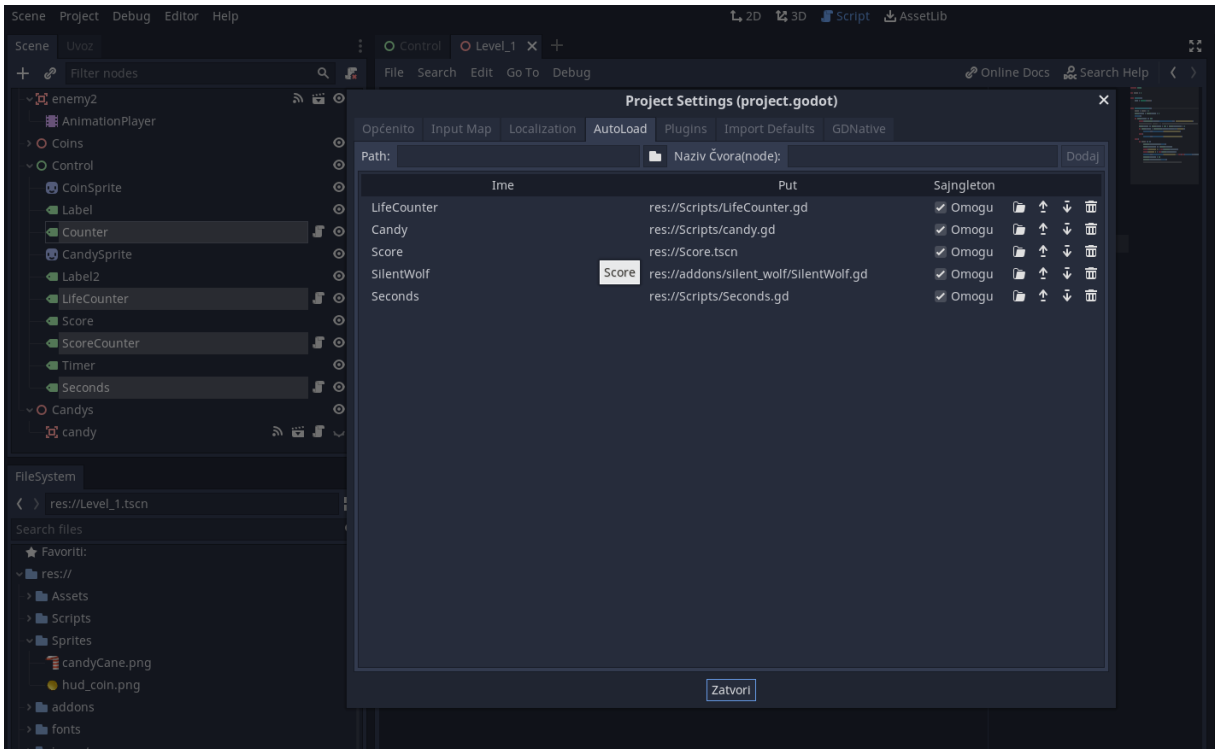


Slika 48: Čvorovi Label



Slika 49: Ostali čvorovi sa skriptama i bitnim funkcijama

Da bi se mogle dohvaćati varijable iz drugih skripti, pošto je to bitan dio kôda, trebaju se te određene skripte postaviti kao globalne skripte. Postoji opcija Autoload ili Singletoni koji nam to omogućuju, a jednostavno je za napraviti. Pod Project – Project Settings ima kartica AutoLoad gdje se u path opciji jednostavno nađe skripta koja se želi staviti kao globalna skripta da bude dostupna cijelom glavnom čvoru i dodamo ime toj globalnoj koje je uvijek ime same skripte (slika 50).



Slika 50: Način dodavanja globalne skripte

U skripti Counter se gleda kada se povećava broj novčića i bodova, kao i kada se prelazi na sljedeći nivo, kada je igrač izgubio i kada je ostvaren novi najbolji rezultat. U skripti od objekta novčić se nalazi signal `CoinCollected` koji se šalje upravo u skriptu Counter tj. `Objekt` Coin i objekt `Counter` su povezani i prilikom skupljanja novčića funkcija u skripti objekta `Coin` šalje signal (opisano prije u poglavlju o implementaciji novčića) koji će aktivirati upravo ovu funkciju u skripti objekta `Counter`. (ispis 10)

```
extends Label


var coins = 0


func _ready():
	text = String(coins)



func _on_coinCollected():
	coins = coins + 1
	Score.coins = Score.coins + 1
	_ready()
	if Score.coins == 25:
		yield(SilentWolf.Scores.get_high_scores(),
"sw_scores_received")
		Score.score += Score.candy * 200 + Score.time_left * 2
		if Score.score < SilentWolf.Scores.scores[0]["score"]:
			get_tree().change_scene("res://YouWin.tscn")
		else:
			get_tree().change_scene("res://NewScore.tscn")
	else:
		if coins == 5:
			Seconds.secondsTimer.stop()
			Score.time_left += Seconds.seconds
			get_tree().change_scene("res://Level_" +
str(int(get_tree().current_scene.name) + 1) +  ".tscn")
			Seconds.seconds = 60
			Seconds.secondsTimer.start()
```

Ispis 10: Kôd čvora Coin

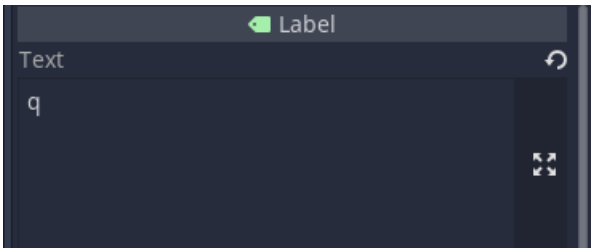
Funkcija `_on_coin_Collected` prvo povećava broj novčića čija se varijabla nalazi u samom kôdu i to na početku kôda, postavljena na nuli. Također se povećava broj novčića u globalnoj skripti Score, gdje je kako je prije navedeno vrijednost uvijek nula prilikom nove igre (zajedno sa životima, vremenom, imenom igrača itd.). Ideja ovdje je ta da se na taj način prati kada je igrač skupio dovoljno novčića da pređe na sljedeći nivo i kada je završio skroz igru. Pošto je za svaki nivo potrebno 5 novčića za prelazak na sljedeći, sa `if` naredbom provjeravamo prvo u globalnoj varijabli da li je broj novčića 25, što bi značilo da je igrač prošao svih 5 nivoa skupivši na svakom po 5 potrebnih novčića. U slučaju da jest, pomoću plug-ina SilentWolfa uzimamo najveći spremljeni rezultat i zbrajaju se sveukupni bodovi od trenutnog igrača na temelju preostalih života, vremena i novčića. Gdje se nakon toga uspoređuje taj glavni rezultat prijašnjih igri i rezultat trenutnog igrača kako bi se provjerilo da li je postignut novi high score. U slučaju da je igrač osvoji prvo mjesto na bodovnoj ljestvici prebacit će se na scenu `NewScore`, u suprotnom će biti scena `YouWin`.

Ako ipak igrač nije skupio svih 25, to znači da još igra i da se mora pratiti kada ide na sljedeći nivo. To se radi tako što se uspoređuju novčići trenutne skripte (Counter), koja je na svakom sljedećem nivou ponovno nula baš zbog toga što je na vrhu koda tako deklarirana i jer se na svakom nivou ponovno pokreće kôd pa tako i sami broj skupljenih novčića. Ako je jednak 5 to znači da je skupljeno dovoljno za prelazak na sljedeći nivo. Tada se ulazi u `if` gdje se zaustavlja vrijeme (da bi se uhvatilo točno vrijeme koje je preostalo igraču i kasnije se ponovno pokrenulo sa novim početnim vremenom za sljedeći nivo), i zbraja se vrijeme preostalo na svakom nivou kako bi se na kraju dobile sveukupne sekunde preostale na svakom nivou. Nagrada je što je veći ostatak vremena to se dobiju veći bodovi. Zatim se mijenja nivo tj. scena tako što se uzima ime trenutne scene i samo se poveća za jedan, jer su scene nivoa nazvane `Level_1`, `Level_2` itd. Nakon toga se vrijeme postavi ponovno na 60 tako što se preko globalne skripte Seconds pristupi i pokrene se ponovno timer.

Ovdje je najbitnije kada se mijenja nivo i kada je igra gotova, što je ostvareno sa dvije varijable namijenjene za istu stvar samo što jedan traje kroz cijelu igru, dok se druga postavlja natrag na nulu nakon svakog prelaska na novi nivo.

Također je bitno da funkcija `ready` prikazuje pravilno broj novčića skupljen za trenutni nivo pa se kroz nju prikazuje na način da u opciju naziva text spremimo trenutni

broj novčića i on će se pokazati svaki put kada se pozove funkcija `ready`, a to je svaki put kada se skupi novčić tj. kada se emitira signal `coinCollected`. (slika 51)



Slika 51: Tekstualni dio čvora Label

Kod skripte čvora `LifeCounter` imaju tri funkcije. Funckija `ready` radi isto što i kod objekta `Conuter`, prikazuje stanje života na zaslonu kroz igru, dok ostale dvije funkcije su zapravo konekcije na čvor `candy`. Signali koje čvor `candy` šalje prilikom skupljanja života (`candyCollected`) i prilikom sudaranja neprijatelja i igrača (`candyLost`) povezuju zapravo ta dva objekta.

Funkcija `_on_candyCollected`, koja je prikazana u kôdu dole, iz globalne skripte Score uzima vrijednost života i povećava za jedan, zatim sa funkcijom `ready` se prikazuje novi broj života. Dok druga funkcija oduzima život za jedan gdje nakon provjerava da li je broj života jednak nuli gdje se onda mijenja scena na scenu `GameOver` označavajući da je igrač izgubio sve živote i izgubio igru. U suprotnom se samo poziva `ready` funkcija kako bi se pokazao smanjeni iznos života.

```
extends Label



func _ready():
	text = String(Score.candy)



func _on_candyCollected():
	Score.candy = Score.candy + 1
	_ready()




func _on_candyLost():
	Score.candy = Score.candy - 1
	if Score.candy == 0:

	get_tree().change_scene("res://GameOver.tscn")
	else:
		_ready()
```

Ispis 11: Kôd čvora LifeCounter

Čvor `ScoreCounter` (ispis 12) služi za prikaz sveukupnih bodova na temelju skupljenih novčića, tako da prima signal `coinCollected` iz čvora `Coin`, pošto su povezani, i aktivira funkciju `_on_coin_coinCollected` što označava da je igrač skupio novčić i broj bodova u globalnoj varijabli se povećava za sto bodova. Upravo zato jer se radi o globalnoj varijabli nikad se neće gubiti vrijednost bodova, samo na kraju kada se pokrene nova igra i u kôdu se vrati na nula bodova. Na taj način se može kroz cijelu igricu prikazati isti rezultat bodova.

```
extends Label



func _ready():
	text = String(Score.score)



func _on_coin_coinCollected():
	Score.score += 100
	_ready()
```

Ispis 12: Kôd čvora ScoreCounter

Čvor `control` je kao što i sama riječ kaže zaslužan za držanje svega pod kontrolom, nadgledanje bodova, koliko igrač ima života, koliko je skupljeno novčića, prikazivanje istih i preko njega se sve to zbraja/oduzima tako da je na neki način najbitniji dio.

4 Zaključak

Ovim projektom je predstavljen rad unutar sustava Godot. Opisano je na koji način je projekt strukturiran i sama implementacija kôda, na koji način su objekti i čvorovi povezani i kako funkcioniraju međusobno. Također su objašnjene funkcije koje Godot pruža kao što su signali, autoload (globalne varijable), scene, gdje su signali pokazani kao vrlo dobar način povezivanja funkcija i objekata, a autoload kao jednostavan način uporabe nekih varijabli iz drugih čvorova pa čak i funkcija jednostavnim dodavanjem istih u poseban file.

Pokazano je kako je vrlo lako proširiti igricu i dodati više nivo samo sa dupliciranjem scena i promjenom vizualnog dijela kako bi se svaka razina razlikovala na svoj način. To se naravno može ostvariti ako se na pravi način rasporedi sami kostur igre i da sve ima svoj slijed i zadaću.

Sve u svemu Godot je jako jednostavan pogonski alat, pogotovo za početnike gdje je sve jednostavno prikazano i ne traži puno znanja osim dijela sa programiranjem. Tako da što se vizualnog dijela, stvaranja poligona i samog izgleda igrice tiče, može se vrlo lako shvatiti i mogu se stvoriti puno veće stvari sa malo više znanja. Dok za programerski dio je opet potrebno malo više znanja kako bi kôd bio što bolje raspoređen i jednostavan za shvatiti, bez nepotrebnih funkcija i skripta i lakim nadograđivanjem same igrice.

U ovom projektu nisu obrađene kompleksne teme kao što je kompletna izrada modela i slično zato što je projekt izrađen s onim mogućnostima Godota koje su ponuđene svakom korisniku bez obzira na iskustvo u izradi igara. To se odnosi na preuzimanje gotovih modela sa interneta s dodacima gdje su svi preuzeti modeli besplatni.

Projekt je izrađen uz pomoć https://www.youtube.com vodica, gdje su pronađene informacije za izradu kompleksnijih elementa igre. A za snalaženje unutar Godot sučelja je korištena službena dokumentacija.

Literatura

[1] Godot Documentation, https://docs.godotengine.org/en/stable/ (posjećeno 8.9.2021.)