Izrada računalne igre u Godot razvojnom okruženju

Pelko, Eva

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

EVA PELKO

IZRADA RAČUNALNE IGRE U GODOT RAZVOJNOM OKRUŽENJU

Završni rad

Pula, rujan, 2024.

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

EVA PELKO

IZRADA RAČUNALNE IGRE U GODOT RAZVOJNOM OKRUŽENJU

Završni rad

JMBAG: 0303094914, redovni student

Studijski smjer: Informatika

Predmet: Dizajn i programiranje računalnih igara

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan, 2024.

Sadržaj

1. Uvod

Kroz ovaj završni rad dokumentiran je razvoj 2D platformera pod nazivom Lock Zero. Lock Zero prati hrabru junakinju koja pokušava riješiti misterij njenog svijeta. Vizualni stil je inspiriran retro igrama poput "Super Mario Bros.".

Jedan od najvažnijih i najutjecajnijih žanrova u povijesti video igara su platformeri. Platformer je žanr video igara koji se fokusira na navigaciju likova kroz različite prepreke i razine pomoću skakanja, trčanja i penjanja [1]. Glavni cilj igre obično uključuje prelazak s jedne točke na drugu, izbjegavanje neprijatelja i prikupljanje različitih predmeta ili bodova. U platformerima, dizajn razina je ključan, a svaka razina često predstavlja jedinstvene izazove i prepreke koje igrač mora prevladati.

Prve platformerske igre pojavile su se krajem 70-ih i početkom 80-ih godina. "Donkey Kong" iz 1981. godine, kojeg je razvio studio Nintendo, smatra se jednom od prvih pravih platformera. Igrač je upravljao likom Jumpman-om, kasnije poznatim kao Mario, koji je morao skakati preko prepreka i penjati se po ljestvama kako bi spasio princezu. Ova igra postavila je temelje za mnoge buduće igre i serijale.

Tijekom 80-ih i 90-ih godina, platformeri su dominirali tržištem video igara. Naslovi "Super Mario Bros." (1985) i "Sonic the Hedgehog" (1991) postali su ikonama žanra. Ove igre su karakterizirale brze akcijske sekvence, precizne kontrole i složene razine koje su poticale igrače na istraživanje i ponavljanje. "Super Mario Bros." posebno je poznat po svojoj revolucionarnoj mehanici skakanja i fluidnom dizajnu razina, koji su inspirirali mnoge buduće igre.

Kako je tehnologija napredovala, tako su i platformeri postajali sve sofisticiraniji. Pojava 3D grafike u kasnim 90-ima donijela je novi val platformera, kao što su "Super Mario 64" i "Crash Bandicoot". Ovi naslovi su unijeli novu dimenziju u žanr, omogućujući igračima istraživanje trodimenzionalnih svjetova s većom slobodom kretanja.

U novije vrijeme, povratak retro estetike i nezavisne scene razvoja igara doveo je do renesanse 2D platformera. Igre poput "Shovel Knight" i "Hollow Knight" uspješno su kombinirale klasični dizajn s modernim tehnikama igranja i pričanja priča. Među njima, Celeste iz 2018. godine ističe se po iznimnoj kombinaciji “retro” osjećaja i fluidnih kontrola.

Inspiriran igrom Celeste, ovaj završni rad fokusira se na razvoj 2D platformera koristeći Godot razvojno okruženje. Godot je open-source engine koji pruža fleksibilnost i snagu potrebnu za stvaranje složenih igara. Ovaj projekt teži replicirati estetiku i mehaničke principe koji su Celeste učinili uspješnim, dok uvodi vlastite inovacije i jedinstvene elemente dizajna.

Kroz ovaj rad, istraženi su izazovi i rješenja vezana uz razvoj 2D platformera, s posebnim naglaskom na grafiku, kontrolu likova i dizajn razina. Cilj je stvoriti ne samo tehnički impresivnu igru, već zabavno iskustvo igračima.

Lock Zero GitHub: https://github.com/EvaPelko/zavrsni-platformer

Link na build: https://drive.google.com/drive/folders/1llT9eCnXEl-PlB7Oby18EyZrEBWIg5F1?usp=drive_link

2. Analiza tržišta

Analiza tržišta je ključni korak kod dizajniranja i razvoja igre jer pruža vrijedne uvide koji mogu pomoći razvojnom timu da bolje razumije svoju ciljanu publiku, konkurenciju i trenutne trendove u industriji videoigara. Ovaj postupak pomaže u donošenju informiranih odluka, smanjenju rizika i povećanju šansi za uspjeh igre na tržištu.

Analiza tržišta pomaže razvojnom timu da precizno definira svoju ciljanu publiku, tj. demografski profil igrača koji bi najvjerojatnije igrali njihovu igru.

Njom je moguće identificirati konkurente koji razvijaju slične igre. Pregled konkurenata pomaže u razumijevanju što njihove igre čini uspješnima ili neuspješnima, te omogućuje razvojnom timu da izbjegne greške koje su drugi napravili ili da usvoji uspješne elemente.

Omogućuje timu da identificira koje platforme (PC, konzole, mobilni uređaji) su najpopularnije među ciljanom publikom.

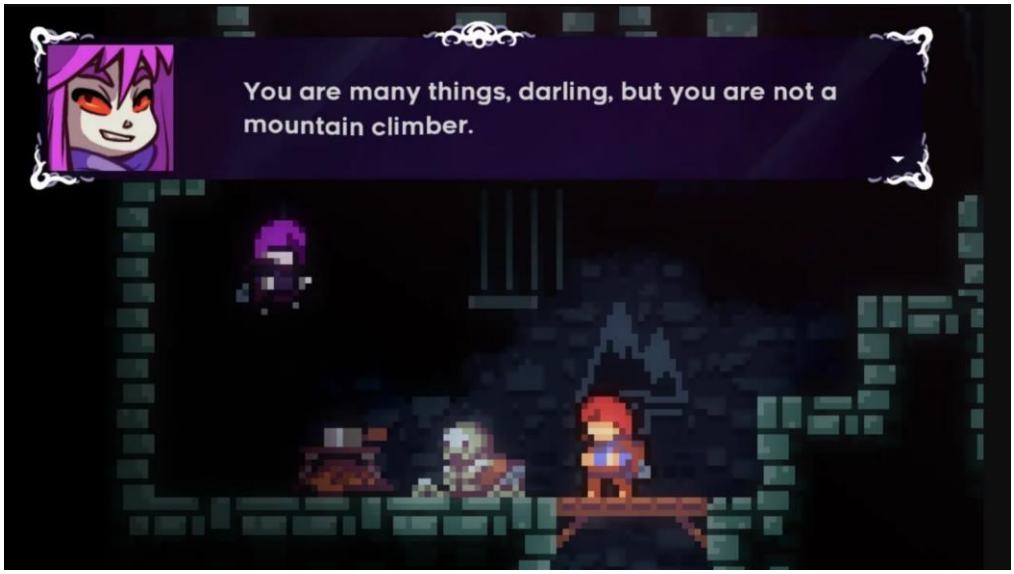
Analiza tržišta pomaže u definiranju marketinške strategije. Razumijevanje ciljane publike i konkurenata omogućuje razvojnom timu da osmisli učinkovit način promocije igre putem kanala koje njihova publika najviše koristi (društvene mreže, YouTube, Twitch itd.).

Za analizu tržišta odabrane su sljedeće dvije igre koje su doživile veliki komercijalni uspjeh i postale kulturalnim fenomenom među igračima.

2.1. Celeste

Celeste je 2D platformer video igra iz 2018. koju je razvio i objavio indie studio Maddy Makes Games. Igrač kontrolira Madeline, mladu ženu s anksioznošću i depresijom kojoj je cilj popeti se na planinu Celeste.

Glavna inspiracija je zato jer se ističe svojom izazovnom mehanikom igranja, preciznim kontrolama i emocionalnom pričom. "Celeste" je savršen primjer kako kombinacija jednostavne mehanike i dubokog narativa može stvoriti nezaboravno igračko iskustvo. Pikselizirani umjetnički stil koji se može vidjeti na slici 1 i pažljivo dizajnirane razine dodatno naglašavaju atmosferu igre, a njena sposobnost da poveže igrače s likovima i njihovim unutarnjim borbama je nešto što je poželjno replicirati.



Slika 1. Snimak ekrana iz igre Celeste [2]

Dizajn razina koji se može vidjeti na slici 2 iznimno je detaljno promišnjen, što je inspiracija mnogim dizajnerima igrica.

Celeste je moguće iskusiti na sljedećim platformama: Nintendo Switch, PlayStation 4, Linux, macOS, Microsoft Windows, Xbox One, Google Stadia.

Ciljana publika su mladi ljudi otvoreni razgovoru o mentalnom zdravlju i bilo tko tko se osjeća drugačije.

Slika 2. Snimak ekrana iz igre Celeste [2]

2.2. Undertale

Undertale je 2D video igra uloga koju je 2015. stvorio američki indie programer Toby Fox, a snimak ekrana je prikazan u slici 3. Igrač kontrolira dijete koje je palo u Podzemlje: veliku, osamljenu regiju ispod površine Zemlje, odvojenu čarobnom barijerom.

Njena jedinstvena priča, koja uspijeva biti duboko emocionalna i često duhovita, te inovativne mehanike koje razbijaju konvencionalne granice RPG žanra, potiču na razmišljanje o tome kako narativne i gameplay elemente spojiti na nove i uzbudljive načine. "Undertale" je dokaz da igre mogu biti snažan medij za pripovijedanje.

Undertale je moguće iskusiti na sljedećim platformama: Nintendo Switch, PlayStation 4, Linux, macOS, Microsoft Windows, Xbox One, Playstation Vita.

Ciljana publika su svi ljubitelji retro igara s neobičnim mehanikama igranja.

Slika 3. Snimak ekrana iz Undertale videoigre [3]

3. Razvojno okruženje

3.1. Godot

Godot je open-source engine za razvoj video igara, poznat po svojoj fleksibilnosti, jednostavnosti korištenja i snažnim mogućnostima. Razvijen je s ciljem da bude pristupačan za početnike, ali dovoljno sposoban za profesionalce. Odabrana verzija Godot-a je 4.2.2-stable, što je među novijim verzijama ovog razvojnog okruženja u vrijeme pisanja rada.

Ovaj engine nudi odvojena, specijalizirana okruženja za razvoj 2D i 3D igara. 2D engine uključuje podršku za piksel-savršenu fiziku, fleksibilno osvjetljenje i animacije, dok 3D engine podržava moderne grafičke efekte, shader-e i alat za fiziku [4].

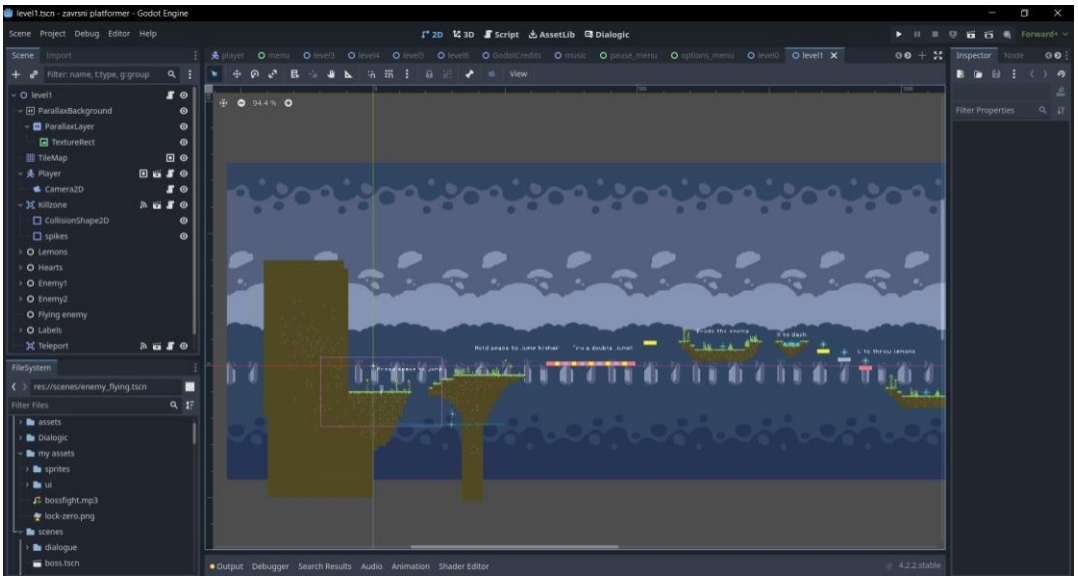
Koristi se jedinstveni sustav scena i čvorova (eng. nodes). Svaka scena sadržava hijerarhiju čvorova kao što se može vidjeti na slici 4, što omogućava modularni i organizirani pristup razvoju igara. Scene se mogu ponovno koristiti i ugnijezditi, što olakšava upravljanje kompleksnim projektima.

Slika 4. Struktura čvorova u Godot razvojnom okruženju na primjeru player scene

Glavni jezik za skriptiranje u Godot-u je GDScript, jezik sličan Python-u koji je dizajniran za jednostavno i brzo programiranje. Godot također podržava VisualScript (vizualno skriptiranje) i druge jezike poput C# i C++.

Budući da je open-source projekt, Godot je besplatan za korištenje bez ikakvih naknada za licenciranje. Aktivna zajednica doprinosi razvoju engine-a, redovito dodajući nove značajke i poboljšanja.

Slika 5 je snimak ekrana u razvojnom okruženju.



Slika 5. Snimak ekrana u Godot razvojnom okruženju

3.2. Aseprite

Aseprite je popularan alat za stvaranje piksel arta i animacija, posebno cijenjen među indie razvojnim programerima i umjetnicima zbog svojih bogatih mogućnosti.

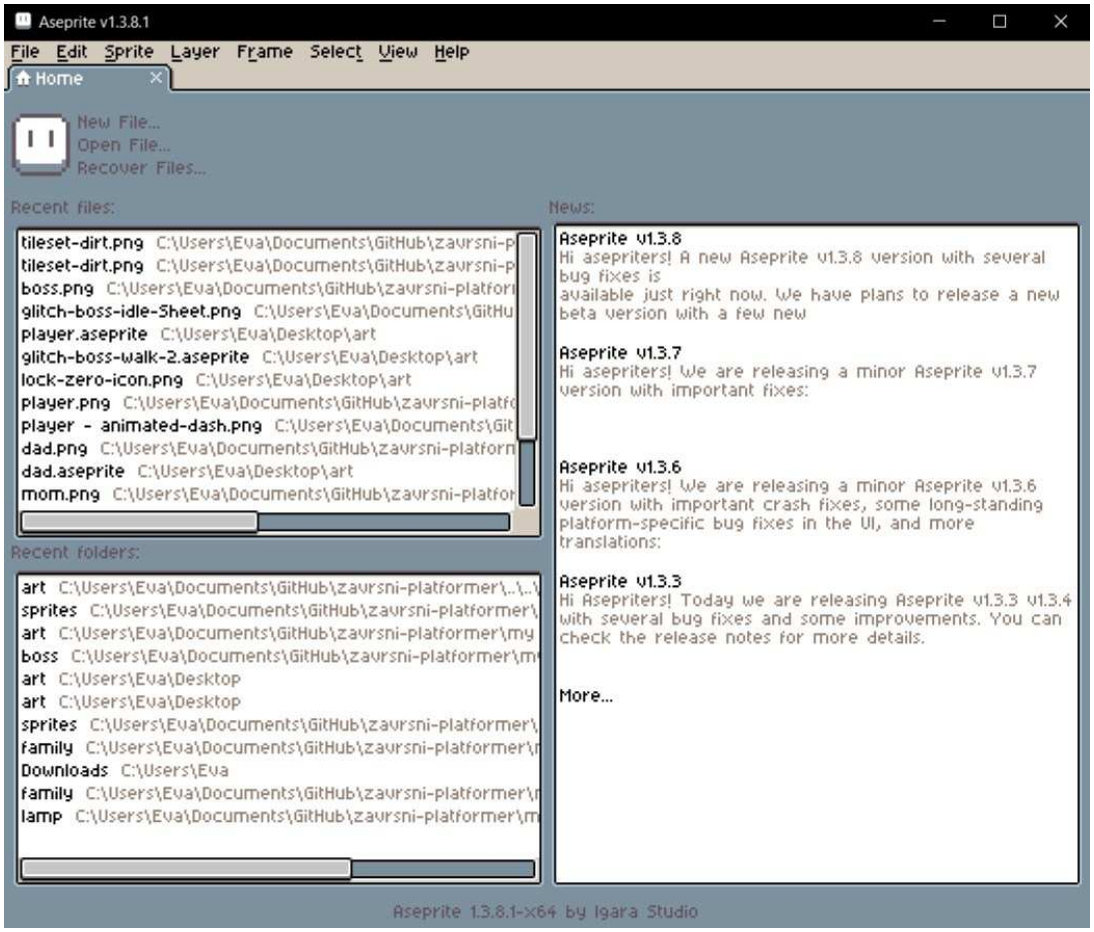
Posebno je dizajniran za piksel art, što znači da su svi alati i funkcije optimizirani za rad na nivou pojedinačnih piksela. To uključuje jednostavne alate za crtanje, poput olovke, kante za bojanje, gumice i gradijente, što se može vidjeti na slici 6.

Slika 6. Snimak korisničkog sučenja u Aseprite-u

On omogućava jednostavno stvaranje i uređivanje animacija. Korisnici mogu raditi s više slojeva i okvira (frames), stvarajući složene animacije s detaljnim kontrolama. Tu su i funkcije poput onion skinning-a, koje omogućuju pregled prethodnih i sljedećih okvira za precizno animiranje.

Aseprite podržava uvoz i izvoz različitih formata datoteka, uključujući GIF, PNG, BMP, i specifične formate za piksel art poput .ase i .aseprite, što olakšava dijeljenje i integraciju s drugim alatima i platformama kao što je prikazano na slici 7.

To je idealan alat za umjetnike koji se bave piksel artom i animacijom, budući da pruža sve potrebne alate i funkcije za stvaranje detaljnih i visoko kvalitetnih radova. Njegova specijaliziranost, jednostavnost korištenja i bogate mogućnosti čine ga popularnim izborom među indie razvojnim programerima i umjetnicima.



Slika 7. Snimka Aseprite programa početnom ekranu

4. Elementi platformer igrica

Platformerski žanr videoigara, često poznat jednostavno kao "platformeri," je jedan od najprepoznatljivijih i najdugovječnijih u povijesti videoigara, stoga je potrebno istražiti ključne elemente koji čine srž žanra.

4.1. Kontrole i kretanje

Kontrola lika i njegovo kretanje su temelj svakog platformera. Igrači obično preuzimaju kontrolu nad protagonistom kojeg vode kroz razine koje uključuju skakanje preko prepreka, penjanje po platformama i izbjegavanje ili eliminaciju neprijatelja.

Preciznost kontrola je ključna; dobro osmišljeni platformeri omogućuju igračima da točno i brzo reagiraju na izazove, čineći kretanje glatkim i zadovoljavajućim. Dobri primjeri uključuju igre poput "Super Mario Bros." i "Celeste," gdje je osjećaj kontrole i odgovora na kontrole neusporediv.

Cilj je učiniti da se protagonist kreće kako igrač želi, a ne točno kako igrač koristi kontrole.

4.2. Dizajn razina

Dizajn razina u platformerima zahtijeva pažljivo balansiranje izazova i nagrada. Razine su obično slojevite i nelinearne, s različitim putevima i skrivenim područjima koja potiču istraživanje. Dobar dizajn razina postupno uvodi nove mehanike i prepreke, omogućujući igračima da uče i usavršavaju svoje vještine, kako napreduju kroz igru. Pritom je ključno provjeriti da razine nisu predvidljive, već da nude osjećaj postignuća prilikom savladanja prepreka.

Skok od igrača treba biti dosljedan i predvidljiv. Razmaci između platformi trebaju biti konzistentni, sukladni s udaljenosti skoka od igrača. Igrač mora intuitivno znati koje razmake može pouzdano preskočiti [5].

4.3. Estetika i atmosfera

Vizualni stil i glazba u platformerima značajno doprinose ukupnom iskustvu igre. Estetika može varirati od jednostavnih, pikseliziranih grafika do bogatih, ručno crtanih animacija. Zvuk i glazba također igraju vitalnu ulogu u stvaranju atmosfere igre. Na primjer, "Ori and the Blind Forest" koristi predivne vizuale i emocionalnu glazbu kako bi stvorio duboko uranjajuće iskustvo, dok igre poput "Super Meat Boy" koriste brzi ritam i žarke boje za dinamično i adrenalinsko iskustvo.

4.4. Mehanika igranja

Mehanika igranja u platformerima može uključivati razne elemente kao što su skupljanje predmeta, borba s neprijateljima, rješavanje zagonetki i manipulacija okolišem. Raznolikosti je ključ; uspješni platformeri kombiniraju različite mehanike kako bi zadržali interes igrača. Primjerice, "Rayman Legends" uvodi inovativne elemente kao što su sinkronizirani ritmički dijelovi, dok "Shovel Knight" miješa borbu, istraživanje i prikupljanje blaga.

4.5. Progresija i nagrade

Progresija u platformerima često se ostvaruje kroz sustav razina ili svjetova koje igrači moraju savladati. Nagrade mogu biti varirati: otključavanje novih sposobnosti, otkrivanje skrivenih područja, ili postizanje visokih rezultata. Nagrade motiviraju igrače da se vrate i ponovno igraju razine kako bi poboljšali svoj rezultat ili otkrili sve tajne.

4.6. Priča i likovi

Iako priča nije uvijek središnji element platformera, mnogi moderni naslovi uključuju narativne elemente kako bi poboljšali iskustvo igrača. Likovi su često karizmatični i lako prepoznatljivi, što pomaže igračima da se povežu s njihovim ciljevima i izazovima. Primjerice, "Hollow Knight" koristi minimalistički, ali duboko emotivan pristup pričanju priče, dok "Celeste" istražuje teme mentalnog zdravlja kroz osobnu priču protagonistice [6].

5. Proces razvoja igrice

Prvi korak koji korisnik treba poduzeti je odabir verzije Godot razvojnog okruženja. Potrebno je uzeti u obzir nekoliko faktora koji se odnose na specifične potrebe projekta, podršku za različite platforme i dostupne značajke.

Ako korisnik traži verziju za produkciju, preporučuje se odabrati stabilnu verziju. Stabilne verzije prolaze kroz temeljito testiranje i idealne su za komercijalne ili dovršene projekte.

Ako korisnik želi eksperimentirati s novim značajkama koje još nisu službeno objavljene, može isprobati beta ili alpha verzije. No, ove verzije mogu sadržavati greške (eng. bugs).

Postoje dvije glavne verzije ovog razvojnog okruženja: Godot 3.x i 4.x.

Ako korisnik započinje novi projekt i želi iskoristiti najnovije značajke, poput naprednog renderiranja, Forward+ renderer-a, Vulkan podrške, i poboljšane performanse u 3D igrama, tada je **Godot 4.x** najbolji izbor. Verzija 4.x donosi mnoge značajne promjene, ali može zahtijevati više hardverskih resursa, posebno za 3D projekte.

Ako je projekt već razvijan u seriji 3.x ili ako korisnik radi na mobilnim platformama ili starijim uređajima, može se odlučiti za Godot 3.5 ili slične verzije. **Godot 3.x** je zreliji za 2D igre, ima niže hardverske zahtjeve i mnogi korisnici su već upoznati s njegovim radnim procesima.

Kada se korisnik odluči za verziju, može je preuzeti sa službene stranice Godot razvojnog okruženja. Dostupne su verzije za različite operacijske sustave (Windows, macOS, Linux). Instalacija Godot engine-a je izuzetno jednostavna.

Godot dolazi kao jedna samostalna izvršna datoteka (eng. executable) koja ne zahtijeva instalacijski proces. Korisnik samo treba preuzeti datoteku sa službene web stranice i odmah je može pokrenuti bez potrebe za instaliranjem dodatnih programa ili ovisnosti.

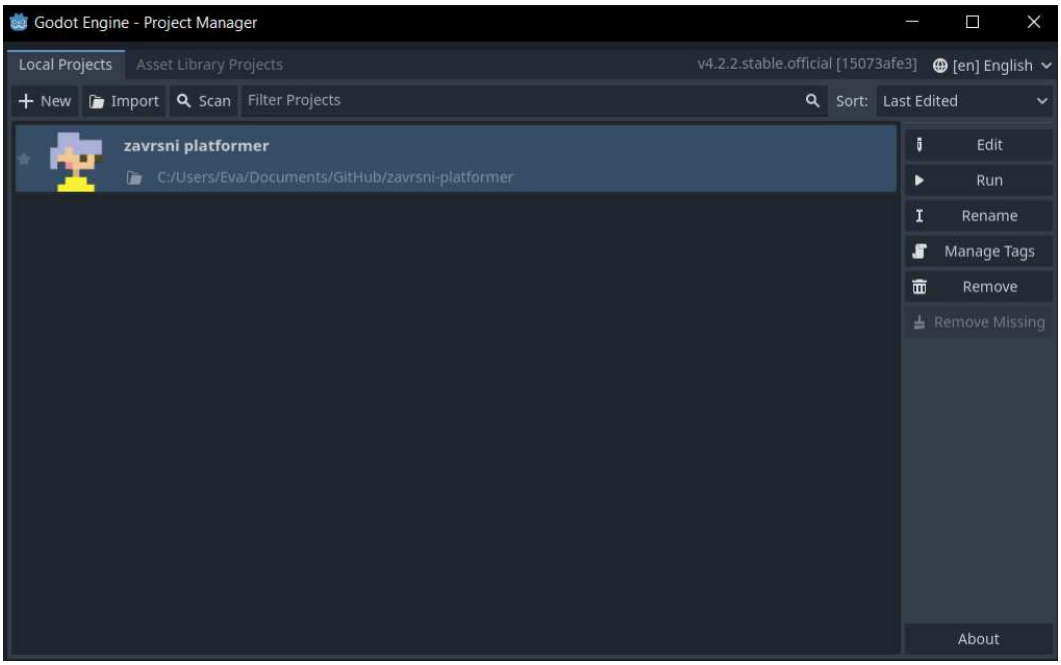
Alternativno, korisnik može kompilirati Godot s njihovog GitHub repozitorija, što omogućuje korisniku da koristi najnovije značajke i izmjene koje još nisu službeno objavljene. Proces zahtijeva preuzimanje izvornog koda i kompiliranje engine-a lokalno na računalu.

Zbog potreba ovog projekta, odlučeno je koristiti 4.2.2-stable verziju. Projekt koristi Dialogic plugin, kojemu je potrebna verzija 4.2.2 nadalje. Budući da je plugin bio implementiran kasnije u razvoju projekta, bilo je potrebno preći s 4.1.1-stable verzije na 4.2.2-stable. Prijelaz je bio jednostavan i potrebno je bilo promijeniti samo par linija koda.

Razvoj projekta je bio primarno obavljen na Windows operacijskom sustavu, a ponekad i na macOS operacijskom sustavu. Prijelaz između operacijskih sustava je bio bezbolan i vrlo jednostavan.

5.1. Prvi koraci u izradi projekta

Izrada bilo koje Godot igre započinje kreiranjem novog projekta u Godot Project Manager-u koji je prikazan na slici 8. Za stvaranje novog 2D projekta u Godot 4 engine-u koji koristi pikselizirane resurse, korisnik treba slijediti nekoliko koraka. Prvo, treba pokrenuti Godot 4 i odabrati opciju "New Project", unijeti ime projekta i odabrati direktorij za pohranu. Nakon toga, otvara se editor.



Slika 8. Godot Project Manager

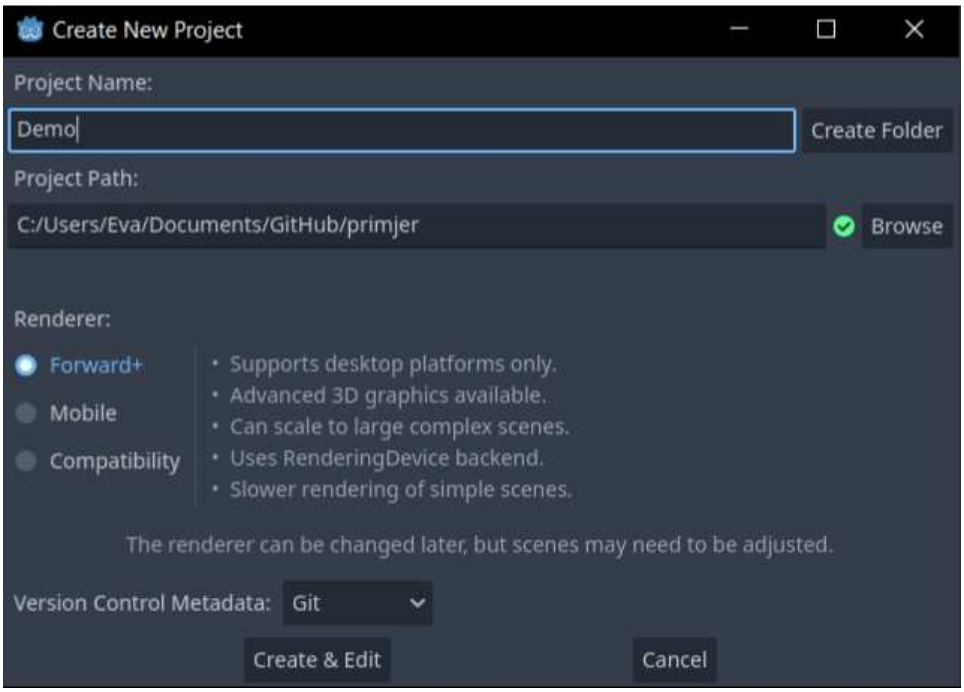
Kada korisnik kreira novi projekt u Godot 4 engineu u Create New Project prozoru koji je prikazan na slici 9, jedan od važnih koraka je odabir renderer opcija, koje definiraju način na koji će se grafika prikazivati u igri. Ove opcije imaju značajan utjecaj na performanse i vizualnu kvalitetu igre, a Godot 4 nudi nekoliko renderer opcija, svaka s različitim prednostima i ciljanom primjenom.

Forward+ je zadani renderer u Godot 4, koji pruža dobar balans između performansi i vizualne kvalitete. Omogućuje napredne efekte osvjetljenja, poput dinamičnog osvjetljenja, sjena i refleksija. Koristi se za većinu modernih igara, jer može prikazati više izvora svjetlosti bez značajnog utjecaja na performanse.

Mobile je optimiziran za mobilne uređaje i sustave s nižim grafičkim mogućnostima. Fokusira se na osiguravanje visoke razine performansi, ali uz kompromis u vizualnoj kvaliteti, bez podrške za napredne grafičke efekte poput refleksija i dinamičnih sjena.

Compatibility renderer osmišljen je kako bi pružio najveću moguću kompatibilnost s različitim hardverskim konfiguracijama, uključujući starije uređaje i grafičke kartice. Ovaj renderer ne podržava moderne grafičke efekte i fokusira se na minimalne zahtjeve, pružajući osnovnu vizualnu funkcionalnost.

Ovaj projekt koristi Forward+ rederer. Odabir renderer-a je moguće kasnije promijeniti u postavkama projekta.



Slika 9. Godot Create New Project

Kako bi postavio ispravne postavke za pikselizirane grafike, korisnik treba otvoriti Project Settings kroz izbornik Project. U kategoriji Display -> Window, potrebno je postaviti rezoluciju na nisku vrijednost, poput 320x180 ili 640x360, te odabrati način skaliranja "2d" s omjerom "keep". Također, preporučljivo je uključiti opciju Pixel Snap radi bolje preciznosti pri radu s pikselima.

Nakon postavljanja projekta, korisnik može uvesti pikselizirane assete tako da ih povuče u FileSystem prozor. Pri odabiru slike u prozoru Inspector, pod karticom Import, treba promijeniti opciju Filter na "Nearest" kako bi pikseli ostali oštri i isključiti Mipmaps kako bi se izbjeglo zamućivanje grafika.

5.2. Izrada player scene

Na priloženoj slici 10 je prikazana hijerarhija čvorova za Player scenu u Godot engine-u, koji predstavlja igrača u 2D igri. Hijerarhija sadrži različite čvorove koji omogućuju animacije, detekciju kolizija i zvučne efekte vezane uz igrača.

Slika 10. Hijerarhija čvorova u player sceni

Player je korijenski čvor za scenu igrača. Ovaj čvor je tipa CharacterBody2D, koji omogućuje kretanje i interakciju s drugim objektima u 2D svijetu. CharacterBody2D nasljeđuje sve funkcionalnosti od PhysicsBody2D, ali dolazi s dodatnim alatima i funkcijama specifičnim za kretanje likova, poput detekcije podloge, skakanja i klizanja. Ovaj čvor se često koristi za likove u platformerskim ili akcijskim igrama.

CharacterBody2D ima funkcije koje su pojednostavnile kretanje likova u 2D prostoru. Glavna funkcija koju korisnik koristi za kretanje je move_and_slide(), koja omogućuje kretanje lika i automatski detektira sudare s drugim tijelima ili zidovima. Omogućuje i kretanje po nagibima, stepenicama i platformama bez potrebe za dodatnim kodom za detekciju takvih prepreka.

CharacterBody2D može automatski detektirati kada je lik na tlu, što je korisno za logiku igre poput skakanja. Korisnik može upotrijebiti metodu is_on_floor() da provjeri je li lik na podlozi, što je često potrebno prilikom izrade mehanike skakanja.

AnimatedSprite2D čvor služi za prikazivanje animacija igrača koristeći različite sprite-ove. Koristi unaprijed definirane frame-ove kako bi prikazivao animaciju, kao što su hodanje, trčanje ili skakanje.

AnimationPlayer je čvor koji omogućuje upravljanje animacijama. Omogućuje korisniku da kreira, pohranjuje i pokreće različite animacije pomoću keyframe-ova. U Player sceni AnimationPlayer se koristi za flashing (hrv. bljeskanje) animaciju kada je igrač ozlijeđen.

NormalCollisionShape2D čvor predstavlja oblik kolizije koja se koristi za detekciju sudara igrača u normalnom stanju (kada igrač stoji ili hoda). Ima oblik CapsuleShape2D.

DuckCollisionShape2D čvor je dodatan oblik kolizije koji se aktivira kada igrač zauzme "čučanj" ili "duck" položaj. Njegov oblik je manji, jer igrač u ovom stanju zauzima manje prostora. Također ima oblik CapsuleShape2D.

HurtBox je posebna kolizijska zona koja određuje područje gdje igrač može primiti štetu. Ovo je drugačije od prethodne kolizije za sudare i koristi se specifično za detekciju štete. Unutar HurtBox čvora nalazi se CollisionShape2D, koji definira oblik te zone. HurtBox je najmanja kolizijska zona u Player sceni, sa svrhom da daje igraču osjećaj veće kontrole. Poželjno je dati igraču što veću šansu za uspjeh.

HurtSound je čvor tipa AudioStreamPlayer2D, koji se aktivira kada igrač primi štetu. Kad HurtBox detektira udarac, ovaj zvuk se reproducira kako bi igra pružila zvučnu povratnu informaciju o šteti.

Priložen kod u slici 11 definira klasu HurtBox, koja nasljeđuje čvor Area2D unutar Godot engine-a. HurtBox je čvor unutar scene igrača koji detektira dolazak u kontakt s drugim objektima, konkretno s objektima koji imaju HitBox i smanjuje zdravlje igrača kad se to dogodi.

```
1   class_name HurtBox
2   extends Area2D
3
4   signal received_damage(damage: int)
5
6
7   func _ready():
8       connect("area_entered", _on_area_entered)
9
10  func _on_area_entered(area: Area2D) -> void:
11      if not is_instance_of(area, HitBox):
12          return
13      if area != null:
14          Player_Health.health -= area.damage
15          received_damage.emit(area.damage)
```

Slika 11. HurtBox.gd isječak koda

Signal received_damage je definiran kako bi obavijestio ostale dijelove igre da je igrač primio štetu. Signal emitira vrijednost koja predstavlja količinu primljene štete (damage: int). Signal

može biti povezan s drugim čvorovima, kao što je Player ili korisničko sučelje (UI), kako bi se ažurirali zdravlje igrača ili pokazao vizualni efekt.

Metoda _ready() se poziva kad se scena ili čvor učita. Unutar metode, HurtBox se povezuje sa signalom area_entered, koji se emitira kada neki drugi Area2D čvor uđe u njegov prostor. Funkcija koja će se pozvati pri ulasku drugog čvora u ovaj prostor je _on_area_entered().

Metoda _on_area_entered(area: Area2D) -> void se poziva svaki put kad neki drugi Area2D čvor uđe u područje HurtBox-a. Prva stvar koju metoda radi jest provjera klase čvora koji ulazi u područje. Korisnik koristi metodu is_instance_of(area, HitBox) kako bi provjerio je li objekt koji je ušao u HurtBox neka instanca klase HitBox (pretpostavlja se da HitBox predstavlja neprijateljski napad ili sličan element koji može nanijeti štetu).

Zatim se provjerava da li je čvor koji je ušao u područje valjan (nije null). Ako je sve u redu, nastavlja se procesuiranje.

Ako je sve ispravno, metoda uzima vrijednost štete (area.damage) iz HitBox-a i smanjuje zdravlje igrača. Pretpostavlja se da je Player_Health klasa ili varijabla koja prati živote igrača, a health predstavlja trenutnu vrijednost života.

Nakon što se oduzmu životi, emitira se signal received_damage, koji obavještava ostale čvorove u igri da je igrač primio štetu, i šalje količinu štete kao argument.

Slika 12 prikazuje strukturu varijabla u player.gd skripti.

```gdscript
class_name Player extends CharacterBody2D

@export var SPEED = 100.0
const JUMP_VELOCITY = -230.0
const FALL_MULTIPLIER = 1.1  # Gravity multiplier for falling
const FALL_ANIMATION_THRESHOLD = 0.1  # Time in seconds before the fall animation plays
const EARLY_FALL_MULTIPLIER = 3.0  # Gravity multiplier for early fall
const ANTI_GRAVITY_APEX_THRESHOLD = 30.0  # Velocity threshold for anti-gravity apex
const ANTI_GRAVITY_APEX_MULTIPLIER = 0.5  # Gravity multiplier at the apex of the jump
var current_speed = SPEED
var current_jump_velocity = JUMP_VELOCITY

const DASH_SPEED = 450.0
const DASH_DURATION = 0.13  # Duration of the dash in seconds
const DASH_COOLDOWN = 0.5  # Cooldown time between dashes
const DASH_END_FALL_VELOCITY = 200.0  # Small upward velocity added at the end of the dash
const DASH_DECELERATION_RATE = 20000.0  # Rate at which dash speed decelerates after dash ends

const COYOTE_TIME = 0.2  # Time allowed to jump after running off a ledge
const JUMP_BUFFER_TIME = 0.2  # Time window to buffer the jump input

var coyote_timer = 0.0
var jump_buffer_timer = 0.0

var gravity: float
var can_doublejump = false
var fall_time = 0.0
var is_dashing = false
var dash_time = 0.0
var dash_direction = 0
var dash_cooldown_time = 0.0

var controls_enabled = true

var player_position = Vector2()
```

Slika 12. player.gd isječak koda, prikaz varijabla

Prikazani isječci koda definiraju ponašanje 2D lika (CharacterBody2D), uključujući hodanje, skakanje, padanje, bacanje projektila, "dash-a" (brzi sprint) i dvostruki skok.

Glavne varijable

- **SPEED**: Brzina kojom se igrač kreće.
- **JUMP_VELOCITY**: Brzina kojom igrač skače, negativna vrijednost jer se kretanje prema gore računa kao negativno u koordinatnom sustavu.
- **FALL_MULTIPLIER**: Faktor koji povećava gravitaciju prilikom padanja.

- **ANTI_GRAVITY_APEX_MULTIPLIER**: Faktor koji smanjuje gravitaciju na vrhu skoka (apeks), što daje osjećaj sporijeg usporavanja dok se igrač približava vrhu skoka.
- **DASH_SPEED**: Brzina kojom igrač sprinta (dash) vodoravno.
- **COYOTE_TIME**: Vrijeme koje igrač ima za skakanje nakon što napusti rub platforme (tzv. "coyote jump").

5.2.1. Glavne funkcionalnosti player skripte

Sljedeći isječci koda prikazuju kako izgledaju glavne funkcije player skripte.

1. Gravitacija i padanje

- Ako igrač nije na podu (is_on_floor()) kao što je prikazano na slici 13, gravitacija se primjenjuje. Ako igrač pada, brzina pada povećava se prema FALL_MULTIPLIER faktoru, što znači da padanje postaje brže.
- Kada igrač doseže vrh skoka (kada je vertikalna brzina mala), gravitacija se smanjuje prema ANTI_GRAVITY_APEX_MULTIPLIER, dajući osjećaj "lebdenja" na vrhu skoka.

```
# Handle gravity
if not is_on_floor():
    if velocity.y > 0:  # Falling
        velocity.y += gravity * FALL_MULTIPLIER * delta
        fall_time += delta
        if fall_time > FALL_ANIMATION_THRESHOLD:
            if not is_throwing:
                animated_sprite.play("fall")
    elif abs(velocity.y) < ANTI_GRAVITY_APEX_THRESHOLD:  # Near the apex of the jump
        velocity.y += gravity * ANTI_GRAVITY_APEX_MULTIPLIER * delta
        if not is_throwing:
            animated_sprite.play("jump")
        fall_time = 0.0  # Reset fall time when rising
    else:  # Rising or at the apex
        velocity.y += gravity * delta
        if not is_throwing:
            animated_sprite.play("jump")
        fall_time = 0.0  # Reset fall time when rising
else:
    velocity.y = 0
    fall_time = 0.0  # Reset fall time when on the floor
```

Slika 13. Implementacija gravitacije u player.gd

2. Skakanje i dvostruki skok

- Igrač može skakati ako je na podu ili koristi "coyote time" kako bi skočio kratko nakon što napusti rub platforme, što se može vidjeti na slici 14.
- Dvostruki skok omogućuje igraču da skoči još jednom dok je u zraku.

```
# Check if the player is grounded
if is_on_floor():
    coyote_timer = COYOTE_TIME
    # Check if there is a buffered jump
    if jump_buffer_timer > 0:
        velocity.y = current_jump_velocity
        jump_buffer_timer = 0
        can_doublejump = true  # Allow double jump after buffered jump
else:
    coyote_timer -= delta

if controls_enabled:
    # Handle Jump and Double Jump
    if is_on_floor():
        if Input.is_action_just_pressed("jump"):
            velocity.y = current_jump_velocity
            can_doublejump = true
    elif coyote_timer > 0 and Input.is_action_just_pressed("jump"):
        velocity.y = current_jump_velocity
        coyote_timer = 0  # Reset coyote timer after jumping
        can_doublejump = true  # Allow double jump after coyote jump
    elif Input.is_action_just_pressed("jump") and can_doublejump:
        velocity.y = current_jump_velocity
        can_doublejump = false
    elif Input.is_action_just_pressed("jump"):
        # Buffer the jump if in the air and not able to jump
        jump_buffer_timer = JUMP_BUFFER_TIME

    # Early fall handling
    if Input.is_action_just_released("jump") and velocity.y < 0:
        velocity.y += gravity * EARLY_FALL_MULTIPLIER * delta
```

Slika 14. Implementacija dvostrukog skoka u player.gd

3. Dash (Sprint)

- Kada igrač aktivira dash (brzi sprint) kao što je moguće vidjeti na slici 15, lik se brzo kreće u određenom smjeru (dash_direction) i gravitacija se privremeno isključuje.
- Nakon završetka dash-a, brzina se smanjuje i primjenjuje se mala vertikalna brzina kako bi prijelaz iz dash stanja bio gladak.

```
# Handle Dash
if is_dashing:
    dash_time -= delta
    if dash_time <= 0:
        is_dashing = false
        dash_cooldown_time = DASH_COOLDOWN
        # After the dash ends, apply a small upward velocity to smooth the transition
        velocity.y = DASH_END_FALL_VELOCITY
        velocity.x = 0  # Ensure no residual dash speed
    else:
        # Override gravity during dash
        velocity.y = 0
        # Decelerate dash speed towards the end
        var deceleration = DASH_DECELERATION_RATE * delta
        if abs(velocity.x) > deceleration:
            velocity.x -= sign(velocity.x) * deceleration
        else:
            velocity.x = 0
        velocity.x = dash_direction * DASH_SPEED
        animated_sprite.play("dash")
else:
    dash_cooldown_time -= delta
    if dash_cooldown_time <= 0 and Input.is_action_just_pressed("dash"):
        is_dashing = true
        dash_time = DASH_DURATION
        dash_direction = -1 if animated_sprite.flip_h else 1
        velocity.x = dash_direction * DASH_SPEED
        animated_sprite.play("dash")
```

Slika 15. Implementacija dash-a u player.gd

4. Bacanje projektila

- Igrač može baciti projektil (npr. limune) kao što je prikazano na slici 16 pod određenim kutom i iz određene visine. Projektili su stvoreni kao novi objekti na temelju unaprijed definiranog resursa (projectile_scene).

```gdscript
func shoot():
    if GameManager.score > 0:
        animated_sprite.play("throw")
        is_throwing = true
        var projectile = projectile_scene.instantiate()
        var direction = Vector2.RIGHT if not animated_sprite.flip_h else Vector2.LEFT
        var angle = throw_angle if animated_sprite.flip_h else -throw_angle
        var throw_direction = direction.rotated(deg_to_rad(angle))
        projectile.direction = throw_direction
        var offset = Vector2(0, throw_offset_y)
        projectile.position = global_position + offset
        get_parent().add_child(projectile)
        GameManager.subtract_point()
    else:
        print("not enough lemons")
        GameManager.show_fade_label("Not enough lemons", global_position)
```

Slika 16. shoot funkcija u player.gd

5. Buffer skoka i coyote time

- **Coyote time** omogućuje igraču da skoči iako je već napustio rub platforme (kroz određeno vrijeme).
- **Jump buffer** pamti ako je igrač pritisnuo gumb za skok prije nego što je mogao stvarno skočiti i omogućuje skakanje kad igrač dođe na pod.

6. Animacije

- Animacije igrača automatski se mijenjaju ovisno o akciji koju igrač izvodi (trčanje, skakanje, padanje, stajanje ili bacanje).
- Ako igrač kleči (duck), mijenja se oblik sudara (collision shape) kako bi se prilagodio manjoj visini igrača.

Funkcije za upravljanje životima i sporije kretanje:

- Funkcije su prikazane na priloženoj slici 17.
- **_on_player_damage_taken()**: Pokreće zvučni efekt kada igrač primi štetu i nakratko uključi nepobjedivost (eng. immortality).
- **_on_player_health_depleted()**: Kada igrač izgubi sve živote, prikazuje poruku "Umro si" i deaktivira sudar igrača.

- **slow()**: Privremeno usporava brzinu igrača i visinu skoka na 50% odnosno 85% njihove vrijednosti, a nakon 2 sekunde vraća na normalu. Funkcija se zove kada boss neprijatelj pogodi igrača projektilom.

```
func _on_player_damage_taken():
    audio_player.play()
    Player_Health.set_temporary_immortality(2.0)
    animation_player.play("flashing")
    await animation_player.animation_finished
    animation_player.stop()


func _on_player_health_depleted():
    GameManager.show_fade_label("You died", global_position)
    normal_collision_shape.set_deferred("disabled", true)
    duck_collision_shape.set_deferred("disabled", true)
    controls_enabled = false

func slow():
    current_speed *= 0.5
    current_jump_velocity *= 0.85
    var timer := Timer.new()
    add_child(timer)
    timer.wait_time = 2.0
    timer.one_shot = true
    timer.start()
    await timer.timeout
    current_speed = SPEED
    current_jump_velocity = JUMP_VELOCITY
```

Slika 17. Funkcije za upravljanje životima i sporije kretanje u player.gd

Ukupni tijek igre

- Igrač se može kretati lijevo ili desno, skakati (jednom ili dvaput u zraku), brzo sprintati, te bacati projektile. Animacije i fizika igre prilagođavaju se ovisno o akcijama koje igrač izvodi, uz upotrebu buffer-a i coyote time-a kako bi kontrola bila intuitivnija i fluidnija.

5.3. Izrada neprijatelja

5.3.1. Kopneni neprijatelj

Slika 18. prikazuje dizajn kopnenog neprijatelja. Dizajn je inspiriran crnim rupama i noćnim životinjama.

Slika 18. Prikaz kopnenog neprijatelja

Priložena slika 19 prikazuje cijeli kod kopnenog neprijatelja. U nastavku su objašnjene sve funkcije ovog isječka koda.

```
1   extends Area2D
2
3   const SPEED = 60
4   var direction = 1
5   var is_alive = true
6
7   @onready var ray_cast_right = $RayCastRight
8   @onready var ray_cast_left = $RayCastLeft
9   @onready var animated_sprite = $AnimatedSprite2D
10  @onready var audio_player = $SplatSound
11
12  # Called every frame. 'delta' is the elapsed time since the previous frame.
13  func _process(delta):
14      if not is_alive:
15          return
16
17      if ray_cast_right.is_colliding():
18          direction = -1
19          animated_sprite.flip_h = true
20      elif ray_cast_left.is_colliding():
21          direction = 1
22          animated_sprite.flip_h = false
23
24      position.x += direction * SPEED * delta
25
26  func _on_health_health_depleted():
27      is_alive = false
28      audio_player.play()
29      animated_sprite.play("die")
30      await animated_sprite.animation_finished
31      queue_free()
```

Slika 19. Isječak koda od kopnenog neprijatelja

Neprijatelj se kreće lijevo-desno, mijenja smjer kada naiđe na prepreku, i reagira na smrt uz animaciju i zvučni efekt.

Funkcija **_process(delta**) kontrolira ponašanje neprijatelja dok je živ.

if not is_alive: return: Ako neprijatelj nije živ (is_alive == false), funkcija se zaustavlja, što znači da neprijatelj prestaje reagirati ili se kretati.

if ray_cast_right.is_colliding(): Ako RayCastRight detektira prepreku (zid ili drugi objekt), neprijatelj mijenja smjer kretanja ulijevo (direction = -1), a animated_sprite.flip_h = true osigurava da se animacija neprijatelja okrene horizontalno kako bi bio usmjeren prema lijevoj strani.

elif ray_cast_left.is_colliding(): Ako RayCastLeft detektira prepreku s lijeve strane, neprijatelj mijenja smjer kretanja udesno (direction = 1), a animacija se prebacuje tako da neprijatelj gleda prema desno (animated_sprite.flip_h = false).

position.x += direction * SPEED * delta: Neprijatelj se kreće prema smjeru definiranom varijablom direction (lijevo ili desno) pomnoženim s brzinom SPEED. Vrijednost delta osigurava da se kretanje prilagođava vremenskoj razlici između frame-ova, što omogućuje glatko kretanje bez obzira na brzinu izvođenja igre.

_on_health_health_depleted() funkcija se poziva kada neprijatelj ostane bez života.

is_alive = false: Ova linija označava da neprijatelj više nije živ, što onemogućuje daljnje kretanje i reakcije neprijatelja.

audio_player.play(): Reproducira zvučni efekt smrti koristeći SplatSound, čime se dodaje zvučna povratna informacija igraču.

animated_sprite.play("die"): Pokreće animaciju smrti neprijatelja, omogućujući da se vizualno prikaže neprijateljeva smrt.

await animated_sprite.animation_finished: Ova naredba čeka da animacija smrti završi prije nego što se nastavi s ostatkom funkcije.

queue_free(): Nakon što animacija smrti završi, neprijatelj se uklanja iz igre.

5.3.2. Leteći neprijatelj

U svakom platformeru je potrebno imati više vrsta neprijatelja koji izazovu igrača na različite načine. Dodatak letećeg neprijatelja učinio je igru zabavnijom. Slika 20 prikazuje dizajn letećeg neprijatelja. Dizajn je inspiriran šišmišima i noćnim životinjama.

Slika 20. Prikaz letećeg neprijatelja

Leteći neprijatelj ima nekoliko ponašanja: patroliranje lijevo-desno, potjera za igračem kad ga detektira, i vraćanje na početnu poziciju kad igrač pobjegne iz dometa. Kod također uključuje logiku za smrt neprijatelja, gdje se neprijatelj uklanja iz igre nakon što umre.

Kao što je prikazano na slici 21, moguće je vidjeti sve varijable od letećeg neprijatelja.

```
1   extends Area2D
2
3   @export var SPEED = 60
4   @export var FLY_TIME = 3.0
5   var direction = 1 # Start by flying right
6   var fly_time_left = FLY_TIME
7   var original_direction = direction
8   var original_position = Vector2()
9   var returning_to_position = false
10  var is_alive = true
11
12  @onready var animated_sprite = $AnimatedSprite2D
13  @onready var detection_area = $DetectionArea2D
14  @onready var ray_cast_right = $RayCastRight
15  @onready var ray_cast_left = $RayCastLeft
16  @onready var audio_player = $SplatSound
17
18  var player = null
19  var chasing_player = false
20
21  func _ready():
22      original_position = position
```

Slika 21. Varijable od letećeg neprijatelja

_process(delta) funkcija prikazana na slici 22 upravlja ponašanjem neprijatelja ovisno o njegovom stanju.

```gdscript
# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
    if not is_alive:
        return

    if chasing_player:
        fly_towards_player(delta)
    elif returning_to_position:
        return_to_position(delta)
    else:
        fly_cycle(delta)

func fly_cycle(delta):
    # Check for terrain collision
    if ray_cast_right.is_colliding():
        direction = -1
        fly_time_left = FLY_TIME # Reset fly time when terrain is detected
    elif ray_cast_left.is_colliding():
        direction = 1
        fly_time_left = FLY_TIME # Reset fly time when terrain is detected

    fly_time_left -= delta
    if fly_time_left <= 0:
        direction *= -1
        fly_time_left = FLY_TIME

    position.x += direction * SPEED * delta
    animated_sprite.flip_h = direction == 1 # Corrected flipping logic
```

Slika 22. _process(delta) funkcija od letećeg neprijatelja

_process(delta) funkcija prikazana na slici 22 upravlja ponašanjem neprijatelja ovisno o njegovom stanju.

if not is_alive: return: Ako je neprijatelj mrtav, prestaje reagirati i kretati se.

Ako neprijatelj juri igrača (chasing_player je true), poziva se funkcija fly_towards_player() koja upravlja kretanjem prema igraču.

Ako se neprijatelj vraća na svoju početnu poziciju (returning_to_position je true), poziva se funkcija return_to_position() koja vraća neprijatelja na početnu točku.

Ako neprijatelj nije ni u jednom od ovih stanja, poziva se fly_cycle() koja upravlja njegovim patroliranjem.

5.4. Izrada boss scene

Kod izrade boss neprijatelja, važno je uzeti u obzir nekoliko ključnih aspekata koji će osigurati izazovno, dinamično i uzbudljivo iskustvo za igrača.

Ponašanje i napadi su najbitniji aspekti kod dizajna bilo kojeg neprijatelja. Potrebno je imati na umu koje napade i pokrete igrač može izvesti, u suprotnom postoji mogućnost da igrač neće moći pobijediti neprijatelja.

Napadi boss-a trebaju imati vizualne ili zvučne naznake prije nego što se izvedu. To omogućuje igraču da reagira na vrijeme, ali i stvara osjećaj napetosti.

Raznovrsnost napada čini borbu zanimljivom i nepredvidivom. Poželjno je da boss koristi mješavinu bliskih i dalekometnih napada.

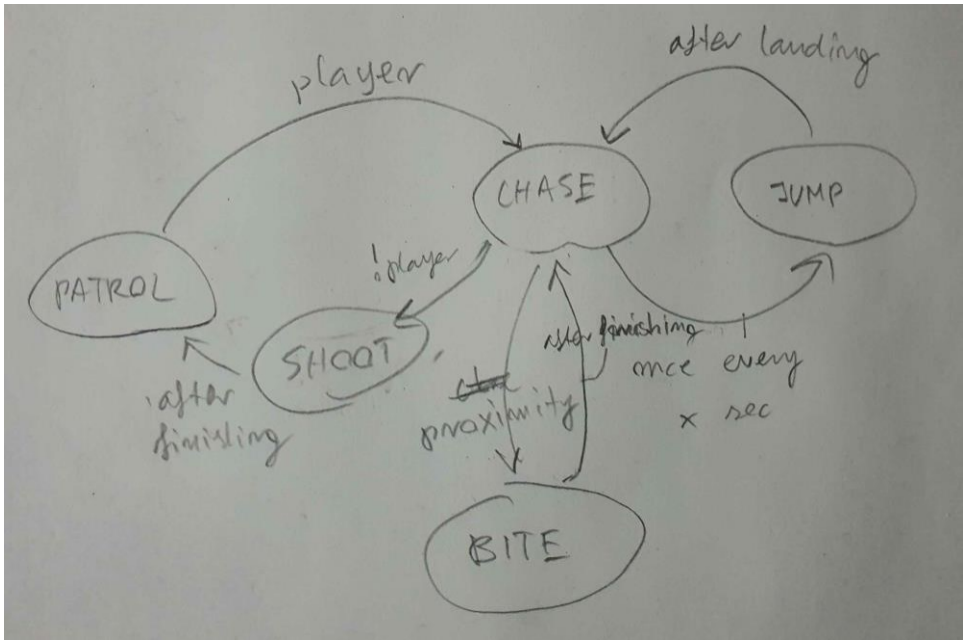
Slika 23 prikazuje dizajn boss-a. Dizajn je baziran na paucima i kukcima.

Slika 23. Prikaz boss-a

Vizualni izgled boss-a treba biti upečatljiv i prepoznatljiv kako bi odmah ostavio dojam na igrača. Boss bi trebao izgledati moćnije ili prijeteće u usporedbi s običnim neprijateljima, što može uključivati njegovu veličinu, animacije, boje ili specifične vizualne efekte.

Za praćenje različitih stanja u kojem može biti boss, odlučena je implementacija finite-state machine (hr. konačni automat). To je sustav koji omogućuje praćenje i kontrolu različitih stanja u kojima se objekt može nalaziti te definiranje pravila prijelaza iz jednog stanja u drugo. State machine je izuzetno koristan alat u razvoju igara jer omogućuje organizirano i modularno upravljanje složenim ponašanjem likova, objekata i sustava unutar igre.

Priložena slika 24 prikazuje skicu state machine za boss neprijatelja. Potrebno je detaljno razmisliti o različitim stanjima koje neprijatelj može poprimiti prije same implementacije. Na taj način može se odrediti odnos između stanja i kako izgledaju različite tranzicije.



Slika 24. Skica state mašine za boss neprijatelja

State machine se sastoji od niza stanja (states) u kojima se neki objekt može nalaziti u bilo kojem trenutku. Svako stanje ima svoja pravila i ponašanja. Prijelazi (eng. transitions) između stanja definiraju kada i pod kojim uvjetima objekt prelazi iz jednog stanja u drugo [8]. U sljedećoj slici 25 prikazana je deklaracija stanja u boss scripti.

```
37  # State machine variables
38  enum State { IDLE, PATROL, CHASE, JUMP, DEAD, SHOOT }
39  var current_state = State.IDLE
```

Slika 25. Deklariranje stanja u boss.gd

Funkcija **state_idle()** prikazana u slici 26 predstavlja stanje mirovanja pauka, gdje pauk ne radi ništa.

pass unutar funkcije znači da trenutno ne postoji specifična logika za ovo stanje, ali može se dodati u budućnosti. Ovo stanje bi moglo biti korisno kad pauk "spava" ili ne obraća pažnju na igrača.

```
163 ▾ func state_idle(delta):
164     pass
```

Slika 26. state_idle(delta) funkcija u boss.gd

state_patrol(delta) funkcija prikazana u slici 27 upravlja patroliranjem pauka. Pauk se kreće lijevo ili desno dok ne naiđe na zid.

Dva RayCast čvora, ray_cast_right i ray_cast_left, koriste se za detekciju sudara sa zidovima ili preprekama. Ako pauk udari u zid dok ide desno, on mijenja smjer i kreće se ulijevo (velocity.x = -SPEED), i obrnuto ako udari u zid s lijeve strane. Logika ovdje omogućuje pauku da se stalno kreće naprijed-nazad u okviru svoje patrole.

```
166 func state_patrol(delta):
167     # Switch the spider's directions if it hits a wall during patrol
168     if ray_cast_right.is_colliding():
169         velocity.x = -SPEED
170     elif ray_cast_left.is_colliding():
171         velocity.x = SPEED
```

Slika 27. state_patrol(delta) funkcija u boss.gd

state_chasing(delta) funkcija prikazana u slici 28 predstavlja stanje u kojem pauk trči za igračem.

Ako je igrač unutar dometa (prepoznat kao player), pauk izračunava udaljenost do igrača koristeći player.global_position - global_position. Ovaj vektor se zatim normalizira (normalized()) kako bi se dobio pravac kretanja prema igraču.

Pauk se zatim kreće prema igraču koristeći taj pravac pomnožen s brzinom (SPEED).

U dodatnom dijelu logike, pauk može i skočiti prema igraču. Ako je prošlo dovoljno vremena od zadnjeg skoka (time_last_jumped) i igrač je dovoljno udaljen (distance_to_player.length() > JUMP_TRIGGER_DISTANCE), pauk prelazi u stanje skakanja koristeći set_state(State.JUMP).

JUMP_COOLDOWN i rand_jump_cooldown_offset osiguravaju da pauk ne skače prečesto i da postoji mala varijabilnost u vremenu između skokova.

```
173 func state_chasing(delta):
174     if player:
175         # Chase player if spotted
176         var distance_to_player = player.global_position - global_position
177         var direction_to_player = distance_to_player.normalized()
178         velocity.x = direction_to_player.x * SPEED
179 
180         # Jump at player if enough time has passed since last jump
181         var time_since_last_jump = Time.get_ticks_msec() - time_last_jumped
182         if distance_to_player.length() > JUMP_TRIGGER_DISTANCE and time_since_last_jump > JUMP_COOLDOWN + rand_jump_cooldown_offset:
183             set_state(State.JUMP)
```

Slika 28. state_chasing(delta) funkcija u boss.gd

state_jumping(delta) funkcija prikazana u slici 29 upravlja paukovim skakanjem. Kad pauk skoči prema igraču, nalazi se u ovom stanju dok ne sleti.

Kad pauk dotakne tlo (provjerava se sa on_ground) i više nije u stanju skakanja (not jumping), te kada vertikalna brzina (velocity.y) postane blizu nule (>= -1), pauk se vraća u stanje jurcanja (State.CHASE).

Ova logika osigurava da pauk skoči, a zatim se vrati u potjeru čim dotakne tlo.

```
186 func state_jumping(delta):
187     # Change back to CHASE state once spider lands
188     if on_ground and not jumping and velocity.y >= -1:
189         set_state(State.CHASE)
```

Slika 29. state_jumping(delta) u boss.gd

state_shooting(delta) funkcija prikazana u slici 30 upravlja pucanjem pauka. Kad pauk puca, koristi se animacija pucanja, a pauk ostaje u tom stanju dok se animacija ne završi.

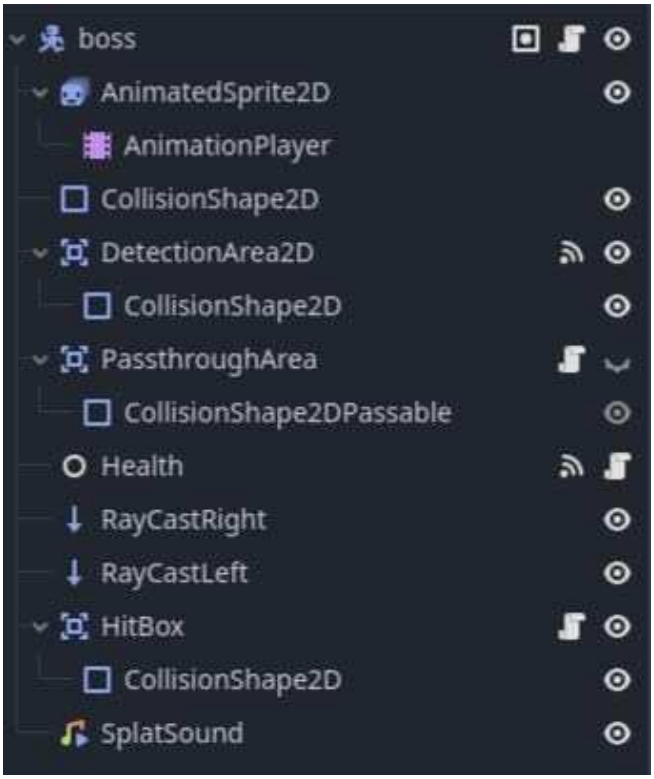
Linija await animated_sprite.animation_finished koristi await za čekanje dok animacija pucanja ne završi. Kada animacija završi, ovisno o prisutnosti igrača, pauk se vraća u stanje jurcanja (State.CHASE) ili ponovno počinje patrolirati (State.PATROL).

Ova logika omogućuje pauku da reagira na igrača i promijeni ponašanje nakon završetka napada.

```
191 func state_shooting(delta):
192     await animated_sprite.animation_finished  # Wait for the shooting animation to finish
193     if player:
194         set_state(State.CHASE)
195     else:
196         set_state(State.PATROL)
```

Slika 30. state_shooting(delta) funkcija u boss.gd

Priložena slika 31 opisuje hijerarhiju čvorova u boss sceni.



Slika 31. Hijerarhija čvorova u boss sceni

5.5. Izrada level scene

Za izradu okoline u video igri, potrebno je obratiti pažnju na nekoliko ključnih aspekata, uključujući planiranje, dizajn, tehničku implementaciju i optimizaciju. Razvoj okoline ne obuhvaća samo vizualni izgled, već i funkcionalnost, interaktivnost i prilagodbu razini performansi igre.

Prvi korak u izradi okoline je definiranje teme i stila nivoa. Dizajner treba osmisliti kako će okolina izgledati i koje će emocije prenijeti igraču. Primjeri mogu biti urbani pejzaži, šume, pustinje, svemirske postaje i slično. Ovdje se definira raspored (eng. layout) nivoa, uključujući strukture, prepreke i putanje koje će igrač slijediti.

U projektu bio je cilj dizajnirati šarmantnu okolinu koja privlači igrače da se upuste u pustolovinu. Kako igrač napreduje, okolina postaje sve opasnija i nepoznatija. Igrač ima osjećaj kao da je ušao u divljinu i treba naći put natrag.

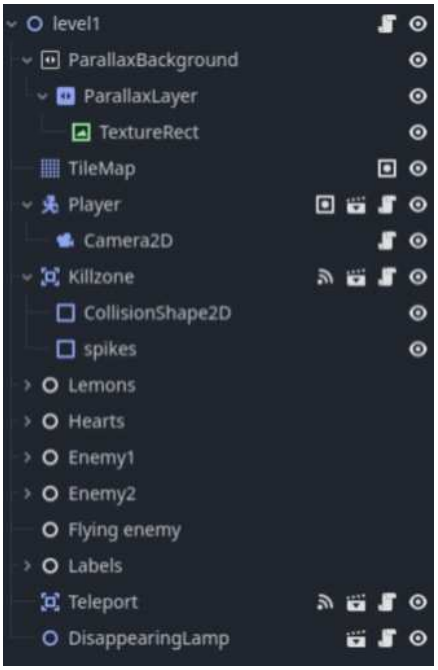
Priloženi tileset, vidljiv na slici 32, inspiriran je retro igrama iz vremena kada su igre bile jednostavnije.

Slika 32. Tileset igre

Za izradu 3D ili 2D modela i tekstura potrebni su alati poput Blender-a (za 3D) ili alata poput Photoshop-a ili Aseprite-a (za 2D spriteove i teksture). Potrebno je stvoriti modele terena, zgrada, objekata, biljaka i drugih elemenata okoline. U 2D okruženju, to uključuje sprite-ove pozadine, platformi i prepreka.

Nakon što je okolina izgrađena, potrebno je izvršiti temeljito testiranje kako bi se osiguralo da svi elementi funkcioniraju ispravno.

Priložena slika 33 opisuje hijerarhiju čvorova u level sceni.



Slika 33. Hijerarhija čvorova u level sceni

level1 je korijenski čvor scene, koji sadrži sve ostale elemente nivoa.

ParallaxBackground čvor koristi se za stvaranje paralakse, što je vizualni efekt gdje se pozadina pomiče sporije od prednjih elemenata, stvarajući dojam dubine. **ParallaxLayer** služi za dodavanje slojeva paralakse. Svi elementi unutar ovog sloja pomiču se u skladu s pomakom kamere. **TextureRect** čvor prikazuje teksturu, koja predstavlja pozadinsku sliku za ovaj sloj paralakse.

TileMap čvor omogućava korisniku postavljanje pločica (eng. tiles) kako bi se kreirao teren nivoa. Umjesto ručnog postavljanja svakog objekta, TileMap koristi unaprijed definirane pločice koje se mogu ponavljati, čineći proces izrade nivoa učinkovitijim.

Player čvor predstavlja igrača u igri. Ovaj čvor sadrži logiku i kontrole koje omogućuju igraču da se kreće i komunicira s okolinom. Camera2D čvor slijedi kretanje igrača po nivou, budući da je dijete od Player čvora.

Killzone je čvor tipa Area2D koji, kada ga igrač dotakne, uzrokuje smrt. CollisionShape2D je oblika WorldBoundaryShape2D koje definira kolizijsko područje koje prepoznaje dodir igrača. Ono se koristi kada igrač padne s nivoa. Spikes čvor je tipa CollisionShape2D, te ima pravokutan oblik koji detektira igrača.

Lemons i **Hearts** čvorovi predstavljaju skupljive predmete (npr. limune i srca). Igrač može prikupljati te predmete tijekom igre kako bi dobio municiju ili povećao broj života.

5.5.1. Dizajn razina

Prije samog dizajna razina bitno je pomno razmisliti koja je svrha razine i koliko će vještine zahtijevati od igrača. Uobičajeno je započeti igru lakšim razinama i postupno uvesti teže elemente poput letećih neprijatelja i zahtjevnih skokova. Također, razine su dio priče igre. Okoliš treba pomoći igraču da se uživi u svoju ulogu junaka.

Igra ima sveukupno 7 razina, uključujući tutorijal.

5.5.1.1. Level 0

Uvodni level u igri služi kao uvod u priču igre. Igrač upozna obitelj glavne junakinje i istražuje obiteljsku kuću protagonistice kao što je prikazano na slici 34.

Slika 34. Prikaz obiteljske kuće na uvodnoj razini igre

5.5.1.2. Level 1

Prva razina je tutorijal. Igrač uči osnovne kontrole: skok, dvostruki skok, dash, izbjegavanje neprijatelja, dash, kako ubiti neprijatelja i kako preći na sljedeću razinu. Dizajn razine može se vidjeti na slici 35.

Slika 35. Prikaz prve razine

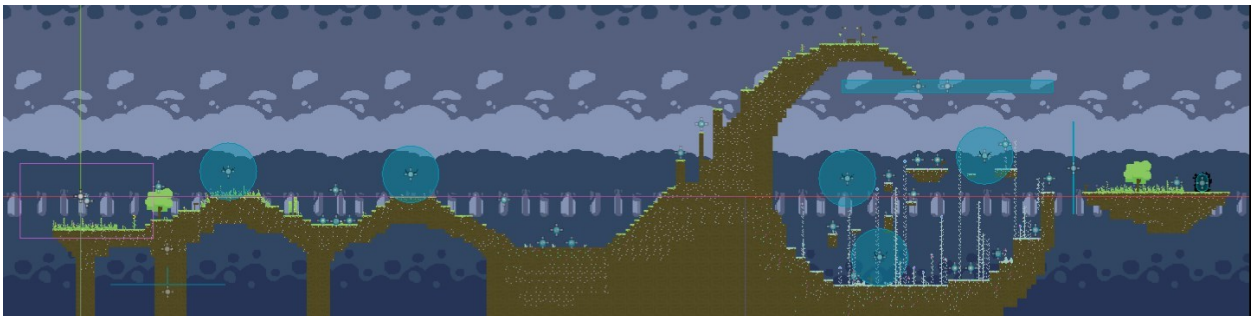
5.5.1.3. Level 2

Druga razina potpuno uvodi igrača u divljinu, kao što je prikazano na slici 36. Igrač više nema pomagala i prvi put će vidjeti letećeg neprijatelja. Dizajn okoline simbolizira sam ulazak u divljinu. Razina započne s urednim drvećem i vrtom, a završi u divljoj šumi.

Slika 36. Prikaz druge razine

5.5.1.4. Level 3

Treća razina nastavlja se na treću, kako je prikazano na slici 37. Kako igrač napreduje, mora izbjegavati ili pobijediti sve veći broj neprijatelja. Okruženje postaje sve opasnije.



Slika 37. Prikaz treće razine

Igrač treba savladati svoje strahove u trenutku koji je prikazan na slici 38 i skočiti s platforme u nepoznato.

Slika 38. Prikaz detalja iz treće razine

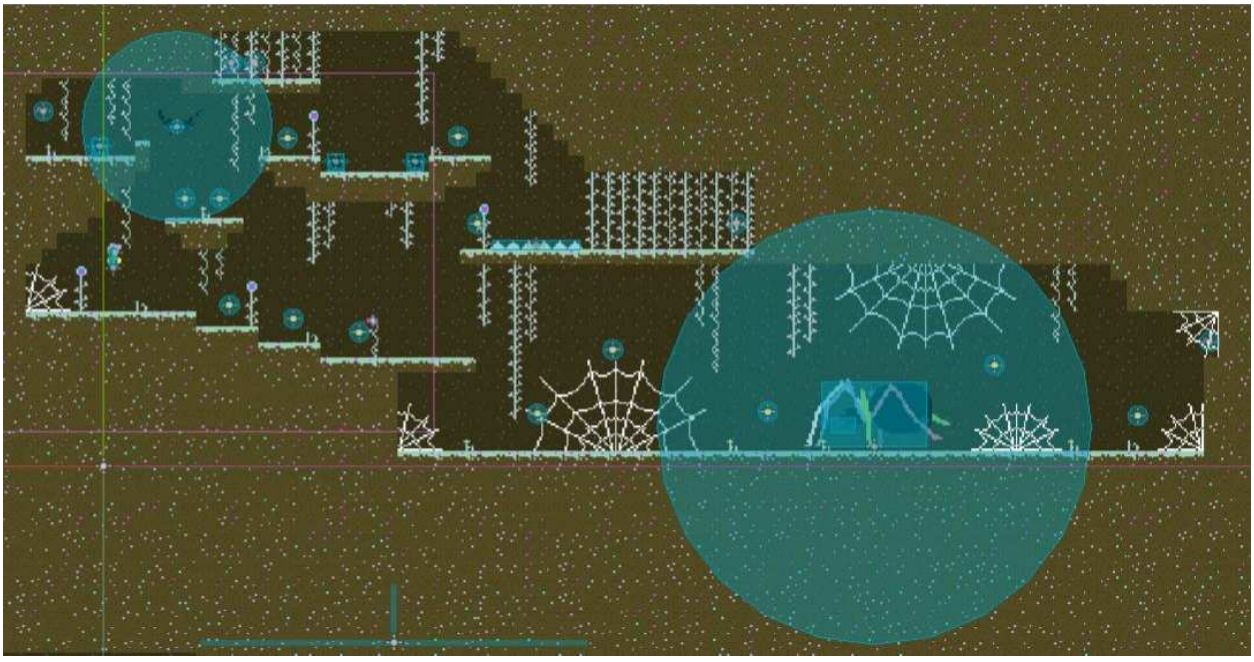
5.5.1.5. Level 4

Četvrta razina sastoji se od dva dijela kako je prikazano na slici 39. Igrač započinje razinu u divljoj šumi, a završi u zastrašujućoj špilji koja je puna neprijatelja. Ova razina zahtijeva najvišu razinu koncentracije od igrača.

Slika 39. Prikaz četvrte razine

5.5.1.6. Level 5

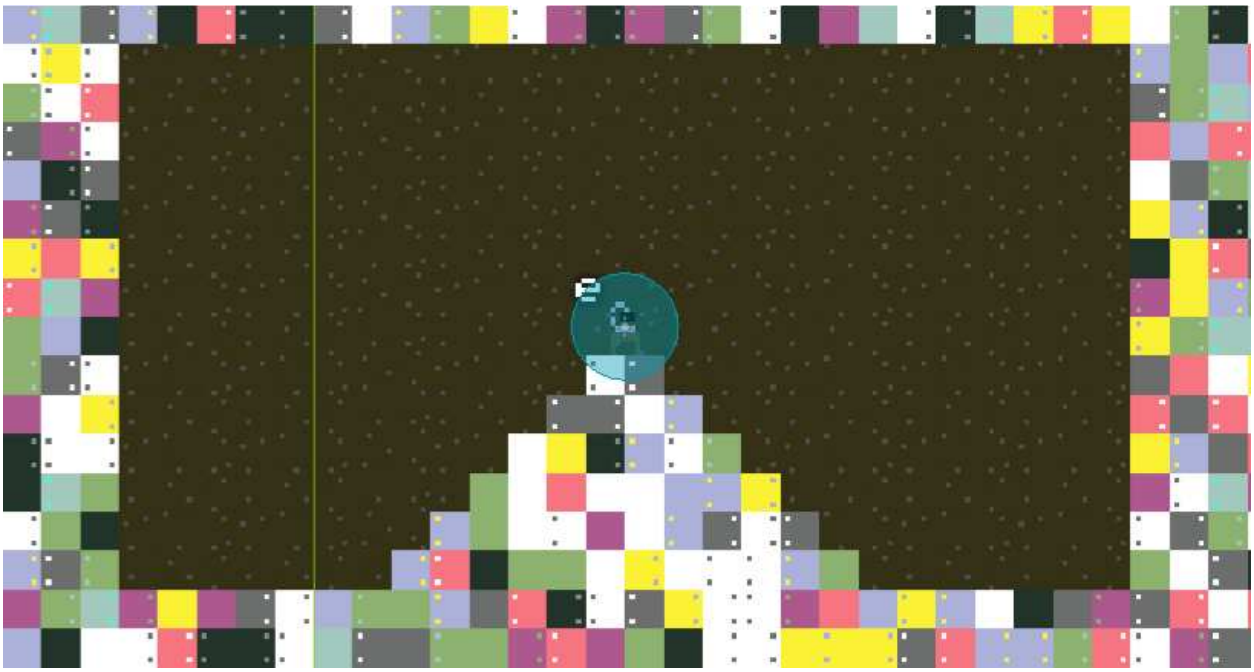
Peta razina nastavlja špilju prethodne razine kao što je prikazano u slici 40. Razina je manja od prethodnih zbog velike bitke koja se desi između igrača i boss-a.



Slika 40. Prikaz pete razine

5.5.1.7. Level 6

U zadnjoj razini igrač saznaje tajnu svijeta i završava igru. Okoliš je kaotičan kao što se vidi na slici 41 i simbolizira unutarnje stanje glavne junakinje.



Slika 41. Prikaz zadnje razine

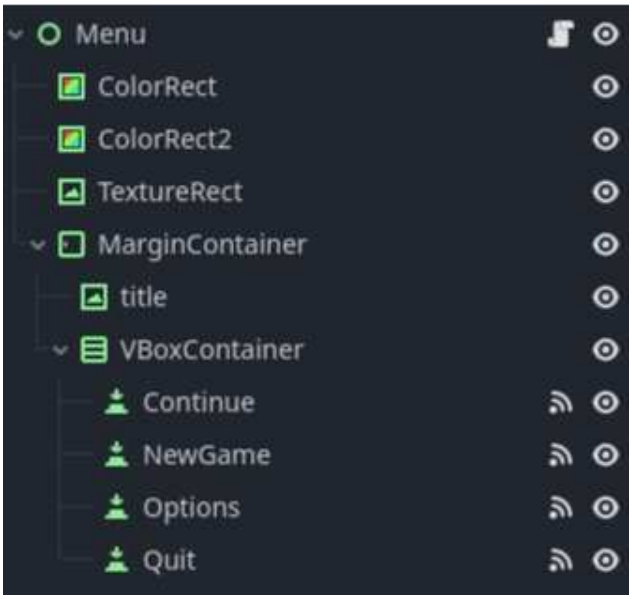
5.6. Glavni izbornik

Dizajn glavnog izbornika u videoigrama ključan je element jer ostavlja prvi dojam na igrača. Dobro dizajniran glavni izbornik može poboljšati korisničko iskustvo i učiniti igru intuitivnijom za korištenje, dok loš dizajn može izazvati frustraciju kod igrača.

Jednostavan i pregledan dizajn je bitan. Korisnik bi trebao odmah razumjeti kako se kretati kroz izbornik [7]. Prevelik broj opcija ili složeni izbornici mogu zbuniti igrača. Stavke izbornika trebale bi uključivati:

- **Play** (Igraj): Gumb za pokretanje igre.
- **Options** (Opcije): Postavke igre, poput grafike, zvuka i kontrola.
- **Quit** (Izlaz): Opcija za izlazak iz igre.

Sljedeća slika 42 prikazuje hijerarhiju čvorova u menu sceni.



Slika 42. Hijerarhija čvorova u menu sceni

Korijenski čvor (eng. root node) glavnog izbornika je tipa Control i imena Menu. Na korijenski čvor je povezana skripta Menu.gd koja upravlja ponašanjem ove scene.

Menu čvor je roditelj svim ostalim čvorovima u sceni. Control čvor u Godot engine-u je osnovni čvor koji se koristi za izradu i upravljanje korisničkim sučeljem (eng. user interface). Svi UI elementi u Godotu, poput gumba, tekstualnih polja, panela i drugih interaktivnih komponenti, nasljeđuju Control čvor.

ColorRect, ColorRect2 i **TextureRect** su čvorovi povezani uz prikazivanje pozadine. TextureRect Sadrži teksturu slike koje je bila dizajnirana u Aseprite-u. Korisna opcija ovog čvora je ponavljanje teksture (eng. tiling). Ova tehnika omogućuje da se manja tekstura

prikazana na slici 43 ponavlja po cijeloj površini, bez rastezanja, čime se stvara neprekidna pozadina.

U Inspector prozoru, korisnik treba pronaći kategoriju Rect i unutar nje postaviti opciju Stretch Mode na Tile. Time se omogućuje ponavljanje teksture po cijelom prostoru unutar TextureRect čvora.

Slika 43. Prikaz plave pozadine

MarginContainer čvor služi za postavljanje i upravljanje marginama (rubovima) oko svog sadržaja. Služi kako bi elementi imali dovoljnu udaljenost od granica ili drugih elemenata u korisničkom sučelju. Čvorovi koji se nalaze unutar MarginContainer-a su raspoređeni tako da poštuju zadane margine, bez potrebe za ručnim pozicioniranjem.

VBoxContainer je čvor u koji automatski raspoređuje svoje dječje čvorove okomito, jedan ispod drugog. Namijenjen je olakšavanju postavljanja elemenata korisničkog sučelja (UI) koji trebaju biti organizirani u stupcu. Ako bilo koji od čvorova unutar VBoxContainer-a promijeni svoju veličinu, VBoxContainer automatski prilagođava raspored svih ostalih elemenata.

Continue, NewGame, Options, Quit predstavljaju pojedinačne stavke izbornika, koje su tipke (eng. button). Budući da su smješteni unutar VBoxContainer-a, automatski su raspoređeni vertikalno jedan ispod drugog.

Prikazani isječak koda u slici 44 upravlja radnjama koje se izvode kada se pritisnu različiti gumbi u izborniku igre, poput **Nastavka igre**, **Novog početka**, **Opcija** i **Izlaza**. Kod koristi nekoliko metoda za promjenu scena i upravljanje podacima o igri. On nasljeđuje čvor Menu.

```gdscript
extends Control

func _on_continue_pressed():
	GameManager.load_data()
	var scene_number = int(GameManager.current_level)
	var scene_path = "res://scenes/level" + str(scene_number) + ".tscn"
	GameManager.show_fade_label("Loading game...", global_position)
	get_tree().change_scene_to_file(scene_path)
	GameManager.show_ui()

func _on_options_pressed():
	get_tree().change_scene_to_file("res://scenes/options_menu.tscn")
	GameManager.last_menu = "main menu"

func _on_quit_pressed():
	get_tree().quit()

func _on_new_game_pressed():
	GameManager.show_fade_label("Starting new game", global_position)
	get_tree().change_scene_to_file("res://scenes/level0.tscn")
	GameManager.delete_data()
	GameManager.load_data()
	GameManager.show_ui()

```

Slika 44. Isječak koda od menu scene

_on_continue_pressed() funkcija se pokreće kada korisnik pritisne gumb Continue (hrv. nastavi). Funkcija koristi GameManager autoload za učitavanje prethodnih podataka o igri pomoću metode load_data(). Zatim dohvaća trenutni nivo igre (current_level) iz GameManager klase, pretvara ga u cijeli broj (int()) i koristi ga za generiranje putanje do odgovarajuće scene. Primjer: ako je nivo 1, generira se putanja poput "res://scenes/level1.tscn".

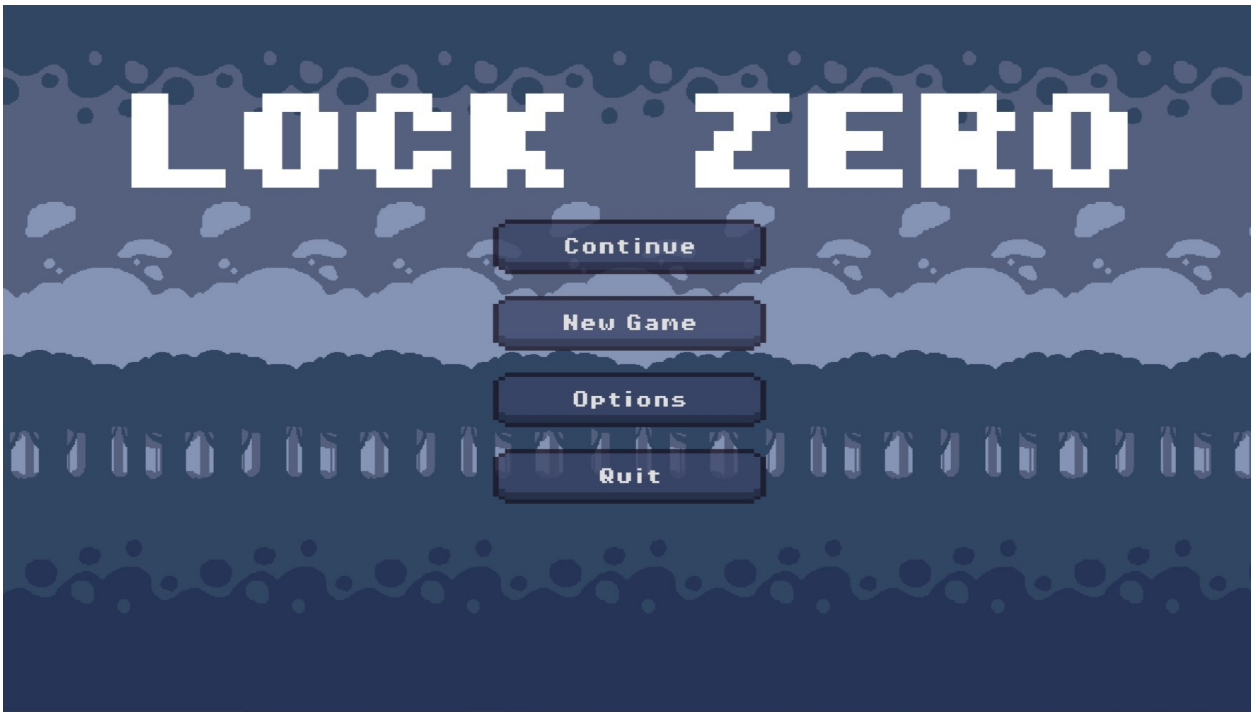
Prije promjene scene, prikazuje se poruka "Loading game...", koristeći show_fade_label(). Nakon toga, koristi get_tree().change_scene_to_file() kako bi promijenio scenu na tu određenu razinu. Na kraju, poziva se GameManager.show_ui() kako bi se prikazalo korisničko sučelje igre koje prikazuje koliko municije i srca ima igrač.

_on_options_pressed() funkcija se aktivira kada korisnik pritisne gumb Options (hrv. opcije). Prebaciti će trenutnu scenu na izbornik s opcijama pomoću get_tree().change_scene_to_file("res://scenes/options_menu.tscn").

_on_quit_pressed() funkcija se poziva kada korisnik pritisne gumb Quit (hrv. izlaz). Funkcija jednostavno poziva get_tree().quit(), što zatvara igru.

_on_new_game_pressed() funkcija se poziva kada korisnik pritisne gumb New Game (hrv. nova igra). Prikazuje poruku "Starting new game" pomoću GameManager.show_fade_label(). Zatim mijenja scenu na prvu razinu igre (level0) pomoću get_tree().change_scene_to_file("res://scenes/level0.tscn"). Nakon promjene scene, poziva GameManager.delete_data(), što briše prethodne podatke o spremljenoj igri, ako postoje. Ponovno se poziva GameManager.load_data() kako bi se postavili podaci za novu igru.Na kraju, prikazuje se korisničko sučelje igre pomoću GameManager.show_ui().

Priložena slika 37 prikazuje kako izgleda glavni izbornik.



Slika 45. Prikaz glavnog izbornika

6. Priča

Prilikom izrade priče za 2D platformer, nekoliko je ključnih elemenata na koje treba obratiti pažnju.

U 2D platformerima radnja obično nije previše složena, ali mora biti dovoljno zanimljiva da motivira igrača. Priča bi trebala pružiti jasne ciljeve, poput spašavanja lika, pobjede nad zlikovcem ili prelaska kroz različite razine. Glavni lik mora imati jasan cilj koji ga pokreće kroz igru. Uobičajeni motivi su osvajanje nečega, borba protiv negativca ili postizanje unutarnjeg ili vanjskog mira.

U ovom projektu glavni je cilj protagonistice pronaći svoju obitelj i otkriti misterij koji polako uništava njezin svijet. Ovi ciljevi daju igraču poticaj da završe igru.

6.1. Likovi

6.1.1. Blu

Glavna junakinja koju igrač kontrolira zove se Blu, te je prikazana na slici 46. Ona ima posebne moći i slobodu kretanja koju nemaju ostali likovi u priči. Njezin cilj je otkrivanje misterija koji joj je svijet postavio.

Slika 46. Prikaz glavne junakinje

6.1.2. Daisy

Daisy prikazana na slici 47 je starija sestra od Blu. Ona se voli brinuti o svom vrtu. Blu se osobito pomno brine o njoj.

Slika 47. Prikaz sestre glavne junakinje

6.1.3. Roditelji

Roditelji od Blu i Daisy se ponekad čine kao jedina konstanta u njihovom životu. Prikazani su na slikama 48 i 49.

Slika 48. Prikaz oca glavne junakinje

Slika 49. Prikaz majke glavne junakinje

6.1.4. Kuća

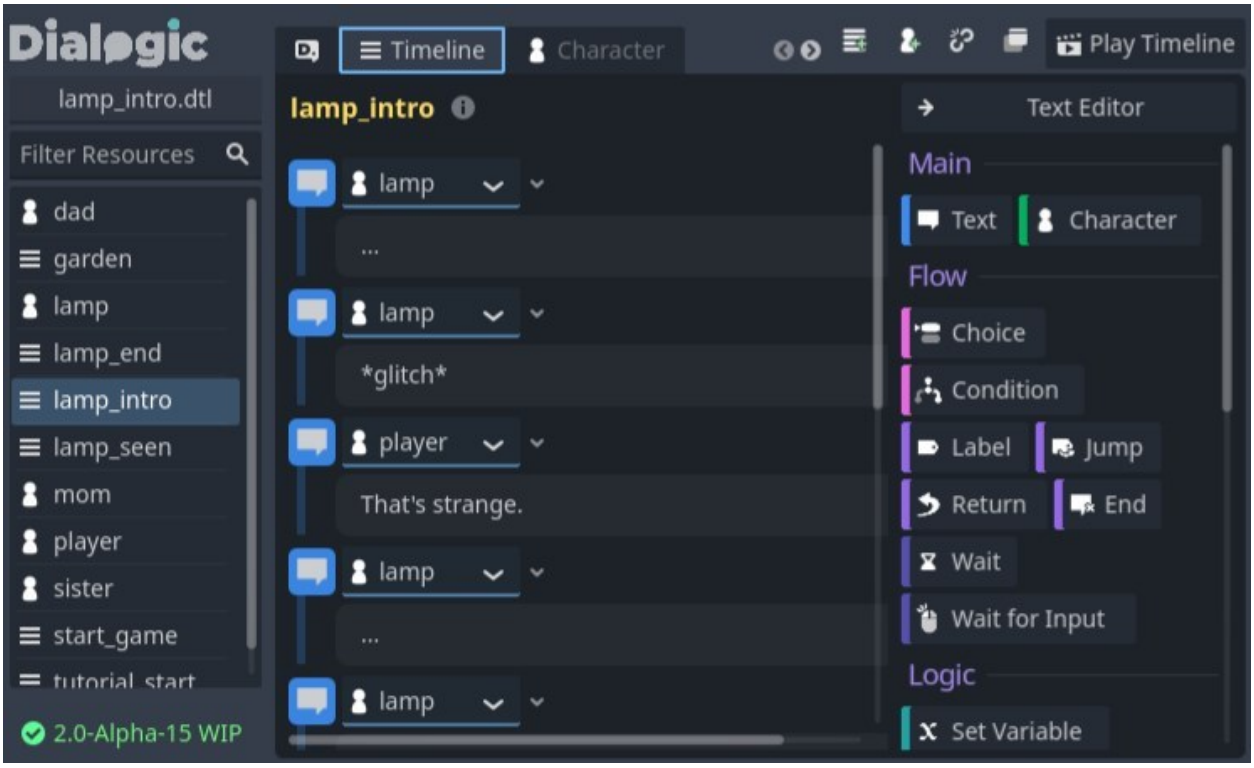
Obiteljska kuća od protagonistice i njezine obitelji. Kuća je prikazana na slici 50.

Slika 50. Prikaz kuće od protagonistice

6.2. Dijalog

Dialogic je plugin za Godot engine koji omogućuje lako upravljanje dijalozima unutar igara bez potrebe za pisanjem kompleksnog koda. Ovaj alat olakšava razvoj narativnih igara ili igara koje sadrže puno dijaloga između likova.

On omogućuje korisniku kreiranje i uređivanje dijaloga putem vizualnog sučelja koji je prikazan na slici 51, bez potrebe za pisanjem skripti. Korisnik može dodavati tekstove, opcije dijaloga, animacije likova i druge elemente kroz intuitivno sučelje unutar Godot editora.



Slika 51. Prikaz korisničkog sučelja za Dialogic plugin

Dialogic omogućuje definiranje likova unutar dijaloškog sistema. Svaki lik može imati svoje ime, portret (sliku) i boju teksta, što olakšava razlikovanje likova u dijalozima. Dizajn dijaloškog prozora prikazan je na slici 52.

Portreti likova se mogu mijenjati tijekom dijaloga, što omogućuje prikaz različitih emocija ili izraza lica likova dok razgovaraju.

Slika 52. Prikaz dijaloga u igri

Dialogic podržava grananje dijaloga, što znači da igrači mogu birati između različitih odgovora i opcija u razgovoru. Ovaj sustav omogućuje stvaranje kompleksnih narativnih struktura gdje izbori igrača mogu utjecati na tijek priče.

Korisnik može postaviti uvjete za određene grane dijaloga, što omogućuje različite reakcije likova ovisno o prethodnim akcijama igrača.

7. Zaključak

Proces izrade videoigre u Godot razvojnome okruženju bio je vrlo ugodan i jednostavan. Ovaj engine je savršena kombinacija jednostavnosti i kompleksnosti, te odgovara potrebi većini programera koji se bave razvijanjem igrica, pogotovo 2D igrica. Nove verzije engine-a stalno izlaze, što znači da će biti još više unaprijeđen. Postoji vrlo aktivna online zajednica koja stalno uči i unapređuje Godot. Uspoređujući ovo iskustvo s Unity iskustvom, može se zaključiti da je Godot znatno brži i jednostavniji.

Kreacija platformera bila je zahtjevna. Kontrole igrača su vrlo bitne i mogu učiniti ili uništiti igru. Potrebno je stalno testirati i iterirati. Dizajn razina vrlo je težak zadatak, budući da je potrebno balansirati težinu sa zabavom igrača.

Aseprite također nije razočarao, alat je idealan za kreiranje resursa za 2D igrice. Proces animiranja i osmišljavanja bio je zabavan. Uvođenje resursa iz Aseprite-a u Godot bilo je vrlo jednostavno.

Platformeri su odlično sredstvo za pripovijedanje i kreiranje narativa. Vrlo je jednostavno kreirati priču koja se mijenja ovisno o igračevim odabirima pomoću Dialogic plugin-a.

Analizirajući ovaj projekt, mogu se zaključiti određene prednosti i mane. Prednost projekta jest njegova jednostavnost i moguća skalabilnost. Vrlo je lako unaprijediti nešto jednostavno. Projekt ima puno potencijala za ekspanziju priče i dodavanja više razina i neprijatelja. Lako je pratiti priču.

Prednost može ujedno biti i mana. Određenim igračima je potrebna veća kompleksnost i moderniji vizuali da bi uživali u igri. Igra je relativno kratka i igrač nema veliki poticaj da ju opet odigra. Moguće rješenje moglo bi biti dodavanje više različitih završetka i dodatak štoperice koja bi mogla mjeriti koliko brzo je igrač završio igru.

Buduća poboljšanja bila bi ovisna o rezultatima dugoročnog testiranja razina. Potrebno je balansirati i unaprijediti razine da se osigura zadovoljstvo igrača. Veliko poboljšanje bilo bi dodatak kompleksnijih dijaloga, više razina i više neprijatelja.

Ponašanje boss neprijatelja može se unaprijediti s implementacijom pametnije umjetne inteligencije. Također, nedostaje UI za prikaz količine života kojim boss barata u bilo kojem trenutku.

Bilo bi poželjno obaviti ekspanziju scene s opcijama s odabirom rezolucije i opcijom promjene poveznice za tipke za kretanje igrača.

Kreiranje ovog projekta bilo je vrlo vrijedno iskustvo. Čak i najjednostavnije mehanike teško implementirati bez prethodnog iskustva.

Literatura

[1] Tech, I. (n.d.). What is a Platform Game? | 10 Design Types & Video Game Examples. https://www.idtech.com/blog/10-types-of-platforms-in-platform-video-games

[2] Thorson, M. (2017, July 3). 'Celeste' releasing January 2018 - Maddy Thorson - Medium. Medium. https://maddythorson.medium.com/celeste-releasing-january-2018-23fbadd4172a

[3] New game Network. (2015, September 27). New Game Network. https://www.newgamenetwork.com/media/17788/undertale/

[4] *Introduction to Godot*. (n.d.). Godot Engine Documentation. https://docs.godotengine.org/en/stable/getting_started/introduction/introduction_to_godot.html#

[5] Tips and Tricks for good platforming games (with example) - Things Made By Dave. (n.d.). http://www.davetech.co.uk/gamedevplatformer

[6] Thorson, M. (2023, February 16). Celeste & Forgiveness - Maddy Thorson - Medium. *Medium*. https://maddythorson.medium.com/celeste-forgiveness-31e4a40399f1

[7] Jonkers, D. (2011b, July 4). 11 Tips for making a fun platformer. Dev.Mag. http://devmag.org.za/2011/01/18/11-tips-for-making-a-fun-platformer/

[8] What is a state machine? (n.d.). https://www.itemis.com/en/products/itemis-create/documentation/user-guide/overview_what_are_state_machines#:~:text=A%20state%20machine%20is%20a,state%20transitions%20and%20produces%20outputs.

Sažetak

Tema ovog završnog rada je razvoj platformer igrice u razvojnom okruženju Godot što obuhvaća opis procesa izrade video igrice, implementacije elemenata videoigre i prikaz videoigre. Pokazano je kako odabrati dobru Godot verziju, kako napraviti novi projekt, struktura najbitnijih scena. Također su objašnjene skripte u projektu i kako one međusobno komuniciraju. Čitatelji ovog završnog rada bi trebali steći potrebno znanje za razvoj slične platformer igrice.

Ključne riječi: Godot, GDScript, razvoj videoigre, programiranje, videoigra, Aseprite

Abstract

The topic of this undergraduate thesis is the development of a platformer game in the Godot engine, which includes a description of the process of creating a video game, implementing game elements, and showcasing the game. It demonstrates how to choose the appropriate Godot version, how to create a new project, and the structure of the most important scenes. Additionally, the project's scripts and how they interact with each other are explained. Readers of this undergraduate thesis should gain the necessary knowledge to develop a similar platformer game.

Keywords: Godot, GDScript, video game development, programming, video game, Aseprite

Popis slika