IZRADA 2D IGRE U PROGRAMSKOM ALATU GODOT

Džale, Dario

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

DARIO DŽALE

Z A V R Š N I R A D

IZRADA 2D IGRE U PROGRAMSKOM ALATU GODOT

Split, rujan 2020.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Predmet: Mobilne tehnologije

Z A V R Š N I R A D

Kandidat: Dario Džale

Naslov rada: Izrada 2D igre u programskom alatu Godot

Mentor: Marina Rodić, predavač

Split, rujan 2020.

SADRŽAJ

SAŽETAK

Cilj ovog završnog rada je izrada *RPG* (engl. *role-playing game*) igre u pokretaču igre Godot 3.2. prema dogovorenim smjernicama same teme.

Ovaj završni rad prolazi kroz osnovne i složenije koncepte izrade 2D/3D igre u izometrijskom prostoru, s posebnim naglaskom na korištenje pokretača igre Godot 3.2. koji je relativni novitet u svijetu izrade igara. Godot je konkurentan 2D pokretač igara, a 3D mogućnosti su u uzlaznoj putanji, te ga svaka nova verzija približava komercijalnim konkurentima poput Unitya te Unreala. Elementi igre su pretežno 2D, dok su likovi i dodatni elementi kreirani u 3D alatu te ubačeni u 2D okruženje.

Igra je konceptualno klasični *RPG*, igra uloga s pripadajućim elementima koje obično prate *RPG* žanr kao što su: inventar, prozor plijena, dijalog, igrač i računalno pokretani likovi (engl. *non-player character*), prikupljanje iskustva, mehanika borbe itd. Od tehnika izrade pokazano je više stavki, izrada logike i mehanike pojedinih elemenata, grafička izrada i dorada, kreiranje čestičnih efekata (engl. *particles effect*), dodavanje zvuka itd. Rezultat rada ogleda se u završenoj *RPG* igri koju je moguće pokrenuti na Windows operativnom sustavu.

Ključne riječi: pokretač igre, Godot, izrada igre, *RPG*

SUMMARY

2D game development in Godot

The main goal of this master thesis is to create an RPG game in the Godot 3.2. Engine according to the defined guidelines. This paper shows basic and complex concepts of 2D/3D game making, in isometric space with special emphasis on the use of Godot 3.2. Engine, which is a relative novelty in the world of game making. Godot is a competitive 2D game engine, with 3D capabilities improving every day, and each new version brings it closer to commercial competitors such as Unity and Unreal game engines.

The elements created for the game are mostly 2D, while the characters and additional elements are created in 3D and imported into 2D environment. The game is conceptually a classic RPG (role-playing game) with elements that usually accompany the RPG genre such as: inventory, loot panel, dialogue, player and non-player characters, gathering experience, combat mechanics, etc.

Most elements of the game are shown, with the accompanying creation of logic and mechanics of individual elements, graphic design and finishing, creating particle effects, adding sound, etc.

The final result is reflected in the completed RPG game that can be started on the Windows operating system.

Keywords: engine, Godot, game making, RPG

1. UVOD

Tema ovog završnog rada je izrada izometrijske 2D igre u Godot pokretaču igre. Na ideju izrade ovog završnog rada autor je došao prvenstveno vođen mišlju o ispoljavanju kreativnog i akademskog duha u novoj tehnologiji van područja s kojima se do sada susreo. Sljedeća misao bila je korištenje alata, koji ima otvorene i besplatne licence, te nije previše poznat i razglašen u svijetu izrade igara. Proučavajući postojeće alate poput Unreala, Unitya i slično, odluka je pala na manje poznat sustav - Godot.

Godot pokretač igre je sustav za razvoj 2D i 3D igara. Dugo vremena bio je gotovo isključivo baziran na 2D elemente, iako je podržavao i 3D proizvode. Godot zajednica je izlaskom verzije 3 počela ulagati u ozbiljniji razvoj i oglašavanje mogućnosti izrade 3D igara. Ono što još uvijek nedostaje za razvoj 3D igara u odnosu na konkurente, je *occlusion culling*. U trenutnom vidnom polju kamere (engl. *frustrum culling*) koje podrazumijeva prikazivanje svih objekata unutar pogleda kamere, *occlusion culling* podrazumijeva neprikazivanje objekata unutar prostora vidnog polja kamere koji nisu vidljivi samoj kameri (primjerice predmet iza zida). Time se opterećenost samog sustava, za pojedine situacije, uvelike smanjuje. Ovakva problematika se izbjegava u 2D svijetu, kao i izometrijskom, kakav se obrađuje u ovom radu. To je bio još jedan od razloga za odluku o dimenziji izrade same igre.

Korišteni jezik, GDScript, je programski jezik koji sintaksom sliči Python jeziku. Bitna razlika je nedostatak funkcija *list comprehensiona* te određenih metoda, posebice vezanih uz riječnike ili nizove. Stoga se bilo potrebno prilagoditi i napraviti kompromise, te preformulirati problem i pronaći rješenje. Rezultat rada je izometrijska *RPG* igra, koju je moguće pokrenuti na Windows operativnom sustavu. Postoji mogućnost pokretanja na Linux operativnom sustavu, no potrebno je pokrenuti zasebnu verziju.

U drugom poglavlju kratko će se opisati korištene tehnologije, njihova povijest te svrha.

U trećem poglavlju opisan je praktični dio. Prvo se opisuje radna okolina i funkcionalni elementi, a potom se kreće u razlaganje svih bitnijih elemenata koji tvore samu igru.

To uključuje glavni izbornik i pripadajuće elemente, *singletone*, te glavni čvor igre sa svim elementima koji se vide te se mogu pozvati u igri. Dodatno su napisane smjernice za brzu izradu sadržaja i prebacivanja iz Blender alata u Godot.

Četvrto poglavlje predstavlja zaključak rada.

Svi izrađeni, prilagođeni te preuzeti sadržaji popisani su u listi resursa.

2. KORIŠTENE TEHNOLOGIJE

2.1. Godot

Godot pokretač igre, razvojno je okruženje namijenjeno izradi igara, dostupno kao projekt otvorenog kôda pod MIT licencom. Prva verzija razvijena je krajem 2007. godine pod vodstvom Juan Linietskyoga i Ariela Manzura. Podržava 2D i 3D elemente, nativni jezik GDScript (sličan Python programskom jeziku), C++, C# te vizualno skriptiranje.

Pokretač igre je moguće pokrenuti na Windows, macOS, te Linux operativnom sustavu. Stvorene projekte (igre) moguće je izvesti (engl. *export*) s nekoliko klikova mišem (i automatskim dohvaćanjem datoteka) za Windows, macOS, Linux, Android, iOS te Web. Sam Godot ne zahtijeva instalaciju, već se skine potrebna verzija kao izvršna datoteka. Sam pokretač igre ima sedamdesetak megabajta pa je veoma praktičan i jednostavan za brzo pokretanje i promjenu verzija.

Značajnu karakteristiku Godota predstavljaju čvorovi i scene. Jedan čvor, gradivni je element određenog tipa, koji ima ime, promjenjiva svojstva, metode i funkcije kao i povratne funkcije za izvršavanje svake sličice (engl. *frame*). Može se naslijediti te biti dodan kao dijete (engl. *child*) drugom čvoru ili sceni. Scena je hijerarhijski slijed jednog ili više čvorova.

Sastoji se od korijenskog (engl. *root*) čvora, može biti instancirana (engl. *instance*) i dodana u svijet, te spremljena i vraćena s diska.

Fleksibilnost stvaranja elemenata, njihovo mijenjanje „u letu“, ubacivanje i micanje iz svijeta, jedna je od glavnih prednosti korištenja Godota, koja omogućuje brže stvaranje rezultata i završnog proizvoda.

2.2. Blender

Za potrebe izrade ili dorade većine objekata u igri, korišten je alat za 3D obradu i modeliranje - Blender. Blender je potpuno besplatan alat, koji je mogućnostima u razini svih komercijalnih alata slične niše. Potencijalna mana Blendera je, što je dugo bio veliki jaz u početnom učenju i korištenju sučelja koje nije bilo intuitivno. Međutim, Blender je od verzije 2.8. prešao na potpuno redizajnirano sučelje koje je puno intuitivnije, modernijeg izgleda i unaprijeđenih funkcionalnosti. Nova verzija trenutno pati od manjka video materijala za učenje.

2.3. Krita

Krita je program otvorenog kôda za crtanje i uređivanje fotografija (bazirana na pikselima). Program je besplatan, a većina funkcionalnosti temelji se na potrebama 2D animacije i digitalnog slikanja. Prva verzija programa izašla je 2005. godine.

Alati i funkcionalnosti uvelike sliče onima GIMP-a i Photoshop-a, a podržani su Windows, Linux te macOS operativni sustavi.

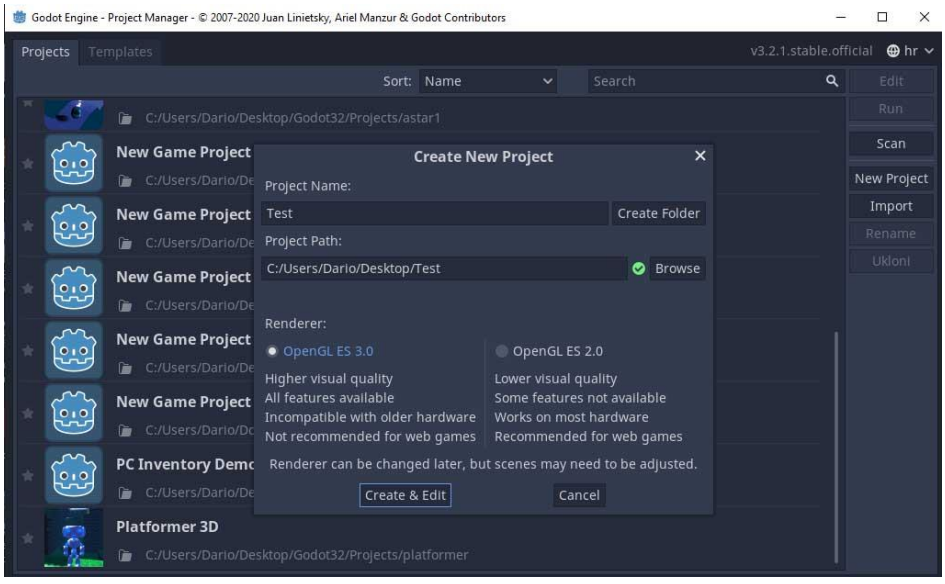
2.4. Mixamo

Mixamo je tvrtka pod okriljem tvrtke Adobe, koja ima *online* platformu za automatsko animiranje likova. Podržava animiranje dvonožnih čovjekolikih likova, a usluga je besplatna i može se koristiti u komercijalne svrhe u sklopu igara ili proizvoda. Nije dozvoljena preprodaja samih animacija i likova s animacijama. Usluga funkcionira preko prijave na besplatan Adobe račun, gdje se željeni 3D lik učita u Mixamo platformu. Nakon odabira animacija i konfiguracije opcionalnih parametara, rezultat se sprema te dalje koristi za potrebe pojedinog projekta.

3. PRAKTIČNI DIO

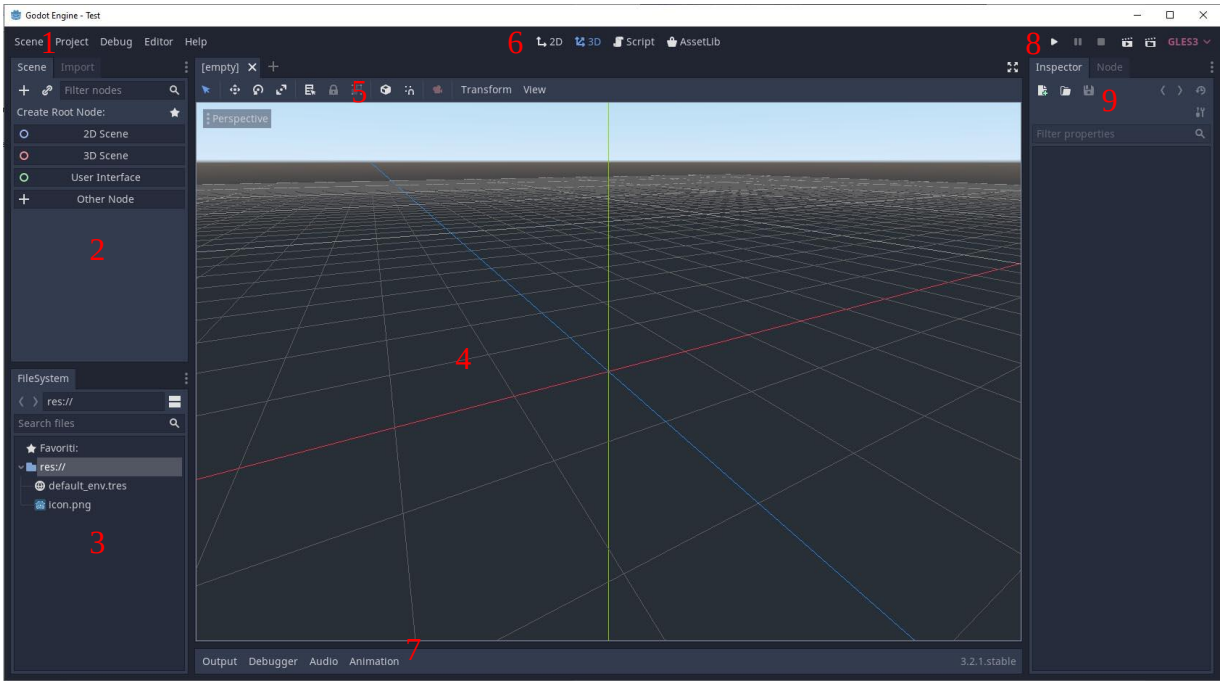
3.1. Upoznavanje s Godot okolinom

Prilikom prvog ulaska u pokretač igre Godot, otvori se početni prozor gdje se bira stvaranje novog projekta. Potrebno je definirati ime projekta, putanju te odabrati iscrtavač (engl. *renderer*) verzije OpenGL ES 3.0. ili 2.0.



Slika 1: Početni prozor pokretača igre Godota

Nakon toga se otvori glavni prozor okruženja Godota.



Slika 2: Glavni prozor okruženja Godota

U krajnjem gornjem lijevom kutu (1) nalazi se glavna traka s izbornicima za scenu, projekt, *debug* opcijama, opcijama urednika (engl. *editor*) te prozorom pomoći.

Ispod (2) se nalazi element za kreiranje čvorova, te hijerarhiju cijelog stabla odabranog korijenskog čvora (prikazuje se tek nakon što se odabere ili kreira čvor). Na poziciji (3) nalazi se prozor s podatkovnim sustavom projekta, pripadajućim datotekama, mapama i generalno svim spremljenim resursima (skripte, scene, materijali itd.).

Sredinu ekrana (4) zauzima prostor manipulacije 2D ili 3D elementima i svijetom – ***Viewport***. U gornjem kraju scene i ***Viewport*** prozora (5) je alatna traka za odabranu scenu ili element. Iznad ***Viewport***-a (6) nalaze se ikone za promjenu vrste prikaza svijeta (2D ili 3D), te odabira skripte ukoliko se želi dobiti pregled svih skripti ili ih mijenjati. Ispod ***Viewport***-a (7) nalazi se donji prozor s karticama *Output*, *Debugger*, *Audio*, *Animation* koje pružaju izliste zapisa igre koja se pokrene, mogućnosti debugiranja i praćenja procesa kao i mogućnosti izrade određenih elemenata zvuka, animacije itd. (ukoliko odabrana scena ima tu mogućnost).

U gornjem desnom kutu (8) je traka s ikonama za pokretanje glavne, trenutne scene, zaustavljanje te odabir preglednika. Ispod (9) se nalazi inspektor (engl. *inspector*) odabranog čvora sa svim opcijama navedenog elementa, te druga kartica čvor (engl. *node*) u kojoj se mogu dodjeljivati grupe te spajati signali odabranog elementa.

U prozoru (2) odabere se opcija ovisno o sceni koju se želi stvoriti. Primjerice, za 2D grafički element koji će biti dio sloja platna, odabere se opcija ***User Interface***. Čvor ***User Interface*** nasljeđuje karakteristike vrste čvora ***Control***, koji je vršni čvor za elemente grafičkog sučelja. Sada se na novostvoreni čvor mogu dodavati drugi čvorovi kao djeca, te svakom čvoru po želji „zakačiti" skriptu. Kada se završi s dodavanjem skripti, čvorova itd. unutar scene, scena se spremi te se pohrani u resurse projekta. Potom se može instancirati negdje drugdje u projektu ili koristiti kao samostalna scena, ovisno o potrebama.

Na kraju se postavlja pokretanje glavne scene (slijeda) koja čini prirodni tok igre, kako bi ulaskom u igru bila otvorena definirana scena.

3.2. Pregled funkcionalnih elemenata igre

Glavni čvor, na koji je moguće spajati i odspajati scene igre, je ***SceneTree***. To je čvor kojem se ne pristupa u kontekstu kreiranja ili skriptiranja, već služi kao proizvoljna točka s koje se može doći do bilo koje druge putanje ili scene u igri, te joj iz bilo koje scene pristupiti. Sadrži korijensku ***Viewport*** scenu, kojoj se pristupa s `get_tree().get_root()` i koja se nalazi na vrhu otvorene scene. Dodatno sadrži informacije o grupama i globalne mogućnosti poput pauziranja.

Potrebno je postaviti scenu koja se prva pokreće prilikom pokretanja igre. Prvi vidljivi element koji se pokreće je dio s glavnim izbornikom koji se sastoji od nekoliko elemenata. U pozadini se pokreću *singleton* čvorovi koji su uvijek prisutni kao djeca glavnog korijenskog čvora, bez obzira koja zasebna scena je trenutno postavljena kao aktivna.

Kada se odabere neka od opcija glavnog izbornika (primjerice ***New game***), glavni izbornik se miče iz trenutnog statusa aktivne scene te se učitava scena svijeta igre sa svim elementima (igrač, neprijatelji, predmeti, okoliš…).

Bitno je napomenuti da je većina elemenata funkcionalno povezana upotrebom grupa i signala. Detaljnije o tome biti će rečeno tijekom procesa opisa pojedinih elemenata rada. Također je bitno spomenuti neke funkcije koje se mogu pronaći u svakom čvoru tj. pripadnoj skripti. Svaka skripta sadrži `ready()` funkciju. Ovo je funkcija koja se pokreće prilikom dodavanja scene ili čvora u aktivnu scenu. Pored `ready()` postoje neke bitne funkcije koje se pokreću za svaku sličicu ovisno ili neovisno o fizikalnom procesu. To su:

- `_process(delta)` – funkcija se izvršava za svaku sličicu u sekundi (engl. *frames per second* - *FPS*) aplikacije. Ovisna je o složenosti sloja za iscrtavanje i performansama uređaja na kojem se vrti, stoga nije uvijek jednaka. Zbog toga se koristi `delta` parametar koji podrazumijeva vrijeme proteklo od zadnjeg poziva ove funkcije. Tako se može podesiti fiksno vrijeme koje će biti jednako neovisno o tome koliko brzo se sličice aplikacije vrte. Sama funkcija nije sinkronizirana sa sustavom fizike.
- `_physics_process()` – funkcija se vrti u fiksnim vremenskim intervalima (moguće je ručno postaviti interval u ***Project Settings -> Common -> Physics FPS***, nativno je 60 po sekundi) i izvršava se uvijek prije akcije sustava fizike.

3.3. Glavni izbornik

Prvi element prilikom pokretanja izvršne datoteke igre je ulazni video slijed s ikonom maskote Godota. Sam element animiran je sjenčarom (engl. *shader*). O sjenčarima će biti riječi u sljedećim poglavljima.

Slijedi klasični glavni izbornik s nekoliko dugmadi. S desne strane je slika na kojoj su korišteni sjenčari te čestični efekti. Dostupna je opcija odabira nove igre, nastavka, opcija (nema trenutnih funkcionalnosti), te izlaza. Klikom na novu igru, ili nastavljanje postojeće, igrača se preusmjeri na novu scenu, tj. slijed scena. Izlaz poziva `get_tree().quit()` te se izlazi iz igre.

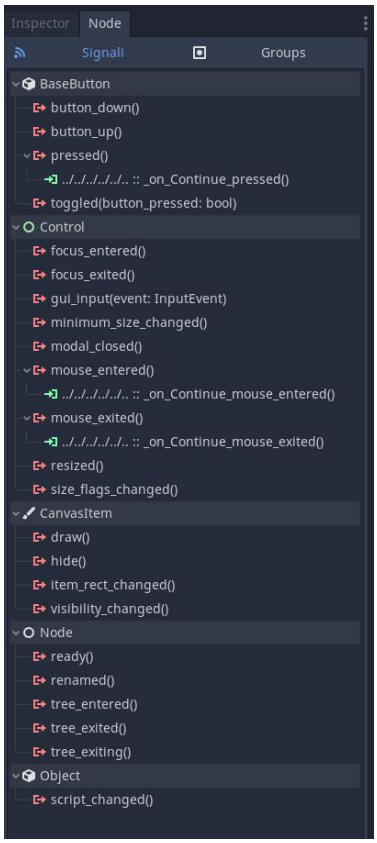
Slika 3: Glavni izbornik igre

3.3.1. Dugmad

Prvi funkcionalni elementi igre su dugmad na glavnom izborniku. Oni su elementi tipa ***TextureButton*** s postavljenim teksturama za ***„Normal“***, ***„Pressed“*** te ***„Hover“*** stanje. Koriste signale kako bi povez ali određenu interakciju s određenom funkcionalnošću.

Signali predstavljaju uzorak dizajna ***Observer*** (engl. *design pattern*). To objektima unutar okruženja Godota omogućuje osluškivanje signala, koji šalje neki drugi objekt ili događaj. Signali mogu biti predefinirani od strane pojedinih tipova objekata u Godotu, a mogu se kreirati i vlastiti signali, koje je potom potrebno povezati s osluškivačem signala.

Korišteni predefinirani signali za dugmad su: `pressed()`, `mouse_entered()`, `mouse_exited()`. Uvjeti okidanja i pozivanja funkcije u skripti na koju su spojeni su vidljivi iz naziva samog signala.



Slika 4: Prozor signala

Signali se ovdje se spajaju na najviši roditeljski čvor, u ovom slučaju tipa ***Node2D***. Unutar roditeljskog čvora definiraju se i dodatni signali `existing` te `new` u `ready()` funkciji, koji će služiti kao uvjet pokretanja učitavanja ili pokretanja nove igre. Signali se spajaju sa *singletonom* `ImportData`, o kojem će biti riječi nešto kasnije. Nakon odabira ***"New game"*** (ili ***"Continue"*** ukoliko postoji već spremljena igra) mijenja se trenutna scena preusmjeravanjem u ***SceneTree***, kako je pojašnjeno u sljedećem poglavlju.

3.3.2. Preusmjeravanje

Kada se igra nalazi na određenom čvoru s aktivnim scenama, nekada je potrebno prijeći na novu scenu (učitati je kao aktivnu). U slučaju glavnog izbornika dugmad ***"New game"*** ili ***"Continue"*** prebacuju trenutno učitanu scenu (glavni izbornik) u ***SceneTree*** na scenu ***Main.tscn*** koji sadrži elemente svijeta igre, sudionike i sve ostale sadržaje. Koristi se:

```
get_tree().change_scene("res://Main.tscn")
```

Ova metoda će izbrisati iz memorije postojeću scenu, te će učitati navedenu novu scenu s pripadajućim podscenama i elementima. U pozadini se koristi `get_deffered()` za sigurno uklanjanje, pošto je potrebno osigurati da se elementi ne koriste od strane fizikalnog svijeta.

3.3.3. Sjenčari

Sjenčar je program koji se izvršava na grafičkoj kartici te manipulira pikselima. Koriste se matematičke funkcije kako bi se izgled nekog materijala promijenio i prikazao na drugačiji način.

U okruženju Godot mogu se raditi vlastiti sjenčari, koji nadopunjuju i mijenjaju izgled željenog materijala. Jezik koji se koristi za pisanje sjenčara sličan je GLSL jeziku koji je sličan C++ jeziku.

Nakon stvaranja elementa, kojem se želi pridjeliti materijal i dodati sjenčar, proces kreiranja sjenčara je sljedeći: odabere se element (npr. ***Sprite*** element) te mu se doda novi materijal i sjenčar. Sada se klikom na novostvoreni sjenčar otvori prozor za pisanje kôda. Prvo je potrebno definirati tip sjenčara, a on može biti:

- ***spatial*** za 3D objekte,
- ***canvas_item*** za 2D objekte i ***Control*** elemente,
- te ***particles*** za čestični sustav.

Iza toga se potencijalno može definirati ***render mode***.

Zatim se bira jedan od tipova vrste procesa sjenčara, a to su:

- `Vertex()` - funkcija koji mijenja svaki vrh (točku ili čvor) objekta,
- `Fragment()` - funkcija koja se izvršava za svaki piksel objekta,
- `Light()` - funkcija koja upravlja načinom kako se svjetlo aplicira na objekt.

Poslije definiranja navedenih tipova i funkcija slijedi pisanje kôda. Nakon pisanja kôda potrebno je spremiti sve stvorene resurse (materijal i sjenčar). Rezultati kôda sjenčara vide se u ***Viewport***-u u stvarnom vremenu.

Igra sadrži nekoliko sjenčara:

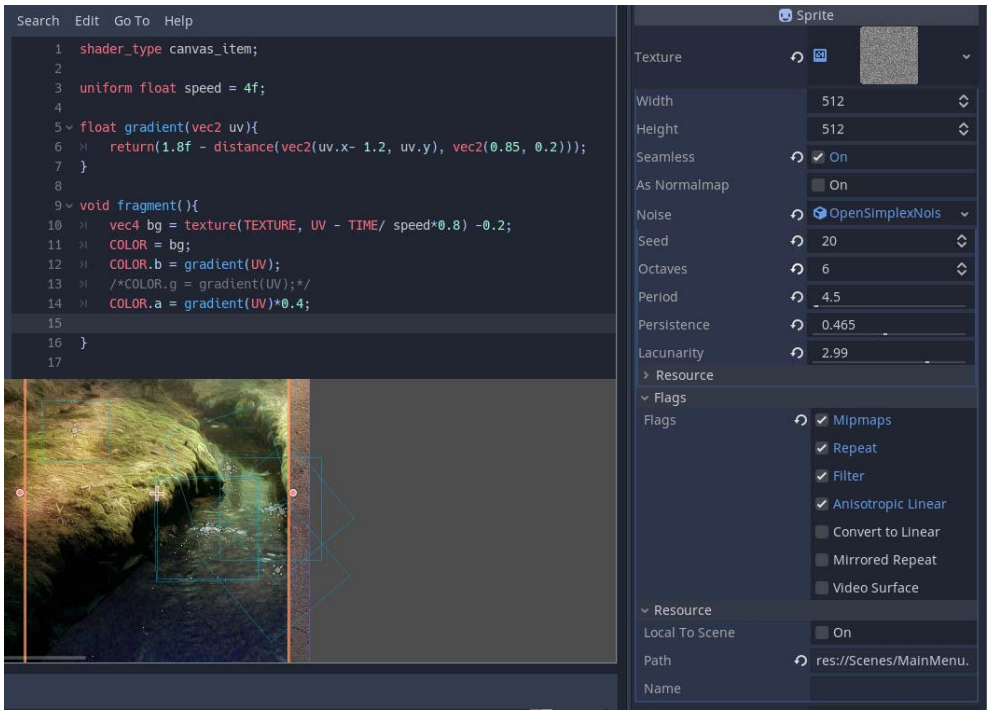
- ***Intro_splash -> TextureRect -> Sprite*** – pojavljivanje i nestajanje Godot sličice na ulasku u igru.
- ***MainMenu -> Node2D -> TextureRect -> HboxContainer -> CenterContainer -> Sprite*** – simuliranje toka rijeke preko 2D slike na početnom izborniku.
- ***SmallLake1 -> StaticBody2D -> Sprite (small_lake1)*** – simuliranje kamenčića koji lebde, vidljivo u igri na magičnom jezeru kraj sela.

Slijedi pojašnjenje kôda za simuliranje toka rijeke (kôd je vidljiv na slici 5). Rješenje je izvedeno kao tekstura tipa ***noiseTexture*** koja se kôdom i podešenim parametrima pomiče i daje privid toka rijeke.

Podešavanje parametara (*seamless, period, octaves* itd.) se pretežno radi vizualno i intuitivno. Kroz kôd se doda varijabla *uniform*, kako bi se mogla proslijediti interaktivno sjenčaru kroz inspektor.

Slijedi funkcija *gradient,* koja prihvaća *vector2* kao ulaz, a vraća prilagođenu *vector2* vrijednost. Zatim *fragment* funkcija, budući da je potrebno mijenjati cijelu ***noise*** teksturu. Definira se *vec4* varijabla koju čini čitanje 2D teksture s dvije varijable. Prva je trenutno učitana tekstura *TEXTURE*. Druga je prilagođeni *UV* dio za *vec2 uv* dio funkcije (koristi se *uniform speed* varijabla koja se mijenja kroz urednik kao ubrzivač ili usporivač vode).

Potom se definirana varijabla *vec4 bg* pridjeli u *COLOR* ugrađenu varijablu te se za dijelove vektora *COLOR*, koji predstavljaju *b* – plavi kanal te alfa kanal, pozove *gradient* funkcija. Prvo se pozove s parametrom *UV* (*vec2* vrijednosti od 0 do 1), a zatim za alfu s *gradient UV* parametrom, koji zatamni teksturu s „kidanjem“ množenjem s vrijednošću 0.4.



Slika 5: Kôd sjenčara za animaciju rijeke te parametri inspektora

3.3.4. Čestični objekti

Čestični (engl. *particles*) objekti se pretežno koriste kao vizualni elementi za unaprijeđenje grafike igre i izradu određenih efekata. U primjeru ovog rada, čestični efekti koriste se na više mjesta: svjetleće čestice na slici šume u glavnom izborniku, mjehurići u rijeci na istoj slici u glavnom izborniku, krijesnice u selu u svijetu, te za simulaciju slapova u svijetu kod jezera.

Postoje tri vrste čestičnih objekata:

- ***Particles2D*** – 2D čestični objekt s punim performansama.
- ***ParticlesCPU2D*** – procesorska verzija 2D čestičnog efekta se izvršava na procesoru i pogodna je za slabija računala (npr. bez dodijeljene grafičke kartice).
- te ***Particles*** – prostorni (engl. *spatial*) 3D objekt za čestice.

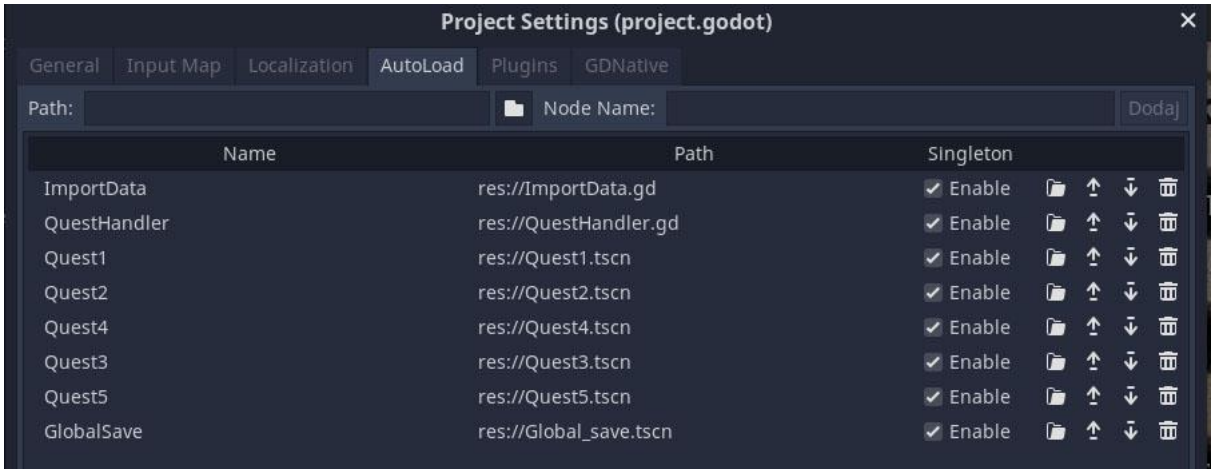
Generalni princip kreiranja čestičnog 2D objekta je: stvori se novi objekt tipa ***Particles2D***, postavi broj čestica, emitiranje, vrijeme i postavke trajanja. Zatim se doda novi čestični materijal, te unutar njega postavljaju razni parametri ovisno o željenom efektu. Moguće je definirati prostor gdje će se očitovati čestični efekt, smjer kretanja čestica, dodati razinu gravitacije, dodatne razne vrste brzine (startno, kutno, kružno) te ubrzanja (linearno, radijalno, tangentno itd.). Također se mogu mijenjati dimenzije čestica, njihova raznovrsnost, boja, vrijeme nakon kojeg čestice gube brzinu i slično. Namještanje parametara je pretežno intuitivno u skladu s željenim rezultatom, a prikaz rezultata se vidi trenutno u ***Viewport*** prozoru.

Slika 6: Parametri za pojedini čestični efekt simulacije prskanja vode u rijeci

3.4. Singletoni

Singleton čvorovi, su objekti koji se pokreću s pokretanjem same igre, i ne ovise o trenutno aktivnoj sceni. Oni su globalno dostupni u svakom trenu stanja i aktivne scene igre, te je ova značajka veoma bitna za mnoge funkcionalnosti izrade igara u Godotu.

Postavljanje *singletona* vrši se preko sučelja pokretača igre Godot odabirom ***Project -> Autoload***, te se dalje izabere skripta koja je prije stvorena i spremljena bilo gdje unutar `res//:` prostora.



Slika 7: Prozor za učitavanje *singletona*

Varijablama, funkcijama i metodama koje su definirane unutar *singletona* može se pristupiti iz bilo koje druge scene, skripte i dijela igre. Za potrebe ove igre stvoreno je nekoliko *singletona*, a svaki će biti ukratko pojašnjen kroz pripadajuće funkcionalnosti i razloge kreiranja.

3.4.1. ImportData

Ovaj *singleton* služi kao glavna točka dohvaćanja i učitavanja podataka iz spremljenih datoteka (JSON formati), kao i za dohvaćanje i pohranu trenutnih varijabli svijeta za potrebe spremanja stanja igre. Neke od tih varijabli su informacije o blagu, stvarima u inventaru, riješeni, aktivni i neotvoreni zadaci, pozicija i zdravlje igrača, neprijatelja, informacija o danu ili noći itd.

Prilikom učitavanja čita generičke datoteke iz `res://Data` lokacije same igre, te ovisno o odabiru ***New game*** ili ***Continue*** dugmeta mijenja varijable svijeta. Tako za odabir ***New game***, prihvaća signal iz glavnog izbornika za pokretanje funkcije `new()`, u kojoj učitava postavke za novu igru. Za ***Continue*** opciju prihvaća signal `continue` iz glavnog izbornika te pokreće funkciju `existing()`, koja čita datoteke iz `user://` lokacije za pohranu spremljenih datoteka igre. Osim učitavanja ovaj *singleton* radi i spremanje preko nekoliko funkcija:

- `on_save_all()` – funkcija u kojoj se dohvaćaju podaci za pohranu inventara, odabranog oružja i magije, informacije dana i noći, liste neprijatelja, stanja zadataka. Sadrži `save_game()` funkciju, koja dohvaća detaljnije podatke objekata u grupi ***Persist***, a to su igrač i neprijatelji s pripadajućim informacijama zdravlja, pozicije itd.
- `load_game()` – funkcija se pokreće iz scene Main.tscn, iz čvora Main, tek kada se Player.tcsn pojavi u stablu, kako ne bi došlo do pucanja prilikom pokušaja referiranja na nepostojeći objekt (nije još dodan u svijet).

3.4.2. GlobalSave

Singleton ***GlobalSave*** čuva određene varijable koje su promjenjive unutar trenutne instance igre, a bitne su za ponašanje svijeta. Točnije, ovaj *singleton* prati varijable dana i noći te shodno tome sprema postojeća čudovišta i neprijatelje ovisno o dobu dana. To se ostvaruje povezivanjem ***Main.gd*** skripte koja prvo spaja signale dana i noći iz ***Main.gd*** na ***GlobalSave*** *singleton*, te potom poziva pripadajuće funkcije ovisno o dobu dana.

Trenutna funkcionalnost spremanja pohranjuje duhove koji izlaze i mijenjaju kosture po noći ovisno o broju postojećih kostura, a spremljene kosture ujutro vraća u svijet neovisno o broju postojećih duhova prošle noći. Tako se izmjenom dana i noći čita lista neprijatelja `GlobalSave.list_of_all_enemies` koji se pojavljuju ukoliko postoje prema prije navedenim pravilima.

3.4.3. QuestHandler

Ovaj *singleton* sadrži liste i riječnike stanja zadataka, tko ih daje, koji zadaci su završeni itd. Upravlja funkcionalnostima dodavanja novog zadatka, promjene stanja postojećeg zadatka, te poziva i mijenja zadatke, ovisno o varijablama koje se mijenjaju u svijetu. Primjerice, skupljen predmet ili ubijen neprijatelj poziva funkciju `quest_death(type)` koja ovisno o pozivatelju prosljeđuje poziv dalje u pojedine zadatke. Završno, prilikom pokretanja samog *singletona* poziva se `reload_quests()` funkcija u `_ready()`, kako bi se ažurirala stanja igračevih zadataka, prilikom svakog otvaranja nove igre ili nastavljanja postojeće.

3.4.4. Quests

Svaki pojedini zadatak (engl. *quest*) je također *singleton*. Uvelike ovisi o pozivima i slanjima kroz ***QuestHandler*** *singleton*. Svaki zadatak sadrži definiranu varijablu tipa neprijatelj, predmet ili događaj, koja uvjetuje pozivanje tj. izvršavanje elemenata. Unutar svakog zadatka su definirani iskustveni bodovi, koji se dodjeljuju pri završenom zadatku. Pored toga, zadatak ima i provjeru jesu li ispunjeni uvjeti za završetak, koji varira od provjere broja ubijenih neprijatelja, do provjere količine specifičnog predmeta u inventaru.

3.5. Glavna scena

Main.tscn, glavni je čvor igre, koji sadrži sve elemente igre osim glavnog izbornika, te *singletonova*. Hijerarhijski gledano sadrži dva ključna elementa. Prvi je sloj platna (tip ***Canvas layer***) u kojem su posloženi svi elementi sučelja koje prate pravila sloja platna i ***Viewport***-a. Drugi element je ***World1.tscn*** scena (***Node2D*** element koji je instanciran kao dijete ***Main.tscn*** scene). Scene koje ne ovise o varijablama okruženja i ponašanju igrača, su većinom već dodane u hijerarhiju sloja platna ili ***World1*** scene.

Druge scene, poput neprijatelja koji se pojavljuju po noći ili inventara koji čeka pritisak slova ***I*** na tipkovnici, se instanciraju i dodaju tek nakon što se određeni uvjet njihovog okidanja pojavi. Postavljaju se na kôdom definirano mjesto u hijerarhiji scena. Pored sloja platna i ***World1*** scene, postoje i dodatni elementi (scene) koje ne pripadaju niti jednom od ova dva elementa, već pretežno služe kao dopunski elementi svijeta ili pojedinih čvorova.

Main.tscn ima dodjeljenu skriptu ***Main.gd***, koja izmjenjuje dan i noć te postavlja neprijatelje u ovisnosti o dobu dana.

Bitnije funkcije su:

- *night_time_timer(time_till_night)* – brojač funkcija s definiranim trajanjem do sljedeće noći. Nakon isteka brojača, poziva funkciju tranzicije u noć *transition_to_night()*.

- *transition_to_night()* – na ***Tween*** čvor koji se nalazi u hijerarhiji: ***Main -> CanvasModulate -> Tween*** definira se metoda interpolacije *.interpolate_property()* sa slijedećim redoslijedom parametara:

    - *$CanvasModulate* – čvor koji se animira,
    - *"color"* – parametar čvora koji se mijenja,
    - *Color(1.0, 1.0, 1.0, 1.0)* – početna varijabla,
    - *Color(0.1, 0.3, 0.36, 1.0)* – završna varijabla,
    - *10* – vrijeme od početka do kraja interpolacije animacije,
    - *Tween.TRANS_LINEAR* – linearna interpolacija animacije,
    - *Tween.EASE_IN* – interpolacija kreće sporo te ubrzava prema kraju.

- *day_time_timer(time_till_day)* – analogija je ista kao i za funkciju *night_time_timer(time_till_night)*.

- *call_off_skeletons()* – dohvaćanje svih neprijatelja preko *.get_nodes_in_group(„Enemy_group")*, te brisanje onih različitih od tipa ***„Ghost"*** s *.queue_free()*. Poziva se funkcija prizivanja bijelih duhova na groblju.

- *summon_white_ghosts()* – dohvaća sve neprijatelje iz *singleton* liste *GlobalSave.list_of_all_enemies* te ih instancira – zasebno bijele duhove u svijetu koji zamjenjuju kosture, zasebno bijele duhove koji se neovisno pojavljuju na groblju.

    Kako bi se igraču omogućilo klikanje na neprijatelja koji se instancira, potrebno je spojiti signal novostvorenog neprijatelja s čvorom i funkcijom igrača:

    ```
    ghost.connect("ENEMY_CLICKED", get_node("World1/YSort/Player"), "enemy_clicked")
    ```

    Dalje se postavlja pozicija duha s:

```
ghost.set_position(Vector2(every_enemy["pos_x"],
                           every_enemy["pos_y"]))
```

te se dodaje u svijet:

```
get_node("World1/YSort").add_child((ghost))
```

Za duhove na groblju radi se iteriranje kroz definirane pozicije na groblju (lista pozicija) te se nasumce određuje broj duhova u definiranom rasponu. Potom se instanciraju, spoje s igračem (funkcija klika), pozicioniraju te dodaju u svijet. Na kraju se poziva funkcija brojača `day_time_timer(time_till_day)`.

- `call_off_white_ghosts()` – analogno funkciji `summon_white_ghosts()`.
- `_on_RichTextLabel_tree_entered()` – funkcija koja prikazuje tekst na početku nove igre. Ukoliko je uvjet `if ImportData.new_game == true` istinit postavlja se uvodni tekst igre u sloj ***Main -> CanvasLayer -> RichTextLabel***. Nakon isteka brojača tekst i labela se skrivaju.

3.5.1. Grafičko korisničko sučelje i sloj platna

Sloj platna (engl. *canvas layer*) ima svoju logiku prikazivanja elemenata kroz slojeve (engl. *layer*) koji nisu kompatibilni sa slojevima 2D tipova čvorova i Z indeksom. Sloj platna je element koji se prikazuje preko ***Viewport***-a u zasebnom 2D sloju, iznad svih drugih elemenata koji su u trenutnoj ***SceneTree*** strukturi. Stoga se često koristi kao roditeljski element, na koji se vežu grafički elementi koje korisnik vidi tijekom cijelog igranja, kao i elementi koji se trebaju pojaviti na vrhu vidnog polja pritiskom na određeno dugme ili kombinaciju tipki – npr. inventar. Za potrebe ove igre, element tipa sloja platna je korišten kao grafičko korisničko sučelje (engl. *graphical user interface* - *GUI*). Slijedi popis i pojašnjenje pojedinačnih elemenata *GUI*-ja.

3.5.1.1. Početni elementi grafičkog korisničkog sučelja

Pri ulasku u igru u ***Viewport***-u vidi se većina elemenata grafičkog korisničkog sučelja (slijeva nadesno): kugla zdravlja, traka vještina (engl. *skill bar*), traka iskustva (engl. *experience bar*), dugme za spremanje igre.

Slika 8: Prikaz elemenata grafičkog korisničkog sučelja

Kugla zdravlja izvedena je kao zasebna scena koja je strukturom ***TextureProgress*** s okvirom ***Sprite***. Iz igračeve skripte prihvaća signal ukoliko se igračevo zdravlje mijenja, te sukladno tome oduzima ili dodaje vrijednost. Vizualna vrijednost zdravlja tako se ažurira pri svakoj promjeni.

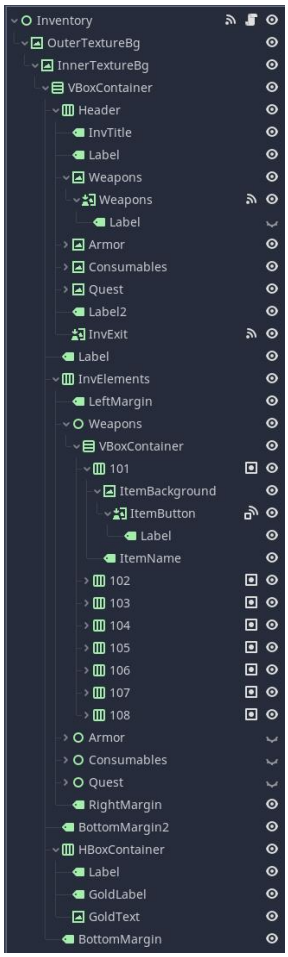
Dugme za spremanje igre šalje signal *singletonu* za spremanje stanja igre i prikazuje poruku ***"Saved game"*** prilikom pritiska na dugme.

Od dodatnih elemenata grafičkog korisničkog sučelja koja nisu stalno vidljiva, koristi se tip čvora ***RichTextLabel*** za uvodni tekst, tip čvora ***Label*** za određene poruke (npr. spremanje igre), inventar s pratećim elemetima za informaciju o predmetu, prozor plijena (engl. *loot panel*) koji se pojavljuje prilikom otvaranja kovčega. Slijedi opis bitnijih navedenih elemenata.

3.5.1.2. Inventar

Scena ***Inventory.tscn*** se instancira u ***CanvasLayer.tscn*** sceni te joj se dodaje kao dijete prilikom pritiska slova ***I*** na tipkovnici.

Scena ***Inventory.tscn*** sastoji se od slijeda elemenata vidljivih na slici 9.



Slika 9: Prikaz strukture elemenata inventara

Struktura inventara slična je strukturi većine elemenata sloja platna kao što su info prozor, prozor plijena itd.

Korijenski čvor je tipa ***Control***, ispod kojeg se prvo slažu različite razine pozadinske grafike. Prva dva čvora ispod su tipa ***TextureRect*** te čine teksturu pozadine inventara. Potom slijedi segmentno dijeljenje prostora inventara kroz vertikalne (***VBoxContainer***) te horizontalne (***HboxContainer***) elemente za slaganje. Pojedini elementi se dodatno prostorno određuju korištenjem sidrišta i margina kako bi djeca pratila roditeljski element. Tako će svaka promjena veličine i pozicije elementa roditelja, uzrokovati praćenje dimenzija i pozicije djeteta, pa nije potrebno ručno mijenjati svaki element ispod roditeljskog čvora.

U elementu naziva ***Header***, nalazi se gornji red ikona, za odabir podizbornika inventara. Odabirom pojedinog dugmeta, pokazuje se pripadajuća hijerarhija s elementima za redni prikaz predmeta u toj kategoriji inventara. Tako imamo hijerarhijski slijed ***InvElements -> Weapons -> VboxContainer -> 101***. Zadnji element (101) je broj mjesta i pozicije predmeta u nizu kategorije inventara. Raspon ide od 101 do 108 za svaku kategoriju, a svaki je spojen u grupu naziva `InventorySlot`. Ispod svakog mjesta (engl. *slot*) se još nalazi slika pozadine, predmeta, te labele s nazivom i količinom predmeta. Slika predmeta tipa ***TextureButton*** spojena je signalom i varijablom na vršni čvor inventara (metoda `_on_ItemButton_gui_input`). Opisano je vidljivo na slici 10.

Slika 10: Prikaz strukture jednog mjesta za predmet tipa *Weapons*

Postoje četiri kategorije predmeta koje je moguće skupiti: oružje (engl. *weapons*), oklop (engl. *armor*), biljke i pića (engl. *consumables*), te predmete zadataka (engl. *quest*).

Prilikom svakog pozivanja inventara poziva se `fill_inventory()` funkcija koja iterira po elementima čvorova u grupi naziva `InventorySlot`:

```
for i in
get_tree().get_nodes_in_group("InventorySlot"):

    itembutton.connect("pressed", self,
    "_on_ItemButton_pressed",
    [itembutton.get_parent().get_parent().get_name()])
```

Na taj način spaja svako dugme s funkcijom `_on_ItemButton_pressed()` koja provjerava postoji li `invslot` (broj mjesta predmeta) u *singletonu* `ImportData` s `ImportData.inv_data.has(slot)`. Ukoliko postoji, instancira se te doda element u desnom kraju odabranog elementa u inventaru, koji se pokazuje kao maleni kotačić (scena ***ItemPopupPanel.tscn***). Dohvaćanje po grupama bitna je funkcionalnost Godota koja se često koristi, jer se iz bilo koje pozicije stabla može dohvatiti pripadnike definirane grupe.

Slika 11: Prikaz kotačiča funkcionalnosti - dodana scena ***ItemPopupPanel.tscn***

Funkcija `_on_Inventory_gui_input(event)` registrira hvatanje prozora inventara lijevom tipkom miša, kako bi se prozor mogao povući.

Slijedi problematika za funkcionalnost pomicanja predmeta po inventaru. Budući da su svi elementi inventara podtipovi čvora Control, nemaju Z indeks manipulaciju slojevima. U čvorovima tipa ***Control*** elementi se iscrtavaju slijedno, tako da posljednje dodan element u hijerarhiji, biva iscrtan na vrhu (iznad ostalih). Kada se "uhvati" predmet, koji je primjerice element na broju 101, te ga hipotetski treba povući do mjesta 108, nastaje sljedeći problem. Zbog redosljeda iscrtavanja, predmet koji se vuče se podvlači pod predmete većeg indeksa, što je vizualno nedopadljivo.

Napravljeno je specifično rješenje problema opisano kroz pojašnjenje funkcije za pomicanje predmeta u inventaru.

_on_ItemButton_gui_input(event, intu) – je funkcija za pomicanje predmeta iz jednog mjesta u inventaru u drugo. Ovo se može koristiti za premještanje predmeta, ali i slaganje više predmeta istog tipa u jedno mjesto. Prvo se provjerava je li kliknuta i držana lokacija predmeta (engl. *slot*) prazna:

```
If
get_node("OuterTextureBg/InnerTextureBg/VBoxContainer/I
nvElements/"+category_selected+"/VBoxContainer/"+str(in
tu)+"/ItemName").get_text()=="":
```

Ukoliko ne postoji, provjerava se je li pritisnut i drži se lijevi klik miša. Zatim se provjerava je li događaj koji se hvata tipa *InputEventMouseButton*. Slijedi provjera:

```
if event.button_index == BUTTON_LEFT
```

koja prati držanje pritisnute lijeve tipke miša. Posljednje se provjerava je li lijeva tipka miša upravo pritisnuta (*if event.pressed*). Ukoliko jest, dohvaća se pozicija gornjeg lijevog kuta elementa kojeg treba pomaknuti:

```
get_node("OuterTextureBg/InnerTextureBg/VBoxContainer/I
nvElements/" + category_selected +
"/VBoxContainer/"+intu).get_position() + Vector2(24,0)
```

Potom se instancira element praznog mjesta inventara (element je scena *res://Scenes/Black_box95.tscn*). Postavi mu se pozicija, ime, te se doda kao dijete lijevoj margini inventara. Sada se novododani element postavlja u pozadinu prikazivanja sa *.set_draw_behind_parent(true)* te se inventar ažurira s *fill_inventory()*.

Popunjava se *drag* varijabla trenutne pozicije miša:

```
get_local_mouse_position() -
get_node("OuterTextureBg/InnerTextureBg/VBoxContainer/I
nvElements/" + category_selected).get_position()-
Vector2(0,node)
```

Slijedi dio:

```
if event is InputEventMouseMotion and drag != null
```

gdje se nakon početnog klika na predmet u inventaru nastavlja vučenje elementa s mišem. Postavlja se pozicija elementa koji se vuče na vrh prikaza (slika 12):

```
get_node("OuterTextureBg/InnerTextureBg/VBoxContainer/I
nvElements/" + category_selected +"/VBoxContainer/" +
intu).set_draw_behind_parent(false)
```

Skriva se naziv predmeta:

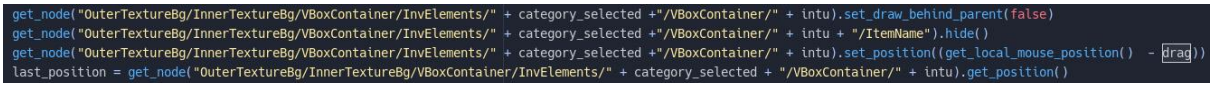
```
get_node("OuterTextureBg/InnerTextureBg/VBoxContainer/I
nvElements/" + category_selected +"/VBoxContainer/" +
intu + "/ItemName").hide()
```

Postavlja se lebdeći predmet na novu poziciju:

```
get_node("OuterTextureBg/InnerTextureBg/VBoxContainer/I
nvElements/" + category_selected +"/VBoxContainer/" +
intu).set_position((get_local_mouse_position() - drag))
```

Posljednje se sprema lokacija zadnje pozicije elementa:

```
last_position =
get_node("OuterTextureBg/InnerTextureBg/VBoxContainer/I
nvElements/" + category_selected + "/VBoxContainer/" +
intu).get_position()
```



Slika 12: Prikaz slijeda za vučenje predmeta mišem

Potom se prati prethodno spomenuti dio za provjeru pritiska miša (*`if event.pressed`*). Budući da je lijevi klik miša već prije pritisnut, a sada se samo drži, funkcija ulazi u dio *`else`* umjesto u *`event.pressed(true)`*.

Unutar *`else`* prvo se briše crna podloga koja se smjestila na mjesto pomaknutog predmeta. Vraća se iscrtavanje redoslijeda povučenog predmeta:

```
.set_draw_behind_parent(true)
```

na početnu hijerarhiju. Slijedi provjera svih mogućih kombinacija elementa, na koji se želi postaviti povučeni predmet, uzevši u obzir njegovo potencijalno stanje.

Prvo se provjerava postoji li isti predmet u inventaru. Ako postoji, provjeraju se opcije:

- ima li pozicija na koju se preklapa predmet točku od vučenog elementa,
- je li riječ o poziciji s koje se krenulo,
- te predmeti nisu u upotrebi (engl. *equipped*).

Ukoliko je sve istina, a predmeti su isti, mogu se presložiti i spojiti (engl. *stack*). Druge se opcije (ukoliko postoje isti predmeti u inventaru) ne izvršavaju jer nisu kompatibilne.

Ukoliko ne postoji isti predmet u inventaru kao onaj koji se vuče, a mjesto ispod je prazno, predmet se premjesti. Ukoliko je predmet bio u upotrebi a potom premješten, pokrene se dio kôda (slika 13) koji zamjenjuje prošli broj odabranog mjesta s novim, te staro mjesto vraća u normalnu boju.

Na kraju se prethodni predmet briše, a inventar ažurira pozivom *`fill_inventory()`*.

```
if ImportData.inv_data[intu][2] == "Equipped":
	
	for skill in ImportData.skill_data:
		if ImportData.skill_data[skill][1] == intu:
			ImportData.skill_data[skill][1] = itembutton.get_parent().get_parent().name
			get_node(str(get_path_to(itembutton.get_parent().get_parent()))+ "/ItemBackground/ItemButton").self_modulate = Color(1,1,1)
			fill_inventory()
		else:
			pass

delete_item(intu)
fill_inventory()
```

Slika 13: Prikaz kôda za premještanje predmeta koji je u upotrebi

3.5.1.3. Skočni kotačić

Kada se odabere dugme, pritiskom lijeve tipke miša na određeni predmet u inventaru, instancira se i doda kotačić na desni kraj predmeta (***ItemPopupPanel.tscn***). Taj kotačić pomoćni je element sa skriptom, koja sadrži većinu funkcionalnosti koje se mogu koristiti pri manipuliranju predmeta u inventaru.

Funkcije unutar skripte ove scene su:

- `_ready()` – kotačič se prilikom stvaranja popunjava funkcionalnim elementima; ***Info, Delete, Drop, Equip/Unequip, Split***.
- `func _on_item_pressed(ID)` – pritiskom na pojedino stvoreno dugme, sukladno pripadajućem identifikacijskom broju, poziva se pripadajuća funkcija.
- `unequip_item()` – opcija se pojavljuje samo ukoliko je predmet u upotrebi. Funkcija uklanja odabrani predmet iz liste predmeta u uporabi (*singleton* `ImportData.inv_data[from_inv_slot][2] = „unequipped“`**)**, te ga miče iz vještina (engl. *skills*) na traci vještina (kroz *singleton* `ImportData.skill_data.erase(item)`).
- `equip_item()` – funkcija radi prohod kroz elemente trake vještina te ukoliko ima mjesta dodaje predmet u traku i ga zaplavljuje u inventaru.
- `split_item()` – ukoliko je broj predmeta u jednom mjestu inventara veći od 1, a u inventaru ima mjesta, ovom funkcijom (te pomoćnom fukcijom `split_second(number)`) ga dijelimo na pola (cjelobrojno).
- `info_panel()` – funkcija instancira te dodaje prozor predmeta s informacijama.
- `delete_item()` – ukoliko predmet nije u upotrebi, ova funkcija ga briše iz inventara.

- `drop_item()` – ukoliko predmet nije u upotrebi, ova funkcija dohvaća broj i tip predmeta koji se želi baciti u svijet. Potom ga instancira, pozicionira u svijetu na igračevu poziciju s:

  ```
  spawn_item.set_position(player.get_global_position())
  ```

  , dodaje u svijet (`world.add_child(spawn_item)`) i dodjeljuje mu količinu sa `spawn_item.quantity = quantity`. Zadnje se poziva `delete_item()` za brisanje originalnog predmeta u inventaru.

3.5.1.4. Prozor plijena

Prozor plijena je prozor koji se otvara prilikom interakcije s nekim od kovčega koji generiraju predmete unutar svijeta igre. Kada se klikne lijevom tipkom miša na određeni kovčeg, šalje se signal kroz ***CanvasLayer.gd*** o samom pritisku kao i potrebnoj blizini igrača. Ukoliko su oba uvjeta ispunjena, instancira se te se doda sama scena prozora plijena u sloj platna (grafičko korisničko sučelje).

Prilikom dodavanje scene poziva se nekoliko funkcija:

1. Generiranje nasumičnog broja uzimajući u obzir minimalni i maksimalni broj naveden u JSON datoteci koju je pročitao *singleton* ***ImportData***.
2. Odabir sadržaja prozora – generira se nasumični postotak, te se odabire predmet koji ima prvi veći postotak šanse pojavljivanja u kovčegu.
3. Grafičko i podatkovno punjenje prozora nakon prethodno obavljenih koraka tj. funkcija.

Dodatne funkcije su:

- Funkcija gašenja prozora koja signalom šalje zastavicu u sloj platna te od tamo gasi prozor.

- Funkcija slučaja pritiska `_on_LootButton_pressed(lootpanelslot)` koja uspoređuje kategoriju odabranog predmeta s kategorijama u inventaru. Ukoliko ima mjesta u ciljanoj kategoriji, sprema informaciju u riječnik `ImportData.inv_data`, koji se prilikom otvaranja inventara čita i popunjava sam inventar. Pokupljeni predmet se briše nakon kopiranja informacije u *singleton* riječnik.
- Funkcija za skupljanje svih predmeta koja ponavlja prošlu funkciju za svaki pojedini predmet u prozoru plijena.

3.5.1.5. Traka vještina

Većina klasičnih *RPG* igara ima traku vještina, s koje igrač bira različite napade, magiju ili mogućnosti interakcije s okolišom.

Prvo je potrebno napraviti vizualni dio same trake vještina, koji će biti prezentiran u obliku pravokutne plohe u donjem kraju samog vidnog polja igre. Traka će sadržavati više kvadratnih područja (mjesta) za mogućnosti koje igrač može odabrati i time iskoristiti odabranu moć ili napad u igri.

Prilikom učitavanja trake vještina u `_ready()`, poziva se spajanje signala svakog dugmeta vještina s funkcijom `SelectSkill()` . U njoj se definira ponašanje pritisnute ikone ukoliko ista postoji u spremniku *singletona* `ImportData.skill_data`. Prilikom odabira određene vještine preko ikone na traci vještina ikona se zaplavi:

```
get_node(skill_path + skill).self_modulate =
                Color(0,1,1)
```

Ponovnim klikom ili klikom na drugu ikonu, prethodno odabrana vještina se vraća u početno stanje uz pomoć metode `.self_modulate = Color(1,1,1)`.

Slijedi funkcija popunjavanja sadržaja trake vještina grafičkim elementima, kroz čitanje postojećih vještina iz `ImportData.skill_data`, te funkcija `disable_skill()` koja provjerava potrebnu razinu (engl. *level*) igrača kako bi se omogućilo korištenje pojedine

vještine. Ukoliko je vještina još nedostupna zbog preniske razine iskustva igrača, instancira se i postavlja `res://Scenes/SkillOverlay.tscn` element preko svake nedostupne vještine.

Posljednje slijede funkcije praćenja ulaska i izlaska miša. Prva prilikom ulaska miša u prostor pojedine vještine, postavlja sadržaj labele iznad trake vještina na ime vještine. Druga prati izlazak miša, te labelu briše ili mijenja novim sadržajem.

3.5.1.6. Traka iskustva

Svaki *RPG* ima neki oblik praćenja napretka lika. Najčešće se ta informacija prezentira kroz iskustvo koje se povećava rješavanjem zadataka ili onesposobljavanjem neprijatelja. Traka iskustva (engl. *experience bar*) izvedena je kao tip čvora ***TextureProgress*** sa zasebnom teksturom za sloj preko, ispod i iznad objekta.

Ono što je potrebno uočiti prilikom skaliranja vrijednosti koja se prikazuje je, da traka iskustva uvijek prikazuje raspon vrijednosti od 0 do 100 te je svaku vrijednost koja se želi preslikati na ovaj objekt potrebno skalirati u navedeni raspon.

Traka iskustva ovdje je samo grafički element. Funkcionalnosti su raspisane u skripti igrača. Detaljnije će biti objašnjenje u istoimenom naslovu „Igrač“.

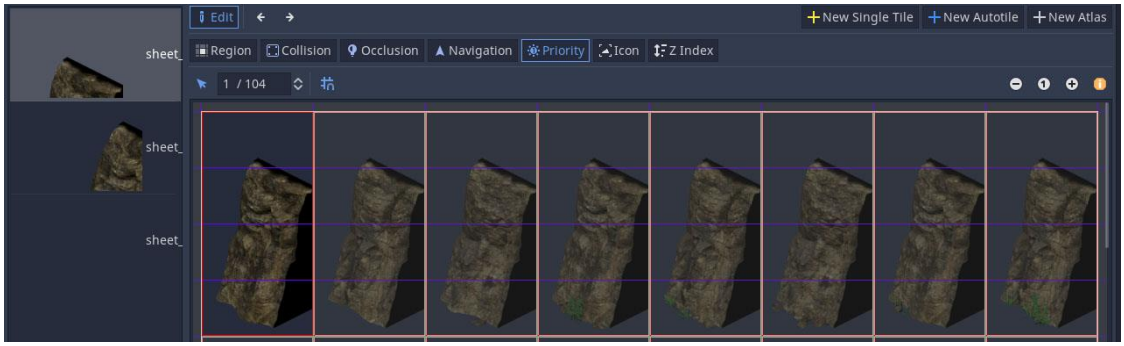
3.5.2. Svijet – čvor World1

Scena ***World1*** je tipa ***Node2D*** te se sastoji od podloge i okoliša koji su pretežno tipovi ***TileMap***, navigacijske plohe, te vrhovnog ***YSort*** čvora.

Ispod vrhovnog ***YSort*** čvora su u hijerarhiji postavljeni igrač, neprijatelji, ljudi u selu, te drugi različiti objekti i scene pod dodatnim različitim tematskim ***YSort*** čvorovima (primjerice ***Trees YSort*** čvor). Slijedi pojašnjenje većine bitnijih elemenata.

3.5.2.1. TileMap čvorovi

Godot ima specifični tip čvora ***TileMap***, koji se koristi za kreiranje ploha i elemenata svijeta u 2D dimenziji. Omogućuje stvaranje „pločica“ (engl. *tiles*) sadržaja koje se mogu dodavati slobodno ili koristeći ***pixel snap*** opciju kako bi se pločice savršeno poredale u definiranu mrežu. ***TileMap*** se sastoji od ***TileSet*** resursa u kojeg se dodaju 2D sličice ili nizovi sličica ***(Sprite Sheets***). Ovdje se mogu izrezati pojedini elementi koji će činiti zasebne pločice.



Slika 14: Prikaz niza sličica u ***TileSet*** opciji ***TileMap*** vrste čvora

Postoje opcije izrade nove pojedinačne pločice, niza automatskog odabira pločica (***Autotile***) te skupine grupiranih pločica (***Atlas***). Nakon odabira opcije definiraju se dodatne stavke:

- ***Region*** – bira se dio niza slika ili pojedinačne slike koje se režu za pločice.
- ***Collision*** – označavanje dijela pločice s kojom će sudarači reagirati sudarom.
- ***Occlusion*** – ukoliko se želi postići da dio pločice ograničava tj. priječi put svjetlu (rezultat je sjena).
- ***Navigation*** – može se definirati prostor pločice koji se koristi kao navigacijska ploha za proračune i izvršavanje kretanja pojedinih likova i pomičnih elemenata u igri.
- ***Priority*** – odabir učestalosti pojedine pločice ukoliko se odredi više pločica u nizu, a koristi ***Autotile*** opcija postavljanja pločica.

- ***Icon*** – odabir ikone za ***Autotile*** (nasumični izbor pločica) funkciju ili ***Atlas*** (grupirane pločice).
- ***Z index*** – postavljanje Z indeksa svakom pojedinačnom elementu.

Zatim se stvoreni ***TileSet*** spremi kao resurs .tres formata, te odaberu sljedeće stavke u inspektoru:

- ***Mode*** – ***Isometric*** – budući da se radi o izometrijskoj igri.
- ***Cell*** – ***Size*** – definiraju se x i y s obzirom na veličinu pločica. Ova veličina obično je već prilagođena i često se nalazi u nazivu resursa, ukoliko se koristi neki resurs s interneta ili se može proizvoljno definirati. Pretežno je korištena dimenzija x = 128, y = 64.

Moguće je podesiti i druge stavke pozicioniranja same pločice prilikom postavljanja na mrežu u svijetu poput ***Tile Origin***, ***Custom Transform*** itd. Nužno je odabrati ***Y Sort*** opciju, kako bi se elementi prilikom slaganja u svijetu pojavljivali jedni iza drugih prateći redoslijed osi y.

Sada se s desne strane ***Viewport***-a vidi prozor s dodanim elementima u ***Tileset***-u, koje je moguće dodati u svijet.

TileMap scene i čvorovi igre su pretežno pozicionirani ispod ***World1*** scene budući da su uglavnom elementi plohe po kojoj igrač hoda, te im je y sortiranje bitno samo međusobno, ne i s igračem.

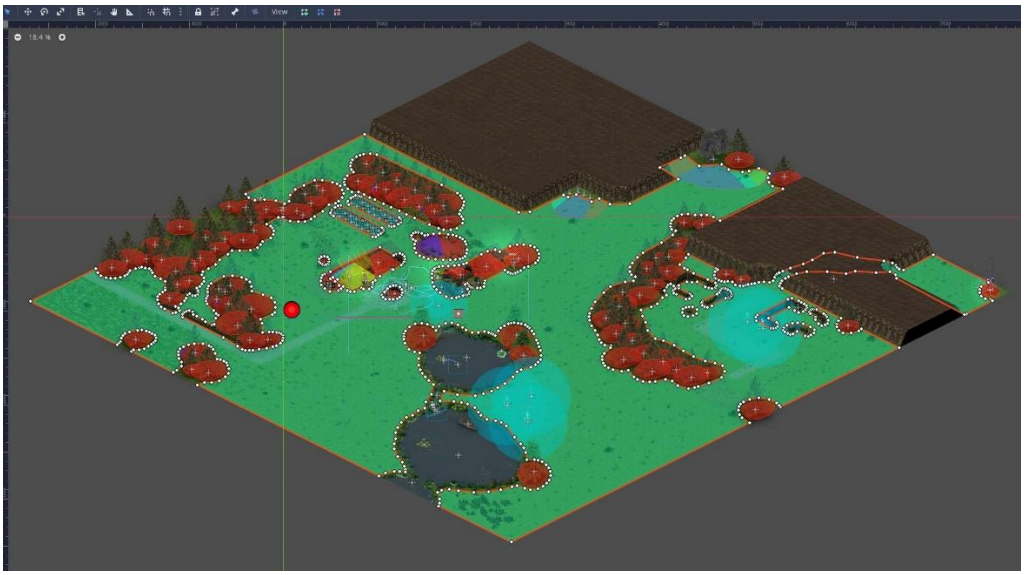
Ukoliko je potrebno kreirati ***TileMap*** scenu ili čvor i koristiti je u y sortiranju skupa s primjerice igračem, potrebno je napraviti određene korake (hijerarhijsko pozicioniranje) o čemu će biti više riječi u „Čvor YSort” naslovu rada.

3.5.2.2. Navigacijski čvor

Budući da se lik igrača pokreće klikom po poziciji na mapi, a neprijatelji se kreću prateći poziciju igrača ili vraćaju na početnu poziciju, potrebno je riješiti problem navigacije i puteva. Koristi se čvor tipa ***Navigation2D***, kojem se kao dijete doda ***NavigationPolygonInstance*** čvor (Za 3D se koriste ***Navigation*** te ***NavigationMeshInstance*** varijacije). Potom se mišom definira prostor navigacijske plohe.

Potrebno je uzeti u obzir maksimalnu širinu sudaračke plohe likova, koji koriste navigacijsku plohu, u odnosu na statične objekte s vlastitim sudaračkim plohama u svijetu.

Razlog tome je slučaj gdje kinematičko tijelo zapinje svojom sudaračkom plohom o statičko tijelo (sudaračku plohu), prilikom kretanja uz rubove navigacijske plohe. Stoga treba dimenzionirati rubove navigacijske plohe, tako da nikada ne dođe do sudara navedenih objekata. Ovo je jedan od problema Godota od samog početka.

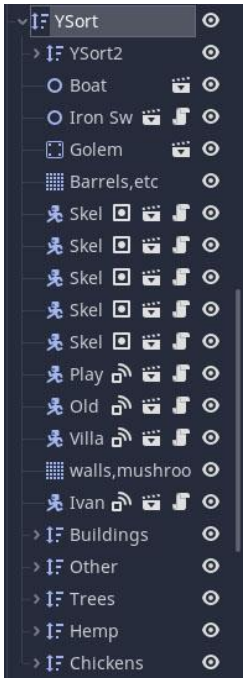


Slika 15: Prikaz definirane navigacijske plohe (zelena površina)

Definirana ploha referencira se u ***Player.gd*** skripti, te u skriptama svih neprijatelja. Detalji korištenja plohe u svrhu kretanja kinematičkih tijela biti će detaljnije objašnjena unutar sekcija igrača i neprijatelja.

3.5.2.3. Čvor YSort

Čvor ***YSort*** je vrsta čvora koja služi za sortiranje svih podređenih čvorova po osi y. To znači da će svaki element dodan kao dijete u pojedini čvor ***Ysort***, biti prikazan ispred ili iza drugog predmeta u istoj hijerarhiji po visini y pozicioniranja na prostoru ***Viewport***-a. Grupiranje tj. nizanje ***YSort*** čvorova se izvodi tako da se kreira roditeljski čvor ***YSort***, te se njemu kao djeca dodaju uzastopni ***YSort*** čvorovi koji su poredani po nekoj logici. Primjer je slika 16.



Slika 16: Prikaz hijerarhije YSort čvorova igre

Treba pripaziti da se prilikom stvaranja ***YSort*** tipa čvora odabere ***Sort*** -> ***Enabled*** opcija u inspektoru, kako bi se sortiranje stvarno i izvršilo.

3.5.2.4. Igrač

Za izradu lika igrača koristi se fizikalno tijelo vrste ***KinematicBody2D***, koje detektira sudar i pokreće se uz pomoć kôda (primjerice unosom tipkovnice ili miša). Glavna značajka ovog tijela je da na njega ne utječe ostala fizika, primjerice gravitacija, udarac drugog objekta i slično. Ovakva tijela pogodna su za implementaciju kinematičkih likova (igrača, neprijatelja i slično) koristeći dostupne metode za kretanje i provjeru sudara – `move_and_slide` te `move_and_collide`, kao i za pomicanje raznih platformi te objekata od strane samog igrača.

Budući da ***Player.gd*** skripta ima gotovo 30 funkcija, ukratko će se pojasniti samo nekoliko bitnijih:

- `add_experience()` – poziva se prilikom rješavanja zadataka i ubijanja neprijatelja. Dodaje se iskustvo na već postojeće iskustvo, te se uspoređuje s listom definiranih razina i potrebnog iskustva za prelazak određene razine igrača. Ukoliko je pređen prag potreban za prelazak razine igrača, poziva se funkcija `level_up()` koja mijenja trenutnu razinu igrača.
- `show_exp()` – funkcija ima nekoliko uvjeta provjere trenutne razine igrača, moguće sljedeće razine, te shodno tome popunjava grafičke elemente scene trake iskustva.
- `_process()` – unutar se pozivaju funkcije `SkillLoop()` te `check_range()`.
- `SkillLoop()` – ukoliko je pritisnuta tipka miša za korištenje vještine te je igrač u mogućnosti koristiti istu, radi se `match selected_skill` provjera odabrane vještine. Ukoliko je odabrana vještina magija s dometom, instancira se scena magije, definira njena rotacija i početna pozicija u odnosu na klik mišem u svijetu, te se scena doda u svijet kao dijete igrača. Izvršavaju se i pojedini brojači ovisno o odabranoj vještini. Ukolik odabrana vještina nije magija, radi se `match` s oružjima te se postavljaju prigodne zastavice (`range_chosen = false`, `melee_chosen = true`).

- *_physics_process(delta)* – unutar funkcije procesiranja fizike pozivaju se funkcije kretanja *Movement(delta)* i animacije *Animation_func()*.
- *Movement(delta)* – povećava se brzina lika kroz *speed+velocity*delta* do maksimalne brzine (200). Dohvaća se početna pozicija igrača te odredišna lokacija klika miša, negdje u svijetu unutar navigacijske plohe. Odredište se hvata funkcijom *_unhandled_input(event)* koja dohvaća najbližu točku navigacijske plohe u odnosu na klik miša:

```
destination=
nav_reference.get_closest_point(get_global_mouse_position()
```

Definira se kvant (fragment) kretanja, *movement* koji je maksimalno jednak:

```
move_speed (200) * delta
```

Potom se iterira kroz veličinu puta (*dest_path.size()*) te postavlja varijabla sljedeće točke na globalnu udaljenost do nje:

```
var next_point_dist = start.distance_to(dest_path[0])
```

Ukoliko je kvant pomaka *movement* manji ili jednak udaljenosti koju je potrebno preći do sljedeće točke, dohvaća se rotacija potrebna za vektor kretanja:

```
move_rotation =
get_angle_to(start.linear_interpolate(dest_path[0],
movement / next_point_dist))
```

Za potrebe okretanja animacije igrača:

```
movement_angle = rad2deg (move_rotation)
```

pretvaraju se radijani u stupnjeve, budući da se u *Animation_func* provjeravaju stupnjevi za potrebe pokretanja prikladne igračeve animacije.

Nakon toga se kreira *Vector2()* koji će imati definiranu prvu komponentu, a metodom rotacije postiže se dvodimenzionalno kretanje (druga komponenta).

Zatim:

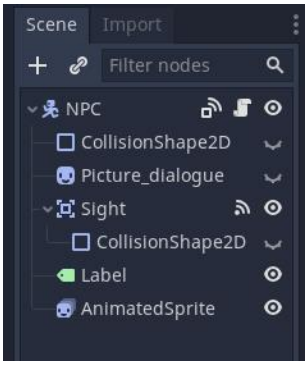
```
Vector2(speed,0).rotated(move_rotation)
```

kreira vektor nagnut u pravilnom smjeru sukladno odredišnoj točki. S *move_and_slide(motion)* izvršava se kretanje kinematičkog tijela po navedenom vektoru te se izlazi iz *for* petlje.

Ukoliko kvant pomaka nije manji od sljedeće odredišne točke, od kvanta pomaka oduzima se vrijednost sljedeće točke, varijabla *start* postaje prva točka destinacije *dest_path[0]*. *dest_path* postaje prva sljedeća točka budući da se s *dest_path.remove(0)* uklanja prethodno prvi element. Naravno, ukoliko je veličina *dest_path = 0* zaustavlja se kretanje postavljanjem *is_moving = false* zastavice.

3.5.2.5. Računalno upravljani likovi

Objekt ***NPC.tscn*** je scena tipa ***KinematicBody2D*** sa sljedećom strukturom (slika 17):



Slika 17: Prikaz strukture ***NPC*** scene

Vršni čvor tipa ***CollisionShape2D*** je doticajna sudaračka ploha s podlogom i drugim sudaračkim objektima. ***Picture_dialogue*** je slika koje se pojavljuje u prozoru dijaloga prilikom razgovora s pojedinim računalno upravljanim likom (dalje skraćeno – lik).

Sight je ***Area2D*** tip čvora, koji preko sudaračke plohe tipa ***CollisionShape2D*** registrira ulazak drugog sudaračkog objekta u definirani prostor. Ukoliko je taj objekt igrač, otvara se prozor dijaloga (ukoliko igrač odabere lika klikom miša). ***Label*** je naziv lika koji se pojavljuje prilikom prelaska miša preko njega. Posljednji element je čvor tipa ***AnimatedSprite*** u kojem se postavljaju animacije lika.

Računalno upravljani lik je zadužen za praćenje blizine igrača, potencijalnog klika od strane igrača na lika, te slanje signala za stopiranje kretanja igrača ukoliko se dijalog prozor otvori.

Iz početne scene računalno upravljanog lika nasljeđivanjem su izvučeni dodatni likovi, a mijenjane stvari u naslijeđenim likovima su: resurs slike svakog pojedinog lika, prilagođene animacije, te `input_event` funkcija ukoliko ima nekih promjena u odnosu na original.

Pojedini naslijeđeni likovi imaju dodatnu funkciju:

```
check_for_items(_texture_ignore)
```

koja se spaja iz signala otvaranja dijaloga, a služi za poziv funkcije provjere posjeduje li igrač specifični predmet koji je potreban za određeni zadatak.

3.5.2.6. Neprijatelji

Svaka *RPG* igra ima neku vrstu neprijatelja, koju protagonist priče mora poraziti ili nadmudriti. U ovoj igri postoji nekoliko tipova neprijatelja:

- ***Skeleton*** – glavna vrsta neprijatelja iz koje se nasljeđuju ostale vrste neprijatelja. Struktura scene kostura je slična računalno upravljanim likovima, s dodatkom ***TextureProgress*** čvora, koji služi kao prikaz „zdravlja“ kostura, te ***Area2D*** sa sudaračkim ***CollisionShape2D*** čvorom za detekciju blizine igrača (koristi se za napad kostura).

  Od bitnijih funkcija kostura izdvojene su:

- `OnHit(damage)` – funkcija za oduzimanje trenutnog zdravlja kostura od strane napada igrača. Kada zdravlje kostura dođe do 0 (ili ispod), kostur se briše iz aktivne scene sa `self.queue_free()`.

  Također se pridjeljuje iskustvo igraču, i šalje informacija u `QuestHandler` da javi pojedinom zadatku. Traka zdravlja se ažurira s npr.:
  `health_bar.set_tint_progress("89ff07")`,
  gdje 89ff07 predstavlja zelenu boju.

- `_physics_process(delta)` – pozivaju se `Animation_func` koja ovisno o kutu kretanja kostura i stanju (*idle*, *melee*, *walk*) poziva prikladnu animaciju, te sljedeći niz provjera. Prilikom učitavanja kostura početni state je `Idle`. Radi se `match` stanja koji za `Idle` postavlja zastavice `moving = false`, `melee = false`. Nakon toga se vrti ***Idle*** animacija kostura i on se ne kreće niti ne napada. Slijedi provjera je li igrač u vidnom polju, kako bi se postavilo određeno stanje kostura. Uz pomoć:

  ```
  space_state = get_world_2d().direct_space_state
  ```

  dohvaća se poslužitelj fizikalnog svijeta sa svim sudarima i informacijama fizike u 2D svijetu.

  Potom se radi ***raycasting***:

  ```
  sight_check = space_state.intersect_ray(position,
  player.global_transform.origin, [self], collision_mask)
  ```

  , koji kreće od pozicije kostura prema poziciji igrača, s time da se isključi kolizija sa samim sobom te kao četvrti argument isključi `collision_mask` cijele grupe kostura. Ukoliko se ***raycast*** zraka sudarila s objektom, čije je ime ***„Player“***, dohvaća se pozicija igrača:

  ```
  dest = nav_reference.get_closest_point(
      player.get_global_position())
  ```

, te se stanje postavlja u ***„Follow“***.

- `_on_Area2D_body_entered(body)` – funkcija provjerava ima li tijelo koje uđe u sudar s sudaračkom površinom metodu `remove_health`. Ukoliko ima, kostur troši zdravlje igrača s brojačem za svaki napad.

- ***Ghost*** – ljubičasti duh koji se pojavljuje kraj kamenog diva. Potrebno ga je odvući do groblja gdje se jedino može poraziti. Ukoliko izgubi igrača iz vida, nestaje i vraća se na početno počivalište kraj kamenog diva. Skripta i scena duha nasljeđuju ***Skeleton*** skriptu i scenu, a dodane su funkcije provjere nalazi li se duh na lokaciji groblja, te nestajanja pri vraćanju na početnu poziciju. To se provjerava preko roditeljske scene tipa ***TileMap*** koja čini tlo u igri te ima zakačene ***Area2D*** plohe groblja i mjesta pojavljivanja duha.
- ***Ghost_white*** – nasljeđuje ***Ghost.gd*** i ***Ghost.tscn***. Ovaj duh (ili duhovi) pojavljuju se noću na groblju neovisno o drugim parametrima, kao i umjesto postojećih kostura.

3.5.2.7. Statični objekti

Ukoliko je potrebno stvoriti objekt koji će imati sudaračku površinu, no neće se pomicati niti tražiti zahtjev za provjeru kolizije od strane fizičkog dijela pokretača igre, koristi se vrsta čvora ***StaticBody2D***. Primjer takvih objekata u ovoj igri su: kuće, ulaz u špilju, sama špilja, većina stabala, jezera itd. Statično tijelo se obično slaže u konstrukciju koja sadrži **Sprite** čvor (sličicu), te sudarački objekt (plohu) kao što su ***CollisionShape2D***, ***CollisionPolygon2D*** itd.

Po potrebi se oblikuje sama sudaračka ploha, sukladno dimenzijama u odnosu na ***Sprite*** čvor. Tako je, na primjeru stabla, sličica ***Sprite*** čvora cijelo stablo, a sudaračka ploha se obično iscrtava kao elipsa oko korijena stabla i oboda krošnje stabla.

Gornji dio stabla u ovom slučaju nije u „dodiru“ s izometrijskom plohom svijeta, tako da sudaračka ploha čini samo donju polovicu objekta stabla.

Ostatak realističnog prikaza postiže se korištenjem ugniježđivanja u ***YSort*** čvorovima, te pozicioniranjem sličice i sudaračkog tijela u ***Viewport*** prozoru.

Budući da je kretanje lika iz početnog kretanja sa strelicama na tipkovnici (kao i ***WASD***) opciji, prebačeno na kretanje dvostrukim pritiskom lijeve tipke miša, većina sudaračkih ploha izgubila je dio funkcionalnosti.

Njihova funkcionalnost, umjesto sudara s tijelima igrača i neprijateljima, postaje sudaranje s instanciranom magijom kako ista ne bi prolazila kroz objekte. Postoje neki izuzetci s posebnim mogućnostima.

Prvi primjer je ***SmallLake1.tscn*** u ***World1*** sceni. Scena ima dodani ***Area2D*** čvor sa sudaračkom površinom koja detektira ulazak i izlazak igrača u blizinu jezera. Ukoliko neko tijelo (engl. *body*) uđe u prostor, provjerava se identitet tijela. Ako je tijelo igrač, dodaje mu se 20 bodova zdravlja s pozivom `player.add_health(20)`.

Drugi primjer je scena ***Cave.tscn*** na lokaciji ***World1 -> Ysort -> Buildings***. Ovu scenu čine dva ***Sprite*** čvora. Prvi je sama unutrašnjost špilje. Drugi je sloj koji obuhvaća zidove ulaza i donje plohe koja vizualno prekriva i preklapa dio unutrašnjosti špilje. Dalje sadrži dva sudaračka sloja, koja određuju gornju i donju granicu sudara objekata u prostoru špilje (npr. neprijatelji ili magija).

Posljednje sadrži ***Area2D*** plohu koja se koristi kao detekcija ulaska i izlaska igrača i neprijatelja te shodno tome radi manipulaciju Z indeksa pojedinih navedenih slojeva. Detaljnija funkcionalnost objasniti će se u naslovu „Manipulacija Z indeksom“.

3.5.2.8. Putovi

Kako bi kinematička tijela ili drugi objekti pratili neki predodređeni put, koriste se ***Path(2D)*** te ***PathFollow(2D)*** tipovi čvorova. ***Path2D*** čvor služi za crtanje putanje. Kreira se nova krivulja puta korištenjem miša i alatne trake iznad ***Viewport***-a. Slijedi stvaranje ***PathFollow2D*** čvora kao djeteta ***Path2D*** čvora.

Budući da je ideja dodati neki objekt (u ovom slučaju kokoši) koji će pratiti definirani put, no pri tome je dodatni uvjet ispravno ***YSort*** sortiranje s drugim objektima u igri, nije moguće direktno postaviti objekt kao dijete ***PathFollow2D*** čvora. Razlog tome je funkcionalnost ***YSort*** čvora koji radi sortiranje samo na direktnoj djeci. Slijedi rješenje problema.

Scena ***ChickenKB1*** se postavlja u hijerarhiju ***World1 -> Ysort -> Chickens*** (***YSort*** tip). Način na koji se povezuje scena kokoši i put koji je prije stvoren, a koji se nalazi izvan ***YSort*** hijerarhije (u ***World1 -> Chicken&FrogPath***) je korištenje ***RemoteTransform2D*** tipa čvora. Ovaj tip čvora može prenijeti poziciju, kretanje te dimenziju (engl. *scale*) jednog čvora na drugi.

Doda se čvor tipa ***RemoteTransform2D*** kao dijete prije spomenutog ***PathFollow2D*** čvora, te mu se kao ***RemotePath*** dodijeli scena kokoši koja se nalazi u prije navedenoj lokaciji. Koriste se globalne koordinate, te definiraju tipovi prijenosa (po volji). Od ostalih opcija potrebno je prenijeti samo poziciju.

Sada je uspješno spojena kokoš i put koji treba pratiti, a kokoš je postavljena u odgovarajući ***YSort*** čvor. Tako će biti ispravno prikazana u prostoru iscrtavanja scene po osi y u odnosu na ostale objekte u prostoru. Kôd za kretanje biti će pojašnjen u naslovu „Dodatni elementi i tehnike“ u dijelu opisa elementa žabe, jer dijele slične funkcionalnosti.

3.5.2.9. Dijalog skripta i prateće JSON datoteke

Dijalog prozor je strukturno posložen veoma slično već navedenim elementima poput inventara i prozora vještina. Dodatno se koriste elementi tipa ***RichTextLabel***.

Potreba korištenja ovog tipa čvora, naspram klasičnih čvorova ***Label*** korištenih do sad, je funkcionalnost `meta_clicked` signala.

On se koristi umjesto pojedinačnih dugmadi za pozivanje odabrane opcije odgovora ili pitanja u dijalogu klikom na dio teksta. Pojašnjenje slijedi u opisu bitnijih funkcija.

- `_ready()` – prilikom svakog pozivanja dijalog prozora spaja se signal `stop_player_movement` s funkcijom `start_player` u skripti igrača, kako bi se isti zaustavio dok je otvoren dijalog.

- `set_sprite(npc_name)` – dohvaćanje imena računalno upravljanog lika slijedom:

  `emit_signal("OPEN_DIALOGUE", sprite_for_dialogue )` iz lika se veže na ***CanvasLayer.gd*** funkciju `open_dialogue(npc_name)` koja poziva `dialogue.set_sprite(npc_name)` funkciju.

  Potom dohvaća sliku lika koja se prikazuje u dijalog prozoru:

  `get_node("TextureRect/TextureRect/VBoxContainer/HBoxContainer/TextureRect").set_texture(load("res://"+ npc_name +".png"))`,

  te poziva funkciju `fill_dialogue()`.

- `fill_dialogue()` – dohvaća sve likove iz JSON datoteke koja se pohranjuje u *singleton* `ImportData.dialogue_data`. Ukoliko je odabrani lik isti kao i onaj pronađen u `dialogue_data` provjeravaju se mogući uvjeti. To su;

  1. ***final*** čvor koji označava da je razgovor s likom došao do zadnje točke. Provjeravaju se elementi u *singletonu* `QuestHandler.quest_giver` u kojem se pohranjuju aktivni zadaci igrača. Ukoliko postoji zapis ***final*** znači da je linija zadataka tog računalno upravljanog lika došla do kraja. Stoga se poziva čvor ***final*** iz ***JSON***-a za lika s kojim se trenutno razgovara.
  2. Provjeravaju se svi otvoreni zadaci računalno upravljanih likova s kojima igrač razgovara te njihova stanja.

     Ukoliko je pojedini zadatak u stanju ***2*** koji označava da su uvjeti završetka zadatka ispunjeni, poziva se metoda specifičnog zadatka za nagradu:

     `get_node("/root/"+quest_name).completedQuest()`,

     te se zadatak briše iz liste aktivnih zadataka:

     `QuestHandler.quest_giver[i].erase(quest_name)`

i poziva sljedeći čvor teksta za završetak zadatka:

```
next_dialogue("ended"+quest_name, npc_selected)
```

3. Treća opcija je generička opcija. Ukoliko nije ***final*** i nije završen zadatak, poziva se čvor ***initial*** za navedenog računalno upravljanog lika, te se u listu spremaju moguće opcije sljedećih čvorova koje se dalje pozivaju klikom na pojedinu opciju.

- `next_dialogue(knot_name, which_npc_selected)` – funkcija nastavka dijaloga. Poziva se ili preko ***initial*** čvora iz prijašnje funkcije ili preko `_on_RichTextLabel_meta_clicked(meta)` funkcije, u kojoj se preko `match meta` omogućuje da klik na pojedini dio meta teksta ***JSON***-a poziva pripadajući element iz liste odabira čvorova.

Provjerava se je li odabrani čvor ***END*** čvor, ***final*** čvor ili ***empty*** čvor te se sukladno rezultatu prikazuje prikladan sadržaj prozora. Ukoliko nije nijedan od navedenih, prvo se gleda mogućnost aktivacije zadatka ukoliko je čvor novi zadatak.

Nakon toga se provjerava koji su mogući izbori. Ukoliko je zadatak već riješen, taj izbor se preskače i ne prikazuje u čvoru ***RichLabelText***.

Ukoliko je zadatak već otvoren, dodaje se u listu kako bi se zadržala dimenzija forme odabira, ali se ne prikazuje opcija odabira već otvorenog zadatka. Dalje se provjerava zadovoljava li zadatak zahtjev (uvjet) rješenog prijašnjeg zadatka. Ako ima, provjerava se postoji li riješeni zadatak u prijašnje riješenim zadacima.

Naposlijetku, sve ostale moguće kombinacije normalno se dodaju u listu i prikazuju.

3.5.2.10. Zvuk

Elementi zvuka dostupni u Godotu su ***AudioStreamPlayer***, ***AudioStreamPlayer2D*** te ***AudioStreamPlayer3D***. Korišteni elementi zvuka u radu su tipovi ***AudioStreamPlayer*** za zvuk bez prostorne potrebe, te ***AudioStreamPlayer2D*** za zvuk koji se prostorno postavlja. Ovisno o potrebi i namjeni zvuka, koriste se različiti formati zvukovnog zapisa. Primjerice ***.ogg*** format za dugotrajne zapise poput zvukova okoline ili zvučne pozadine, te ***.wav*** za zvukove koji su jednokratni. (npr. neki udarac ili glas). Moguće je postaviti udaljenost s koje će se određeni zvuk čuti ukoliko se koriste 2D ili 3D tipovi zvuka s prostornom komponentom.

Primjer korištenih zvukova su:

- ***AudioStreamPlayer*** sa snimkom pozadinske glazbe naziva ***„par_akorda_duzi“.ogg***.
- ***AudioStreamPlayer2D*** sa snimkom glasanja kokoši u ***.wav*** formatu koji je postavljen prostorno u selu. Ima dodatnu varijablu ***„Max Distance“*** postavljenu na 600, kako bi se zvuk čuo samo u relativnoj blizini kinematičkih objekata kokoši.

  Sličan je i zvuk žabe na lopoču, te osobe u selu.

3.5.2.11. Manipulacija Z indeksom

Z indeks je parametar dostupan u većini tipova čvorova i zaslužan je za kontrolu redoslijeda iscrtavanja objekata. Objekti s većim Z indeksom iscrtavaju se ispred onih s manjim.

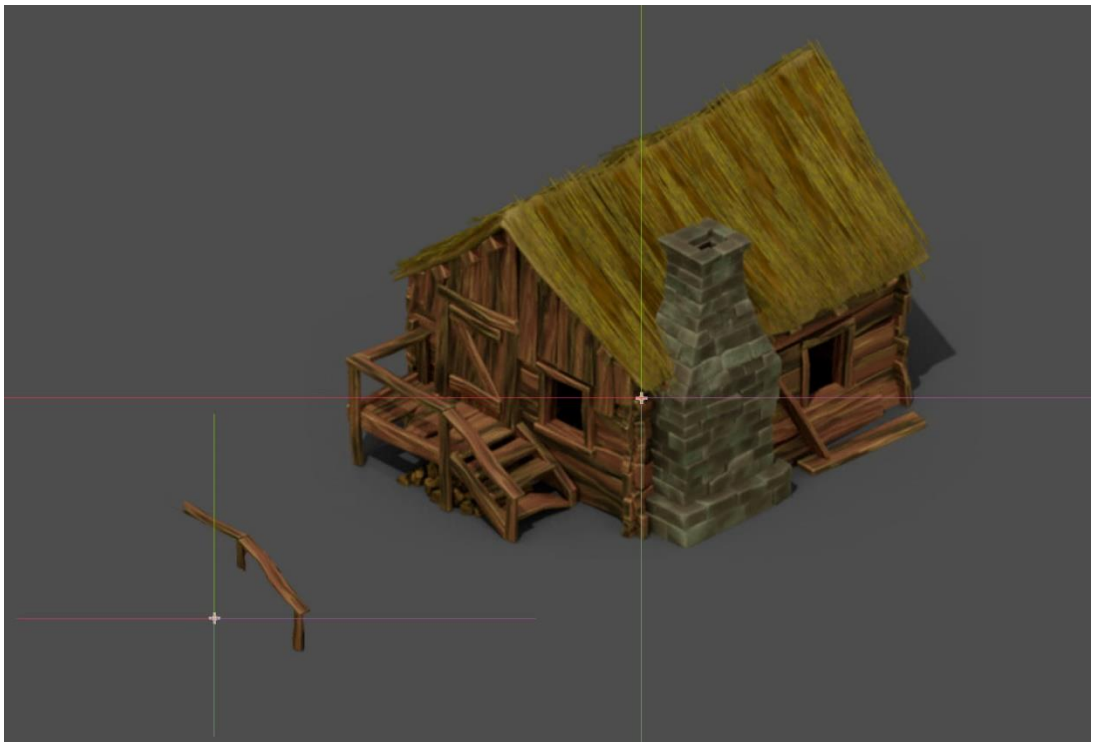
Prvo slijedi primjer višeslojnog slaganja slojeva, korištenjem pozicije objekta na y osi unutar istog Z indeksa (nativno je 0). Ideja je da se igrač uspne stepenicama kuće, a da mu pritom rukohvat stepenica bude prikazan ispred igrača (realistično). Budući da je riječ o 2D igri (izometriji), potrebno je pronaći rješenje koje će pružiti ovu simulaciju realističnosti.

Način na koji se ovakav problem rješava je izrada dvaju zasebnih elemenata (scena).

Prvi je sam statični objekt kuće, a drugi rukohvat (izrezan iz samog originala kuće, bez ostatka kuće). Potrebno ih je pojedinačno pozicionirati u odnosu na os y u lokalnom ***Viewport***-u (unutar svoje scene). Lokalno se definira točka u kojoj objekt u interakciji odlazi iza ili ispred elementa. Kada tijelo igrača prođe ovu liniju, biti će prikazano iza ili ispred objekta (ovisno o vrijednosti osi y). Potom se rukohvat i kuća preklapaju u sceni svijeta.

Kada se igrač popenje stepenicama rukohvat će se prikazati iznad, a kuća i stepenice ispod igrača te će se dobiti dojam 3D dimenzije i ispravnog iscrtavanja slojeva. Razlog tome je što se igračeva pozicija u lokalnom ***Viewport***-u (bitna za redoslijed iscrtavanja) uspoređuje s lokalnim pozicijama navedenih elemenata.

Na isti način se može i drugim objektima dati efekt dubine. Isječci pozicioniranja dvaju objekata u odnosu na lokalne x i y osi su vidljivi na slici 18.



Slika 18: Prikaz pozicioniranja elementa rukohvata i kuće. Elementi su napravljeni zasebno, spojena slika prikazuje pojedinačno pozicioniranje u odnosu na osi x i y (crvena i zelena crta)

Primjer manipulacije Z indeksa kôdom, vidljiv je u skripti scene ***Cave.tscn***. Bitno spomenuti da se cijela scena špilje u svijetu prvotno postavlja na `Z index = -3`. Tako je početno prekrivena (ispod ***TileMap*** pločica vanjskog svoda špilje) dok igrač ne uđe u nju. Unutar same scene, gornja ***Sprite*** ploha je postavljena na `Z index = 0`, a donja na `Z index = 2`. U skripti špilje bitnije funkcije su:

- `_on_Area2D_body_entered(body)` – u funkciji se provjerava ulazak tijela u definirani ***Area2D*** prostor. Ukoliko je tijelo koje je ušlo u prostor igrač, igrač se zasjeni s `.modulate = Color8(85,85,85)`. Color8 se koristi za 8-bitno definiranje boje (0-255). Potom se sa `.set_z_index(2)` postavlja indeks igrača na 2, indeks špilje na 1, te se postavlja zastavica `player_inside = true`. Potom se dohvaćaju sva tijela u prostoru:

  ```
  targets = $Area2D.get_overlapping_bodies()
  ```

  kako bi se moglo manipulirati neprijateljima koji potencijalno slijede igrača u špilju. Slijedi iteriranje po svim uhvaćenim objektima u prostoru. Ako je tijelo igrač, propušta se slučaj s `pass`, ako je neko drugo tijelo postavlja se indeks toga tijela na 2. Ideja iza ovoga je da se u trenutku kada igrač uđe, osim same manipulacije Z indeksom igrača, postave i svi već postojeći neprijatelji u špilji na prikladne Z indekse. Slijedi dio s `else`, gdje se provjerava je li igrač već unutra. Ako jest, tijelo koje ulazi također mora proći proces postavljanja sjene te Z indeksa na 2.

- `_on_Area2D_body_exited(body)` – slično kao i prijašnja funkcija, s time da ovdje namješta igrača na izlazu na `Z index = 0`, te ga modulira u početnu boju s `Color(1,1,1)`. Samu špilju vraća na `Z indeks = -3`, te zastavicu `player_inside = false`, a tijela koja nisu igrač vraća na `Z index = 0`. Ukoliko tijelo koje izlazi nije igrač nego neprijatelj, također mu postavlja `Z index = 0`, te se modulira boja u početnu s `Color(1,1,1)`.

- `_on_Area2D_area_entered(_area)` – funkcija prati ulazak ***Area2D*** objekata u prostor. U ovom slučaju služi za praćenje ulaska magije (objekata koje igrač ispaljuje) u prostor, te ih postavlja iznad svega u prostoru sa `Z index = 4`.

3.5.2.12. Nasljeđivanje

Nasljeđivanje – Primjer nasljeđivanja je klasa Example_Item.tscn (scene i skripte su praktično klase). Sama klasa predstavlja bazični, generički predmet s pripadajućim funkcijama koje su potrebne kako bi se predmet uklonio iz inventara te uzeo natrag.

Wood Sword, Berries i ostali predmeti nasljeđuju baznu skriptu i scenu. Mijenja se naslov predmeta i ikona kako bi dobili nove varijacije, a zadržali funkcionalnosti naslijeđene scene i skripte.

Imenovanjem iste funkcije u naslijeđenoj klasi premošćuje se roditeljska klasa, a ukoliko je potrebno pozvati roditeljsku klasu iz djeteta poziva se na način:

```
.imeRoditeljskeKlase
```

Primjer toga je scena ***Ghost_white*** s istoimenom klasom, gdje se u funkciji `OnHit(damage)` poziva roditeljska klasa nakon premošćivanja s uvjetom ako je duh na groblju, te se nakon provjere uvjeta poziva roditeljska funkcija s `.OnHit(damage)`.

3.5.2.13. Dodatni elementi i tehnike

Izrada projektila (magija) – Nakon kreiranja ***RigidBody2D*** objekta odabere se detekcija kontakta s `contact monitor = on`, te broj kontakata na koja će reagirati na 1 s `contacts_reported = 1`.

Unutar `_ready()` funkcije primjenjuje se impuls te rotacija s metodom:

```
apply_impulse(Vector2(), Vector2(speed, 0).rotated(rotation)
```

Prva varijabla je pomak (engl. *offset*), a druga sam impuls (element x je brzina gibanja objekta). Element y je nula te se metodom `.rotated` postiže rotacija tijela u odnosu na globalne koordinate odredišta, dok je početna točka u središtu samog tijela.

Žaba (***Frog.tscn***) – kinematičko tijelo sa sljedećim elementima: ***AnimatedSprite, CollisionShape2D*** te ***AudioStreamPlayer2D***. U ***Frog.gd*** skripti, pored brojača za pojavljivanje, skok i nestajanje, bitna funkcija je `Movement(delta)` s kojom se objekt kreće po putanji ***Path2D*** čvora na koji se veže. Ukoliko je žaba u fazi skoka, koristi se:

```
follow_path.set_offset(follow_path.get_offset() +
                       speed * delta)
```

koja dohvaća trenutni pomak žabe, te ga uvećava kroz umnožak definiranog parametra brzine i fiksnih vremenskih intervala `delta`. Kada vrijednost prijeđe duljinu puta (180 piksela), s:

```
get_parent().get_parent().get_node("Particles2D8_frog_sprinkle").set_
emitting(true)
```

dohvaća se, te emitira, čestični objekt prskanja vode, te se žaba skriva sa `self.hide()`. Kokoši su izvedene na sličan način, samo što se prvo dohvaća početna globalna pozicija (prethodna pozicija) puta na kojeg je smještena kokoš, miče sa `.set_offset` metodom, ali i računa kut kretanja kokoši potreban za prikazivanje pravilne animacije kokoši.

3.6. Kreiranje sadržaja u Blenderu

3.6.1. 3D modeliranje

Za potrebe ovog rada napravljeno je više 3D objekata, animacija, tekstura i dodatnog sadržaja. Budući da je izrada 3D sadržaja i popratnih materijala i procesa generalno kompleksna za kratko pojašnjenje, cilj ovog poglavlja nije napraviti detaljni pregled, već prikazati neke brze korake koji mogu ubrzati izradu dodatnih sadržaja, korištenjem besplatnih materijala s interneta.

Za detaljne upute korisno je pretražiti mnogobrojne sadržaje dostupne na internetu.

Ovaj kratki pregled počinje od pretpostavke da je pribavljen određeni besplatni materijal, primjerice lik koji čini igrača. Korištenjem 3D alata Blendera 2.8. za modeliranje kreira se primjerice magični šešir. Šešir se dodaje igraču na glavu, kako bi ga pratio u postojećoj animaciji kretanja. Potrebno je odraditi ***„parenting“*** na drugi predmet, u ovom slučaju igrača. Odabere se objekt šešira te drugi objekt (igrač), tako da oba budu označena. Bira se spajanje s vrhom (engl. *vertex*), kako bi objekt pratio pomak i kretnju samog roditeljskog objekta. Primjer toga je dodan šešir igraču, koji se sada u animaciji kreće kao i roditelj.

Slična stvar je i s kreiranim mačem s tim da se u ovom slučaju odabere mač, a potom odabere kost šake na glavnom objektu (primjerice čovjeku u ovom slučaju). Tako se primjerice spoji mač i kost (zglob) ruke pa mač prati putanju ruke tokom aktivne animacije. Ovo uvelike olakšava manipulaciju i brzo kreiranje različitih animacija s različitim predmetima, koji se naknadno mogu dodati.

Za potrebe izrade prototipa i brzog kreiranja animacija, preporuka je koristiti Mixamo, web stranicu za automatsku animaciju likova.

3.6.2. Kamera i teksture

Kreirano je osam izometrijskih kamera, za osam strana izometrije u igri. Bitne stavke su rotacije kamera čije su vrijednosti X = 54.7 stupnjeva, Y = 0, a Z kutevi redom: 45, -45, 90, 135, -135, 180, 270, 360 stupnjeva.

Slijedi primjer za otvaranje i spajanje postojećih (djelomično transparentnih) tekstura unutar besplatnih materijala. Kada se otvori ***.blend*** file s primjerice objektom stabla, dodaju se ručno pripadajuće teksture na postojeći podobjekt debla i listova na ***principle BSFD*** materijal. Potom je potrebno ući u ***Shader editor*** i spojiti alfa kanal teksture s alfa kanalom iz ***principle BSFD .blend*** materijala. Potrebno je to učiniti za sve materijale (i teksture). Sada će se objekt moći normalno iscrtati, uključujući transparentne teksture, i koristiti za potrebe projekta.

3.6.3. Iscrtavanje scene i izvlačenje slika

Proces iscrtavanja scene i likova vrši se na sljedeći način. Stvoreni i animirani lik ili objekt postavljen je u centar područja manipulacije u Blender ***Viewport***-u. Ispod objekta napravljena je ploha (engl. *plane*) koja je proširena tako da oponaša tlo (zapravo će hvatati sjenu). Dodano je svjetlo (sunce ili točka, različitih intenziteta, ovisno o potrebi).

U ***Object properties*** kartici napravljene plohe potrebno je uključiti opciju ***Visibility -> Shadow Catcher***, kako bi ploha hvatala sjenu, a pod ***Render*** karticom objekta odabrati opciju ***Film -> Transparent***, kako bi podloga plohe za hvatanje sjene bila prozirna. Za omogućavanje opcije ***Shadow Catcher***, potrebno je iscrtavanje izvesti s iscrtavačem ***Cycles render***, koji se odabire pod opcijom ***Render properties -> render engine -> Cycles***.

Odabere se jedna po jedna kamera, te se na alatnoj traci u zaglavlju Blendera odabere kartica ***Render -> Render animation***. Pričeka se završetak iscrtavanja svih sličica animacije, koje se mogu pronaći na definiranom mjestu ili pod `C://tmp` na Windows operativnom sustavu.

Za potrebe ovog rada sve sličice pojedinih animacija su ubačene u radnu mapu Godot projekta, te su dodane u ***AnimatedSprite*** čvor unutar kinematičkih tijela (igrač, neprijatelji itd.) pod zasebnim imenima. Format sličica je **.png**.

4. ZAKLJUČAK

Izrada igre izazovan je i zahtjevan proces. Specifični *RPG* žanr igara, dodatno povećava kompleksnost svojim standardima miješanja više žanrova.

Cilj ovog rada bio je napraviti funkcionalnu igru, u pokretaču igre i jeziku koji autor ne poznaje, bez prijašnjeg iskustva u izradi igara. Za izradu igara potrebna je velika kreativnost, vrijeme te volja za učenjem. Budući da je Godot još uvijek novost među pokretačima igara, a dostupni materijali siromašni, veliki dio vremena učenja i izrade sveo se na izvlačenje dijelova i djelića znanja i literature potrebnih da se napravi konačni rad.

Završena igra ima većinu elemenata prikladnih *RPG* žanru: praćenje iskustva i otključavanje mogućnosti, istraživanje, borbeni element, prijateljske i neprijateljske likove, zadatke, dijalog, glazbu, inventar, skupljanje i korištenje predmeta te dodatne funkcionalne elemente poput pohrane i učitavanja spremljene igre.

Uspješno su riješeni problemi i proučen pokretač igre Godot, a završni rezultat moguće je dalje nadograđivati i koristiti kao znanje za druge projekte.

5. LITERATURA I RESURSI

[1] https://docs.godotengine.org/en/stable/index.html (09.08.2020.)

[2] A. Manzur, G.Marques, Godot Engine Game Development in 24 Hours, Sams Teach Yourself: The Official Guide to Godot 3.0, 1st Edition, Kindle Edition

[3] https://www.reddit.com/r/godot/ (09.08.2020.)

[4] https://www.youtube.com/c/GameDevelopmentCenter/ (14.7.2020.)

[5] Lik igrača – osnovni 3D model: https://www.youtube.com/watch?v=gMnH9vvMLYg (20.02.2020.) modificiran, dodane teksture i boje, samostalno napravljene kosti, IK kosti, „težinsko bojanje“, animacije kretanja i napada lika u Blenderu

[6] Magični šešir – samostalno napravljen 3D model u Blenderu, tekstura Krita

[7] Igračev mač – samostalno napravljen 3D model u Blenderu, tekstura Krita

[8] Kokoš – https://www.turbosquid.com/3d-models/christmas-chicken-grey-art-3d-1266316 (11.06.2020) samostalno napravljene kosti, IK kosti i animacija. Modulacija boje u Godotu

[9] Zvuk kokoši – https://freesound.org/people/Rudmer_Rotteveel/sounds/316920/ (24.06.2020.) CC0

[10] Font Anglodavek – https://www.fontspace.com/anglodavek-font-f20512 (01.06.2020.) freeware licencija

[11] Magično stablo – https://www.cgtrader.com/free-3d-models/plant/other/12-stylized-trees-in-4-texture-options (12.6.2020.) modificirano u Blenderu te izvučene slike

[12] Kameni div – https://www.cgtrader.com/free-3d-models/character/fantasy/ancient-earth-golem-v2 (12.6.2020.)

[13] Ikone zaglavlja inventara – https://opengameart.org/content/simple-stat-icons (17.06.2020.) CC0

[14] Gljive, kamenčići – https://opengameart.org/content/10-free-isometric-graphics-from-3d-models (23.06.2020.) CC0

[15] Svitak – https://opengameart.org/content/low-poly-scroll (21.06.2020.) CC0, obrađen u Blenderu, dodatna grafička obrada u alatu Krita

[16] Model kostura – https://www.turbosquid.com/FullPreview/Index.cfm/ID/659093 (31.05.2020.) animiran u Mixamo-u, izvučene slike preko Blendera

[17] Odjeća igrača – samostalni rad u Blenderu – 3D izrada, animacija odjeće, izrada teksture

[18] Žaba – zvuk https://opengameart.org/content/mutant-frog (13.06.2020.) CC0

[19] Žaba model – https://www.cgtrader.com/free-3d-models/animals/reptile/arrow-frog-rigged-vr-ar-low-poly-3d-model (12.06.2020.) animiran u Blenderu

[20] Magično jezero – samostalni rad u alatu Krita

[21] Spojena jezera – https://pixabay.com/photos/lake-aerial-view-nature-landscape-2755907/ (11.06.2020.) modifikacija u alatu Krita te Godotu

[22] Špilja (unutrašnjost) – samostalni rad u alatu Krita

[23] Slike – Ivan the Rose, Villager, Old man https://opengameart.org/content/public-domain-portraits (24.06.2020.) obrađene u alatu Krita.

[24] Ivan the Rose – 3D model je modificirani model igrača u Blenderu. Dodan šešir, teksture. Animiran u Mixamo-u

[25] Villager i Old man 3D modeli – https://www.turbosquid.com/3d-models/male-character-3ds/525305 (04.07.2020.)

[26] Ikone predmeta i dijela magije – https://opengameart.org/content/skill-item-and-spell-icons (11.12.2019.) CC0, dio je obrađen u alatu Krita

[27] Dio magije – https://opengameart.org/content/spell (07.03.2020.) CC0

[28] Dio magije – https://opengameart.org/content/nature-spell (07.03.2020.) CC0

[29] Bunar, kočija, bačve, kovčezi, posudice, kutije, kante – https://opengameart.org/content/isometric-medieval-props (06.07.2020) CC0

[30] Ruševine i zidići – https://opengameart.org/content/free-isometric-ruins (23.06.2020.) CC0

[31] Stolica u selu – https://opengameart.org/content/isometric-dungeon-tiles-60 (25.11.2019.) CC0

[32] Kuće u selu – https://opengameart.org/content/isometric-log-cabins (04.06.2020.) CC0 obrada u alatu Krita

[33] Stablo (varijacija 1) – https://opengameart.org/content/isometric-trees-1 (04.06.2020.) CC0 obrada u Blenderu

[34] Tileset brda, ulaza u špilju, dijela trave i biljaka te tla – https://opengameart.org/content/grassland-tileset-1 (02.06.2020.) CC0

[35] Tileset tla – https://opengameart.org/content/isometric-ground-tiles (30.05.2020.) CC0

[36] Teksture za magiju (Spritesheets) – https://opengameart.org/content/animated-particle-effects-1 (10.03.2020.) CC0

[37] Čamac – https://www.turbosquid.com/3d-models/free-boat-3d-model/1094364 (11.06.2020.) animiran u Blenderu

[38] Intro Godot logo – https://commons.wikimedia.org/wiki/File:Godot_logo.svg (01.06.2020.) CC by 3.0

[39] Duhovi – samostalni rad u Blenderu (model, tekstura)

[40] Novčić (inventar), dugme izlaza (inventar), bobice - samostalni rad u alatu Krita

[41] Lopoči – https://www.turbosquid.com/3d-models/free-obj-mode-water-lily-flower/1125153 (12.06.2020.)

[42] Dugmad glavnog izbornika – samostalan rad u alatu Krita

[43] Pozadina glavnog izbornika – https://pixabay.com/photos/wall-background-texture-grunge-1217083/ (23.05.2020.) obrada u alatu Krita

[44] Slika šume i potoka u glavnom izborniku – https://pixabay.com/photos/background-image-fantasy-forest-3497025/ (23.05.2020.) obrada u alatu Krita

[45] Tekstura dugmeta za spremanje stanja igre – https://pixabay.com/photos/background-leather-brown-closeup-1838494/ (07.03.2020.) te dodatna tekstura iz alata Krita

[46] Tekstura zdravlja igrača i neprijatelja – samostalni rad u alatu Krita

[47] Tekstura okvira kugle zdravlja i mjesta u traci vještina – https://opengameart.org/content/health-orb-11 (26.05.2020.) CC0, dorađeno u alatu Krita

[48] Inventar – dugmad (tekstura) i pozadina https://pixabay.com/photos/cement-gray-grey-texture-concrete-340454/ (16.05.2020.) obrada u alatu Krita

[49] Trava, biljke i grmlja, bambus – https://opengameart.org/content/free-isometric-plants-pack (26.11.2020.) CC0

[50] Zvuk čovjeka, pozadinska glazba (gitara) – autorski rad, obrada u alatu Audacity

[51] Znak kraj jezera – https://opengameart.org/content/wooden-sign CC0 (24.06.2020.)

[52] Kotačić (ItemPopupPanel) – https://pixabay.com/vectors/gear-mechanics-settings-icon-1119298/ (02.06.2020.)

[53] Stablo (varijacija 2) – https://www.turbosquid.com/3d-models/pine-tree-model-1508559 (09.06.2020.) iscrtavanje i obrada u Blenderu

[54] Stablo (varijacija 3) – https://www.turbosquid.com/3d-models/3d-evergreen-tree-1346603 (12.06.2020.) iscrtavanje i obrada u Blenderu

[55] Grobovi – samostalni rad u Blenderu

[*] Svi korišteni materijali djelo su autora ovoga rada ili su sadržaji dostupni pod besplatnim licencama koji se mogu koristiti u akademske svrhe