Razvoj računalne igre Void Born u Unity okruženju

Pakter, Mateo

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

Mateo Pakter

RAZVOJ RAČUNALNE IGRE VOID BORN U UNITY OKRUŽENJU

Diplomski rad

Pula, rujan 2024. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

Mateo Pakter

RAZVOJ RAČUNALNE IGRE VOID BORN U UNITY OKRUŽENJU

Diplomski rad

JMBAG: 0303082649
Studijski smjer: Informatika
Kolegij: 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. godine

SAŽETAK

Cilj ovog diplomskog rada je detaljno prikazati proces izrade 2D računalne igre korištenjem Unity alata. U uvodnom dijelu objasnit će se kako postaviti novi projekt u Unity okruženju, uz osnovni pregled funkcionalnosti samog programa. Nadalje, prikazat će se što su spriteovi te kako ih iskoristiti za izradu 2D igre. Kroz praktične primjere, demonstrirat će se kreiranje glavnog lika, te način na koji se upravlja scenama unutar Unity. Detaljno će se objasniti vještine glavnog lika, uz pregled njihovih funkcionalnosti i implementacije u igri. Sljedeći dio rada fokusirat će se na kreiranje neprijatelja, uključujući njihove animacije i programsko ponašanja. Također će biti objašnjena priča igre i način na koji je ispričana kroz različite elemente unutar igre. Poseban naglasak bit će stavljen na izradu korisničkog sučelja, uključujući kanvas, gdje će se demonstrirati kako ga prilagoditi i primijeniti unutar igre. Također, prikazat će se korištenje audio miksera i njegova primjena kroz cijeli projekt, osiguravajući kontrolu nad zvukom i efektima.

Ključne riječi: Unity, 2D računalna igra, sprite, glavni lik, priča, kanvas, audio

ABSTRACT

The aim of this master's thesis is to provide a detailed overview of the process of creating a 2D computer game using Unity. The introductory section will explain how to set up a new project in the Unity environment, along with a basic overview of the program's functionalities. Furthermore, the concept of sprites will be presented, as well as how they can be utilized to create a 2D game. Through practical examples, the creation of the main character will be demonstrated, along with how to manage scenes within Unity. The skills of the main character will be explained in detail, with an overview of their functionality and implementation in the game. The next section of the thesis will focus on the creation of enemies, including their animations and programmed behavior. The game’s story will also be explained, as well as how it is told through various elements within the game. Special emphasis will be placed on creating the user interface, including the canvas, where it will be demonstrated how to customize and apply it within the game. Additionally, the use of the audio mixer will be presented, along with its application throughout the project, ensuring control over sound and effects.

Key terms: Unity, 2D computer game, sprite, main character, story, canvas, audio

Sadržaj

1. Uvod

Video igre su posljednjih godina postale sve popularniji oblik zabave. S napretkom tehnologije, igranje je evoluiralo od jednostavne pikselizirane grafike do zapanjujućih, sveobuhvatnih svjetova koji nude interaktivna iskustva pripovijedanja. Kombinacija zabave i interaktivnog pripovijedanja očarala je milijune igrača diljem svijeta, čineći video igre dominantnom snagom u industriji zabave. Videoigre su prešle dug put od vremena Ponga i Tetrisa. Uvođenjem kućnih konzola kao što je Atari 2600 u kasnim 70-ima i ranim 80-ima, videoigre su počele dobivati mainstream priznanje. Ove rane igre bile su relativno jednostavne, sastoje se od osnovne mehanike igranja i ograničenih grafičkih mogućnosti. Međutim, postavili su temelje za ono što će postati industrija vrijedna više milijardi dolara (Anderson, 2023).

Video igre su brzo evoluirale iz manje popularnog hobija u jedno od najvećih tržišta u industriji zabave. Trenutno tržište videoigara vrijedi 282 milijarde dolara, a svake godine bilježi značajan rast. Predviđa se da će do 2027. godine vrijednost industrije premašiti 363 milijarde dolara. Globalno, postoji oko 3,09 milijardi aktivnih igrača videoigara, a taj broj bi trebao porasti na 3,32 milijarde do kraja 2024. godine. Sjedinjene Američke Države imaju više od 3000 profesionalnih esport igrača, dok je Azija, najveća gaming regija, dom gotovo 1,5 milijardi igrača. Gaming zajednica je raznolika; 53% muškaraca se identificira kao igrači, a 52% svih igrača pretplaćeno je na barem jednu gaming uslugu. Industrija videoigara bilježi godišnji porast broja igrača od oko 5%, što odražava njezinu kontinuiranu ekspanziju i utjecaj (Howarth, 2024).

Interes za videoigre iznimno je velik, no njihova izrada složen je i zahtjevan proces koji može trajati mjesecima, pa čak i godinama. Zbog veličine i raznolikosti zadataka, razvoj igre obično zahtijeva rad cijelog tima stručnjaka unutar studija. Neke od ključnih uloga za uspješnu realizaciju igre uključuju dizajnera, programera, audio dizajnera i animatora. Svaka od ovih uloga može se dodatno podijeliti na specijalizirane podskupine. Primjerice, programeri se mogu usmjeriti na različita područja kao što su mrežno programiranje, gameplay programiranje, razvoj umjetne inteligencije, arhitektura sustava ili razvoj game enginea. Dizajneri mogu biti specijalizirani za različite aspekte igre, poput dizajna razina, gdje se fokusiraju na kreiranje zanimljivih i izazovnih okruženja, ili dizajna korisničkog sučelja (UI/UX), gdje se bave stvaranjem intuitivnih i vizualno privlačnih interakcija za igrače. Također, dizajneri sustava rade na izradi složenih mehanika igre, balansiranju pravila i

osiguravanju da različiti dijelovi igre rade zajedno u harmoniji. Audio dizajneri mogu se specijalizirati za komponiranje glazbe, stvaranje zvučnih efekata ili dizajn zvučnih pejzaža koji pridonose atmosferi igre. Animatori se također dijele na više specijaliziranih uloga, kao što su 3D animatori, koji stvaraju pokrete likova i objekata u igri, ili animatori za cinematske scene, koji rade na kreiranju filmskih sekvenci unutar igre. Svaka od ovih uloga doprinosi stvaranju cjelovitog i koherentnog iskustva, a suradnja između specijaliziranih stručnjaka ključna je za uspješan razvoj videoigre (Club, 2022).

Kako bi se značajno smanjila količina programerskog rada i ubrzao proces razvoja, mnogi studiji odlučuju se za korištenje postojećih programskih okvira poznatih kao game engine (Halpern, 2018).

2. Značaj videoigara

2.1. Razvoj PC igrica

Razvoj videoigara započeo je u istraživačkim laboratorijima sredinom 20. stoljeća, postavljajući temelje za globalnu industriju koja danas obuhvaća milijarde igrača. Prvi značajan korak dogodio se 1952. godine kada je britanski profesor A. S. Douglas razvio OXO, verziju igre križić-kružić, kao dio svoje doktorske disertacije na Sveučilištu u Cambridgeu. Samo nekoliko godina kasnije, 1958. godine, William Higinbotham stvorio je Tennis for Two, jednu od prvih interaktivnih igara, koristeći analogno računalo i osciloskopski ekran u Brookhaven National Laboratoryju.

Godine 1962. Steve Russell s Massachusetts Institute of Technology razvio je Spacewar!, prvu računalnu videoigru koja se mogla igrati na više računala, što je bio veliki iskorak za računalne igre. Do kraja šezdesetih godina, Ralph Baer i njegov tim u Sanders Associates, Inc. izumili su "Smeđu kutiju", prototip sustava za videoigre koji se mogao igrati na televiziji. Taj uređaj je 1972. godine licenciran Magnavoxu i postao poznat kao Odyssey, prva kućna konzola za videoigre koji je prikazan u slici 1. Jedna od igara s Odyssey konzole inspirirala je Atarijev Pong, prvu komercijalno uspješnu arkadnu videoigru, koja je 1975. godine dobila i svoju kućnu verziju. U isto vrijeme, industrija je nastavila rasti s razvojem novih tehnologija i ideja, Atari koji se nalazi u slici 2, je 1977. godine izdao Atari 2600, konzolu s izmjenjivim igrama, što je označilo početak druge generacije konzola za videoigre (editors, 2017).

Slika 1. Prva igraća konzola Odyssey (Burke, 2023)

Kasne 1970-te i rane 1980-te bile su ključne godine za industriju videoigara. Izdanje igre Space Invaders 1978. godine, osnivanje Activisiona, prvog nezavisnog studija za razvoj igara, te dolazak japanskog Pac-Mana u Sjedinjene Države, označili su prekretnice u razvoju. U tom periodu, Nintendo je predstavio Donkey Kong, igru koja je uvela svjetski poznatog lika Marija, dok je Microsoft 1982. godine izdao svoj prvi Flight Simulator, postavljajući temelje za razvoj simulacijskih igara na računalima (editors, 2017).

Slika 2. Atari 2600 Game System (play, 2007)

Ovi rani uspjesi postavili su temelje za budući razvoj računalnih igara, omogućivši daljnji rast i evoluciju industrije koja se danas proteže na sve dijelove svijeta.

2.2. Žanr

Videoigre su kategorizirane po žanrovima kako bi igrači lakše prepoznali kakav tip igre mogu očekivati i odabrali iskustvo koje najbolje odgovara njihovim preferencijama. Neki od najpoznatijih žanrova uključuju akcijske igre, RPG-ove (role-playing games), strategije, simulacije, sportske igre i puzzle igre (Shahbazi, 2024). Svaki žanr ima nešto po čemu je poseban i prepoznatljiv. Akcijske igre često se fokusiraju na brzu i dinamičnu igru, gdje je refleks ključan, dok RPG-ovi omogućuju igračima da preuzmu ulogu lika, razvijaju njegove vještine i donose odluke koje oblikuju tijek priče.

Metroidvanija je specifičan podžanr akcijsko-pustolovnih igara koji se ističe svojim jedinstvenim pristupom dizajnu svijeta i igrivosti. Naziv ovog žanra dolazi od kombinacije dvaju kultnih igara, Metroid i Castlevania, koje su postavile temelje za ovaj stil igre. U

Metroidvanija igrama, igrači istražuju velike, međusobno povezane svjetove koji su često nelinearni, što znači da ne slijede strogo definiranu putanju, već zahtijevaju od igrača da se kreću naprijed-nazad, otkrivajući nove dijelove mape (Stegner, 2021).

Ključna karakteristika ovog žanra je postupno otključavanje novih sposobnosti ili opreme koje omogućuju igračima pristup područjima koja su na početku igre bila nedostupna. Na primjer, igrač može naći vrata koja su u početku zaključana ili prepreku koju ne može savladati bez određene vještine, poput dvostrukog skoka ili mogućnosti prolaska kroz uske prolaze. Tek kasnije u igri, nakon što steknu potrebne sposobnosti, igrači se mogu vratiti na ta mjesta i istražiti ih dalje.

Ovaj ciklus povratka i istraživanja stvara osjećaj postepenog napredovanja i nagrađivanja, jer svaki novi posjet već poznatim lokacijama može otkriti skrivene tajne, poput novih predmeta, nadogradnji ili dodatnih priča. Borba također igra važnu ulogu, često zahtijevajući od igrača da savladaju velike i male neprijatelje koristeći kombinaciju novostečenih vještina i strateškog razmišljanja. Ovakav dizajn potiče igrače na istraživanje i eksperimentiranje, stvarajući duboko uranjajuće iskustvo koje nagrađuje strpljenje i domišljatost. Igrica „Void Born“ koristi iste te koncepte te spada u kategoriju metroidvanije.

3. Slične igre

3.1. Hollow knight

Hollow Knight je akcijsko-pustolovna igra iz žanra metroidvanije, razvijena od strane studija Team Cherry. Smještena je u prekrasno animirani, ručno crtani svijet Hallownest, drevno kraljevstvo koje je palo u zaborav. Igrač preuzima ulogu tajanstvenog viteza koji istražuje ovaj podzemni svijet, prepun raznolikih neprijatelja, zamršenih staza i skrivenih tajni. Slika 3 prikazuje glavnog lika kako istražuje podzemnu okolinu.

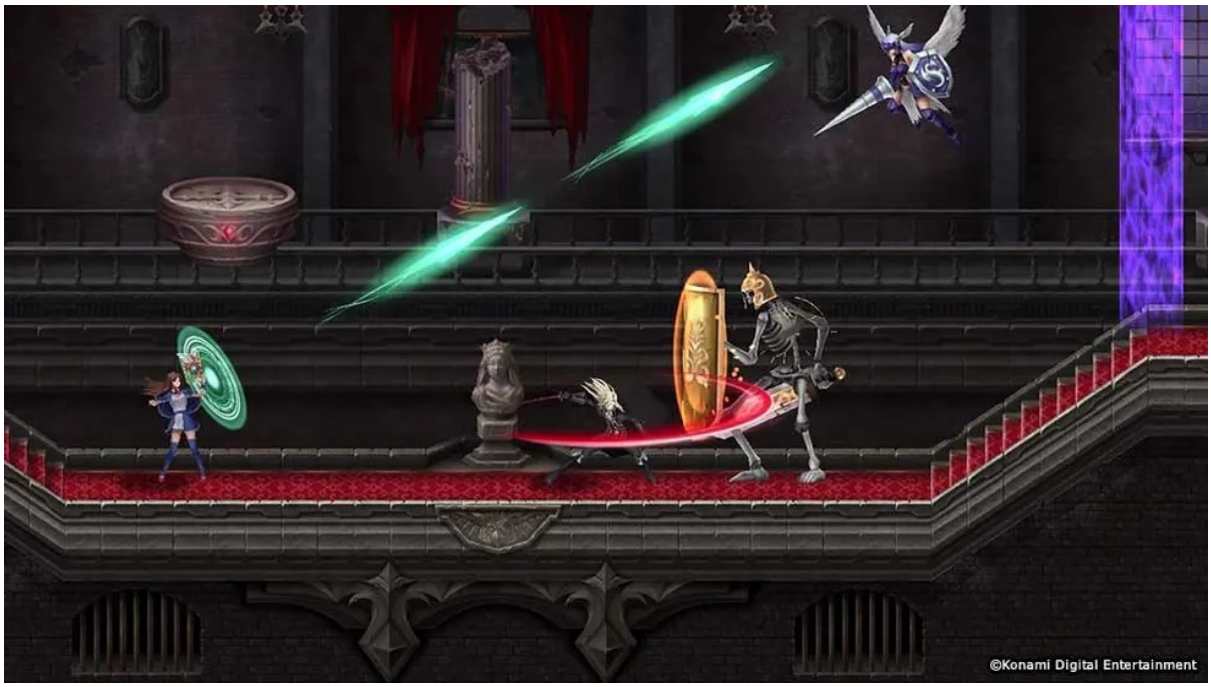
Slika 3. Hollow knight video igra (Cherry, 2017)

Igra se ističe izazovnim borbama i kompleksnim dizajnom mape, gdje igrači otkrivaju nove sposobnosti koje im omogućuju pristup ranije nedostupnim područjima. Atmosfera igre je mračna i tužna, a prati je bogata zvučna podloga koja pojačava osjećaj misterije i samoće. Hollow Knight je poznat po svojoj dubokoj priči, koja se otkriva kroz interakcije s NPC-ovima i skrivenim zapisima u svijetu igre. Igra je hvaljena zbog svoje umjetničke vrijednosti, fluidnih kontrola i zadivljujuće težine, te je postala jedan od najvažnijih naslova u metroidvanija žanru.

3.2. Castlevania

Castlevania je klasična akcijsko-pustolovna igra, originalno razvijena od strane Konamija i prvi put objavljena 1986. godine. Radnja je smještena u gotički svijet, gdje igrač preuzima

ulogu Simona Belmonta, lovca na vampire, koji se mora suočiti s grofom Drakulom i njegovim legijama čudovišta u Drakulinom dvorcu. Slika četiri prikazuje borbu između glavnog lika i neprijatelja.



Slika 4. Castlevania video igra (Luke, 2024)

Igra se ističe svojom mračnom atmosferom, kompleksnim dizajnom nivoa i izazovnim borbama, gdje igrači koriste mogu koristiti razna oružija koja će pronalaziti kako putuju kroz mračni dvorac. Castlevania je također poznata po svojoj glazbi, koja kombinira klasične horor motive s energičnim ritmovima, pojačavajući osjećaj napetosti i opasnosti.

Kombinirajući elemente platformera s akcijom, Castlevania je postavila temelje za buduće naslove u serijalu i pridonijela razvoju metroidvanija žanra. Igra je postala kultni klasik, hvaljena zbog svog inovativnog gameplaya i utjecaja na razvoj videoigara, čime je postala jedan od najprepoznatljivijih naslova u povijesti igara.

3.3. Ori and the Will of the Wisps

Ori and the Will of the Wisps je predivna akcijsko-pustolovna igra iz žanra metroidvanije, razvijena od strane studija Moon Studios i objavljena 2020. godine. Smještena je u očaravajući, ručno crtani svijet koji se zove Niwen, prikazan u slici 5, gdje igrač preuzima ulogu Ori, malog duhova šumskih bića, u potrazi za pronalaženjem izgubljenih prijatelja i vraćanjem ravnoteže u svijetu.

Slika 5. Ori and the Will of the Wisps video igra (Tyrell, 2020)

Igra se ističe svojom izuzetnom vizualnom prezentacijom, s bogatim, detaljno animiranim okruženjima i fluidnim animacijama koje pružaju impresivno estetsko iskustvo. Ori and the Will of the Wisps nudi dinamičan gameplay s dubokim borbenim mehanikama, gdje igrači koriste razne nadogradnje i sposobnosti za borbu protiv opasnih neprijatelja i prevladavanje kompleksnih platformskih izazova.

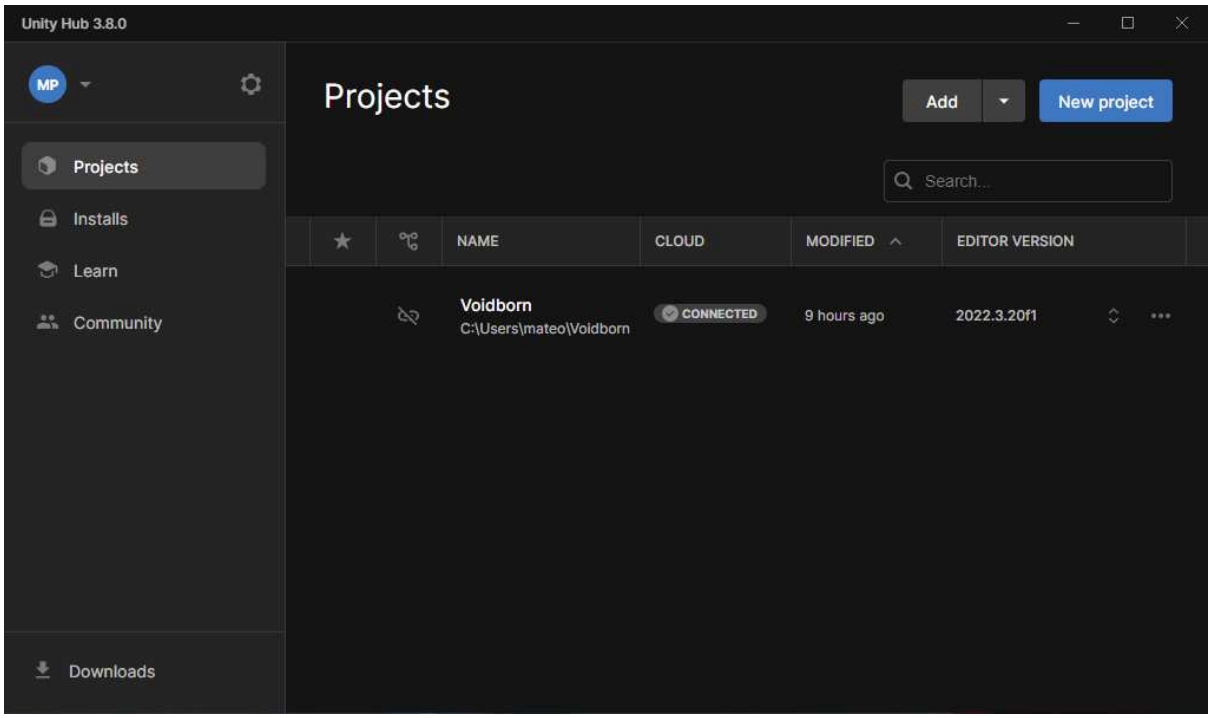
Glazba igre doprinosi njenom emocionalnom naboju, stvarajući upečatljive i dirljive trenutke kroz cijelo iskustvo igre. Priča igre je emotivna i dirljiva, otkrivajući složene teme prijateljstva, hrabrosti i samožrtvovanja. Ori and the Will of the Wisps je hvaljena zbog svoje inovativne igrivosti, predivne grafike i duboke naracije, čineći je jednim od najzapaženijih naslova u modernom metroidvanija žanru.

4. Implementacija igre

4.1. Novi projekt

Unity je softver dostupan na sva tri popularna operativna sustava: Windows, macOS i Linux. Iako je Unity besplatan za korištenje, za ozbiljne tvrtke koje žele razvijati visokokvalitetne igre, dostupna je plaćena verzija pod nazivom Unity Pro, koja nudi napredne značajke i podršku (Unity, Unity, 2024).

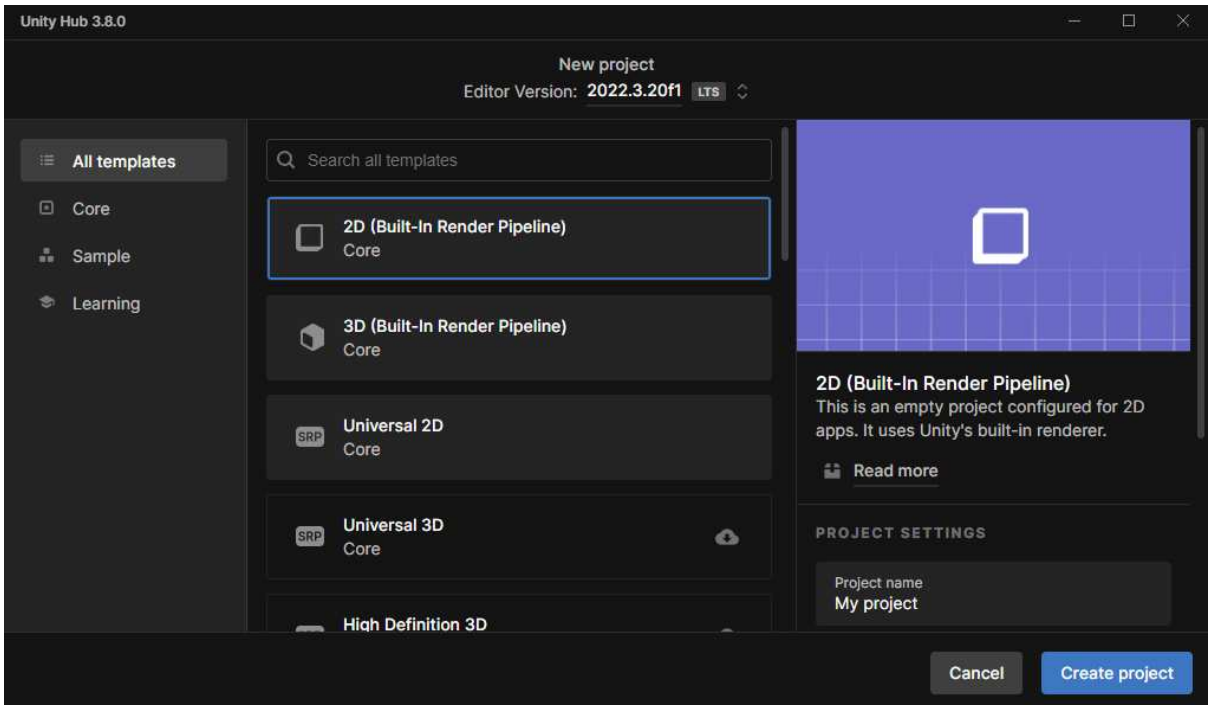
Nakon preuzimanja programa Unity započeti ćemo sa pravljanjem novog projekta. Prvi prozor koji se prikazuje je Unity Hub i prikazan je u slici 6. Preko Unity Hub-a imamo mogućnost dodati već postojeće projekte, napraviti novi projekt, instalirati druge verzije programa, pogledati što su drugi programeri koji koriste Unity napravili i naučiti kako koristiti Unity.



Slika 6. Unity Hub

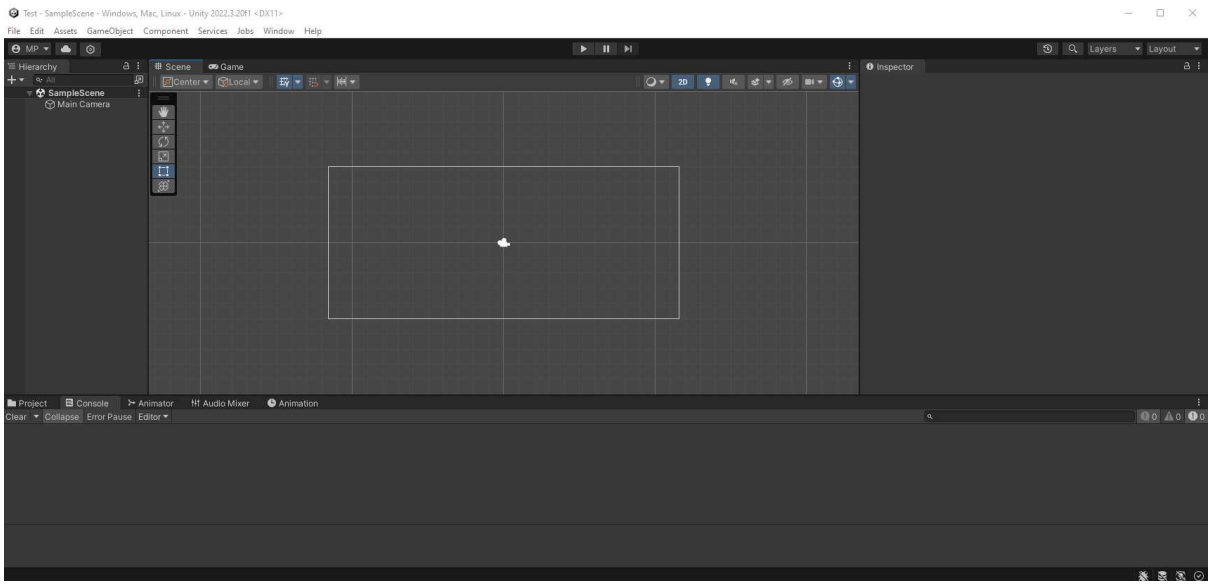
Klikom na gumb New Project u Unity, otvaraju se različite opcije koje omogućuju odabir vrste igre koju želimo razvijati. Ove opcije obuhvaćaju razne predloške prilagođene različitim vrstama igara, poput 2D, 3D, VR, ili mobilnih aplikacija. Iako se sučelje i opcije mogu povremeno ažurirati, predlošci olakšavaju novim korisnicima snalaženje u programu. Omogućuju brzi početak rada s postavkama koje su već optimizirane za određeni tip projekta,

čime se smanjuje potreba za ručnim konfiguriranjem i ubrzava proces razvoja igre kao što je prikazanu na slici 7.



Slika 7. Izrada prve aplijacije

Nakon što je projekt napravljen, otvara se program u kojemu će korisnik moći napraviti svoju igricu. Iako Unity izbornik, koji je prikazan na slici 8, izgleda komplicirano, dijeljenjem njegovih komponenti na više dijelova će nam omogućiti lakše razumijevanje.



Slika 8. Prikaz glavnog sučelja

Nakon što je projekt kreiran, Unity se otvara s glavnim sučeljem u kojem korisnik može započeti izradu svoje igre. Iako na prvi pogled izbornik Unity-a može djelovati složeno, ako ga podijelimo na ključne komponente, postaje puno lakše razumjeti njegovu strukturu i funkcionalnost.

Glavne komponente Unity sučelja uključuju (Unity, Unity Documentation, 2024):

1. Scene View: Ovo je glavni prostor za rad u kojem možete vizualno uređivati svoju scenu, postavljati objekte, kamere i svjetla. Nalazi se u sredini sučelja.
2. Game View: Ovaj prozor prikazuje kako će igra izgledati kada se pokrene. Ovdje možete testirati svoje scene u stvarnom vremenu.
3. Hierarchy Panel: Prikazuje sve objekte u trenutnoj sceni. Ovdje možete pregledavati i organizirati objekte u hijerarhiji. Nalazi se na lijevoj strani sučelja.
4. Project Panel: Ovdje se nalazi pregled svih datoteka i resursa u vašem projektu, uključujući skripte, modele, teksture i zvukove. Nalazi se u donjem dijelu sučelja
5. Inspector Panel: Prikazuje detaljne postavke i atribute za odabrani objekt. Ovdje možete prilagođavati komponente i parametre svakog objekta u sceni. Nalazi se na desnome dijelu sučelja
6. Console: Prikazuje poruke o pogreškama, upozorenja i logove koji su korisni za praćenje stanja igre i debugging.

4.2. Unity asseti

Kako bismo započeli izradu igre u Unity, važno je razumjeti što su asseti, kako ih koristiti i gdje ih možemo preuzeti. Asseti su sve datoteke i resursi koji se koriste u određenom projektu, poput (Games, 2023):

1. Modela – oblikovanje objekata
2. Tekstura – dodavanje vizualnih detalja površinama
3. Zvukova – stvaranje akustičnih efekata
4. Animacija – prikaz kretanja i prikaz promjena
5. Skripti – logika unutar igre

Ti resursi čine osnovu igre i omogućuju stvaranje različitih vizualnih i funkcionalnih elemenata.

Najčešći način dodavanja asseta je jednostavno povlačenje i ispuštanje datoteka iz vašeg sustava u Project Panel u Unity. Ovdje se asseti organiziraju u mape, a vi ih možete lako pronaći i koristiti u sceni. Unity također ima vlastitu trgovinu, Unity Asset Store, gdje možete preuzeti besplatne ili plaćene resurse koje su kreirali drugi developeri. Ovi asseti mogu značajno ubrzati razvoj vaše igre jer vam omogućuju korištenje unaprijed izrađenih modela, zvukova, skripti i drugih resursa. Unity također nudi Package Manager, alat koji omogućuje preuzimanje i upravljanje službenim paketima. Ovi paketi često sadrže dodatne alate, skripte ili čak cijele module (poput sustava za fiziku, mrežno igranje, ili VR podršku) koje možete uključiti u svoj projekt.

Vrste asseta u Unity (Games, 2023):

3D modeli: Koriste se za stvaranje likova, okruženja i objekata u igri.

2D spriteovi: Grafički elementi za 2D igre, poput likova, pozadina i objekata.

Zvukovi i glazba: Dodaju atmosferu i interaktivnost vašoj igri.

Animacije: Postavljaju pokrete i akcije vaših likova ili objekata.

Skripte: Pisane u C#, omogućuju vam dodavanje logike i ponašanja u vašu igru.

Materijali i teksture: Primjenjuju se na 3D modele kako bi im dali boju, refleksije i druge vizualne efekte.

Jednom kada dodate asset u svoj projekt, jednostavno ga možete povući iz Project Panela u Scene View kako biste ga postavili u svoju scenu. Na primjer, ako imate 3D model lika, možete ga povući u scenu i odmah početi raditi na njegovoj animaciji ili interakciji s drugim objektima.

4.3. Izrada okoline

Oblikovanje okoline unutar igre započinje izradom terena, koji predstavlja osnovu za sve ostale elemente unutar igre. U 2D igrama, umjesto trodimenzionalnih terena, koriste se 2D objekti poput ploča (tiles), spriteova i pozadina, koji zajedno čine vizualni prikaz svijeta igre. Prilikom kreiranja novog 2D projekta u Unity, preporučuje se odabir 2D predloška, koji automatski postavlja ključne postavke, poput 2D renderanja i fizike, olakšavajući time rad s 2D sadržajem.

Dodavanje 2D objekata u scenu odvija se putem izbornika GameObject > 2D Object, gdje se može birati između različitih vrsta objekata, kao što su Sprite, Tilemap ili UI elementi. Sprite objekti predstavljaju osnovne grafičke elemente igre, poput likova, prepreka ili dekoracija. Ovi

objekti mogu se dodati povlačenjem željenog spritea iz Project Panela u Scene View ili putem funkcije uvoza datoteka (import file).

Tilemap alat omogućuje jednostavnu izradu složenih razina koristeći ploče (tiles), gdje je moguće odabrati i postaviti pojedinačne ploče, stvarajući dinamičnu i prilagodljivu okolinu. Svojstva dodanih objekata prikazuju se u Inspector Panelu, gdje je moguće mijenjati parametre kao što su veličina, boja, pozicija i sloj objekta. Prilikom rada s Tilemapom, alat Tile Palette omogućuje jednostavno postavljanje i uređivanje ploča unutar scene, s mogućnošću kreiranja prilagođenih paleta za različite dijelove razine, poput podova, zidova ili pozadina.

Za složenije razine, moguće je dodati više slojeva unutar Tilemapa kako bi se postigao dojam dubine ili kreirale složenije strukture. Tako, na primjer, jedan sloj može biti rezerviran za podlogu, drugi za prepreke, a treći za pozadinske elemente. Korištenje Collider komponenti na dodanim objektima omogućuje definiranje fizičkih granica objekata, što omogućava interakciju likova s terenom, poput hodanja po podu ili sudaranja s preprekama (Giri, 2022).

Slika 9. Prikaz osnovnih spriteova u Unity

Svaka okolina u „Void Born“ smještena je unutar vlastite zasebne scene, dizajnirane kako bi igraču pružila iskustvo istraživanja. Ove su scene međusobno povezane, stvarajući prostran svijet koji igrač može slobodno istraživati. Povezanost između različitih okolina omogućuje igraču da samostalno bira svoj put kroz igru, otkrivajući nove izazove, skrivene tajne i alternativne rute. Takav dizajn potiče osjećaj slobode i avanture, jer igrači nisu ograničeni na linearnu progresiju, već imaju priliku sami odlučiti kojim će redoslijedom istraživati različite dijelove svijeta u stilu u klasičnom žanru metroidvanije. Ova nelinearnost omogućuje svakom igraču da doživi igru na svoj način, birajući strategije i rute koje najbolje odgovaraju njegovom stilu igre. Povezanost scena također doprinosi dinamičnom i fluidnom osjećaju igre, gdje svaki prijelaz između okolina donosi novo uzbuđenje i izazov.

4.3.1. Okolina – Mahovinska šuma

„Void Born“ se sastoji od pet različitih okolina, pri čemu svaka od njih donosi jedinstven ugođaj, specifične neprijatelje, te zamke i prepreke koje igrač mora svladati kako bi napredovao u igri. Prva od tih okolina je mahovinska šuma (Maaot, Itch.io, 2019), koja je prikazana na slici 10. Ova lokacija posebno se ističe bogatim zelenilom i živopisnim vizualima, uz spriteove koji vjerno prikazuju prirodnu atmosferu šume prekrivene mahovinom. Mahovinska šuma se sastoji od raznolikih neprijatelja koji su specifični za to područje, uključujući agresivne gljive, šišmiše i sluzavce koji zajedno stvaraju izazovno okruženje za igrače.

Slika 10. Prikaz mahovinske šume

Mahovinska šuma nije samo početna zona igre, već i ključno područje koje se povezuje s drugim okolinama unutar svijeta „Void Born“-a. Igrači će se na početku možda osjećati izgubljeno u gustoj šumi zbog kompleksnog i neprohodnog terena, što povećava osjećaj avanture i neizvjesnosti. Međutim, kako igra napreduje, igrači će otkriti posebne vještine, poput dvostrukog skoka, koje bitno mijenjaju dinamiku igre i olakšava prolazak kroz šumu. Ova vještina omogućuje lakšu navigaciju kroz šumu, otvarajući nove putove i skrivene prolaze, što igraču pruža osjećaj postignuća i omogućuje mu da temeljitije istraži sve tajne koje ovo područje skriva.

Uz napredak kroz mahovinsku šumu, igrači će također susresti zamke koje testiraju njihove refleksne sposobnosti i logičko razmišljanje. Ova okolina postavlja temelj za ton i atmosferu cijele igre, pružajući igračima uvod u izazove i nagrade koje ih očekuju u daljnjim okolinama „Void Born“-a.

4.3.2. Okolina – Duboka špilja

Za razliku od mahovinske šume, Duboka špilja predstavlja mnogo veći izazov za igrače zbog svoje prostranosti i opasnih zamki koje se kriju na svakom koraku (Maaot, itch.io, 2020). Špilja, koja je prikazana na slici 11, je ogromna s brojnim padinama i strmim liticama, gdje svaki pogrešan korak može značiti pad u zamku. Igrači će morati biti iznimno oprezni dok se suočavaju s neprijateljima u ovoj tjeskobnoj i zlokobnoj atmosferi.

Slika 11. Prikaz špilje

Ambijent Duboke špilje značajno se razlikuje od prijašnje okoline. Mračna i tmurna atmosfera, prožeta osjećajem opasnosti, tjera igrače da pažljivo promišljaju svaki svoj potez. Ova okolina testira ne samo igračeve borbene vještine, već i njegovu sposobnost prilagodbe i snalaženja u nepoznatim uvjetima. Za razliku od šume, prostor je puno otvoreniji, no to će samo otežati prolazak kroz ovu tešku okolinu.

Kako bi stigao do samog dna špilje, igrač će morati prevladati razne izazove, uključujući složene platformerske sekvence, zamke koje zahtijevaju brzo razmišljanje, i borbe s neprijateljima. Duboka špilja ne oprašta pogreške, ali pruža i osjećaj postignuća kada igrač uspješno savlada sve prepreke i otkrije tajne koje se skrivaju u njezinim najmračnijim dubinama.

4.3.3. Okolina – Stari dvorac

Stari dvorac predstavlja najteži izazov u igri. Ovaj stari i napušten dvorac je prepun zamkama i neprijateljima koji ne ostavljaju prostora za pogreške (brullov, 2018). Svaka prostorija dvorca nosi sa sobom novu opasnost, bilo da se radi o zamkama, složenim penjanjem ili spuštanjem ili izuzetno snažnim neprijateljima.

Atmosfera Starog dvorca se vidi na slici 12 i potaknuta je mračnim srednjovjekovnim ugođajem, s ruševnim hodnicima i oronulim zidinama. Ulazak u dvorac zahtijeva visoku razinu pripremljenosti. Igrač će se morati osloniti na sva dosad stečena unaprijeđena i vještine kako bi imao šanse za preživljavanje. Svaka bitka unutar dvorca je test izdržljivosti, dok neprijatelji koriste raznovrsne taktike i napade kako bi zaustavili napredak igrača.

Slika 12. Prikaz dvorca

4.3.4. Okolina – Koruptirana šuma

Visoko iznad mahovinske šume, zlokobne sile su koruptirale nekad bujni zeleni okoliš, pretvarajući ga u prijeteće i mračno područje, kao što se vidi na slici 13. Ono što je nekada bilo svijetlo i životno, sada je prekriveno sjenama i ispunjeno opasnostima koje vrebaju iz svakog ugla. Igrači će se u ovoj okolini suočiti s moćnim neprijateljima, predvođenima čarobnjakom, čija prisutnost unosi strah u sve što se nađe na njegovom putu (corwin-zx, 2022).

Slika 13. Prikaz Koruptirana šuma

Ova okolina odlikuje se tamnim, gotovo sablasnim izgledom. Oštri obrisi mrtvih stabala i iskrivljenih grana, uz tamne oblake koji prekrivaju nebo, stvaraju osjećaj nelagode i neizvjesnosti. Igrač morati pronaći put kroz ovaj mračni i neprijateljski prostor, suočavajući se s hordama opasnih i starih stvorenja i drugim zloslutnim silama koje su preuzele kontrolu nad ovim područjem.

4.3.5. Okolina – Praznina (Void)

Praznina je posljednja okolina u igri, gdje se igrač susreće sa glavnim neprijateljem igrice (ansimuz, 2022). Praznina se prikazuje na slici 14 kao područje koje je potpuno sakriveno od ostatka svijeta.

Slika 14. Praznina

Vizualno, prazina je u potpunosti izolirana od svijeta i može se doći do nje samo nakon što igrač uspješno pobijedi bitku sa glavnim neprijateljem u dvorcu, nakon čega igrač se može teleportirati ovdje i suočiti sa finalnim neprijateljem.

4.4. Izrada glavnog lika

Glavni lik u igri mora biti pažljivo usklađen s vizualnim i atmosferskim karakteristikama okoline u kojoj se nalazi. Ovo je ključno za postizanje dosljedne i uvjerljive vizualne estetike koja će igraču omogućiti uranjanje u svijet igre. Budući da su mnoge piksel-art igre zasnovane na spriteovima, odabir i implementacija tih spriteova zahtijeva posebnu pažnju kako bi se osigurala estetska koherencija s okolinom.

Spriteovi su 2D slike koje predstavljaju likove, objekte i druge elemente unutar igre. Prilikom odabira spriteova za glavnog lika, važno je uzeti u obzir stil igre – bilo da se radi o retro piksel umjetnosti, modernom piksel-artu ili nekoj drugoj vizualnoj estetici. Spriteovi bi trebali reflektirati boje, oblike i stil koji odgovara specifičnoj okolini u kojoj će se lik nalaziti. Na primjer, ako je okolina igre mračna i tmurna, spriteovi bi trebali imati tamnije tonove i ozbiljniji izgled kako bi se uklopili u tu atmosferu (Low, 2024). Nakon što su odgovarajući spriteovi odabrani, i dodani u projekt (Ng, 2017), sljedeći korak je kreiranje glavnog lika kao GameObjecta unutar Unity. Ovaj proces uključuje stvaranje GameObjecta koji će se nazvati PlayerController. Ovaj objekt neće samo predstavljati vizualni prikaz lika već će biti i centralni upravljački element koji će rukovoditi svim akcijama glavnog lika, uključujući kretanje, skakanje, interakciju s drugim objektima i likovima, kao i borbu, ukoliko je predviđena u igri. Slika 15 prikazuje glavnoga lika nakon što je importan u projekt.

Slika 15. Player character

Unutar PlayerController objekta, spriteovi se dodaju kao komponente koje definiraju izgled lika. Osim toga, potrebno je implementirati razne skripte koje će definirati ponašanje lika u igri. Na primjer, skripta može kontrolirati kako lik reagira na pritiske tipki, kako se animira tijekom kretanja ili napada, te kako komunicira s fizičkim objektima u svijetu igre. Ispravan odabir spriteova i njihova pravilna integracija unutar PlayerControllera ključni su za osiguranje da glavni lik ne samo da izgleda odgovarajuće u kontekstu okoline, već i da njegovo ponašanje bude u skladu s očekivanjima igrača. To znači da lik treba imati fluidne animacije koje odgovaraju njegovim akcijama, te da treba postojati dosljednost između vizualnog stila spriteova i ostatka igre. Osim toga, pri razvoju lika važno je razmotriti kako će se njegov izgled i ponašanje mijenjati u različitim dijelovima igre. Na primjer, ako igra sadrži različite okoline s različitim stilovima može biti korisno kreirati varijacije spriteova glavnog lika koji će se

prilagođavati tim promjenama. Na taj način, svaki aspekt igre doprinosi stvaranju koherentnog i estetski privlačnog iskustva za igrače, što je ključno za uspjeh igre.

Nakon što je glavni lik uspješno importan u projekt, sljedeći korak je razvoj skripte koja definira njegovo ponašanje. Kako bi osigurali da igrač može napadati u svim smjerovima, slika 16 prikazuje „Attack Settings“ koji konfigurira različite aspekte napada. Ovaj dio koda koristi transforme za određivanje središta područja napada u bočnim, gornjim i donjim smjerovima, te vektore za definiranje veličine tih područja. Damage varijabla definira koliko štete će igrač nanositi neprijateljima, dok slashEffect sadrži animaciju za napad igrača.

```
[Header("Attack Settings:")]
[SerializeField] private Transform SideAttackTransform; //the middle of the side attack area
[SerializeField] private Vector2 SideAttackArea; //how large the area of side attack is

[SerializeField] private Transform UpAttackTransform; //the middle of the up attack area
[SerializeField] private Vector2 UpAttackArea; //how large the area of side attack is

[SerializeField] private Transform DownAttackTransform; //the middle of the down attack area
[SerializeField] private Vector2 DownAttackArea; //how large the area of down attack is

[SerializeField] private LayerMask attackableLayer; //the layer the player can attack and recoil off of

[SerializeField] private float timeBetweenAttack;
private float timeSinceAttack;

[SerializeField] public float damage = 1; //the damage the player does to an enemy

[SerializeField] private GameObject slashEffect; //the effect of the slashs

bool restoreTime;
float restoreTimeSpeed;
[Space(5)]
```

Slika 16 . Prikaz headera za napad

Sljedeća slika 17 prikazuje funkciju koja omogućava igraču da izvrši napad. Funkcija Attack() kontrolira napad tako što prvo provjerava je li prošlo dovoljno vremena od posljednjeg napada, a zatim aktivira animaciju napada i reproducira zvučni efekt. Ovisno o smjeru napada, koji se određuje na temelju varijable yAxis, funkcija koristi odgovarajući Transform (kao što je SideAttackTransform za bočni napad, UpAttackTransform za napad prema gore, ili DownAttackTransform za napad prema dolje) i definira područje napada (SideAttackArea, UpAttackArea, DownAttackArea). Ako je napad prema dolje i igrač nije na tlu, ili napad prema gore kada je yAxis veći od 0, funkcija odgovarajuće postavlja efekte i kontrolira odbijanje kako bi se osigurao realističan odgovor na napad.

```
void Attack()
{
    timeSinceAttack += Time.deltaTime;
    if (attack && timeSinceAttack >= timeBetweenAttack)
    {
        int _recoilLeftOrRight = pState.lookingRight ? 1 : -1;
        timeSinceAttack = 0;
        anim.SetTrigger("Attacking");
        audioSource.PlayOneShot(dashAndAttackSound);

        if (yAxis == 0 || yAxis < 0 && Grounded())
        {
            Hit(SideAttackTransform, SideAttackArea, ref pState.recoilingX, Vector2.right * _recoilLeftOrRight, recoilXSpeed);

            Instantiate(slashEffect, SideAttackTransform);
        }
        else if (yAxis > 0)
        {
            Hit(UpAttackTransform, UpAttackArea, ref pState.recoilingY, Vector2.up, recoilYSpeed);
            SlashEffectAtAngle(slashEffect, 80, UpAttackTransform);
        }
        else if (yAxis < 0 && !Grounded())
        {
            Hit(DownAttackTransform, DownAttackArea, ref pState.recoilingY, Vector2.down, recoilYSpeed);
            SlashEffectAtAngle(slashEffect, -90, DownAttackTransform);
        }
    }
```

Slika 17. Funkcija za napad

Glavna skripta za igrača uključuje funkcije za primanje i prestanak primanja štete. Funkcija TakeDamage(float _damage), koja je prikazana na slici 18, upravlja štetom koju igrač prima: prvo provjerava je li igrač živ i zaustavlja sve prethodne korutine za prestanak primanja štete kako bi se spriječilo preklapanje. Ako je igrač živ, reproducira zvučni efekt ozljede, smanjuje zdravlje igrača i provjerava je li zdravlje palo na nulu. Ako je zdravlje nula ili manje, pokreće se korutina Death() koja upravlja smrtima. Ako igrač nije umro, pokreće se korutina StopTakingDamage() koja omogućava privremenu nepropusnost (invincible status), prikazuje efekt krvi i pokreće animaciju ozljede.

Funkcija StopTakingDamage() upravlja procesom prestanka primanja štete: nakon što se potvrdi da je igrač još uvijek živ, postavlja se status invincible na true, stvara se efekt krvi i pokreće animacija ozljede. Nakon što se animacija odigra, status invincible se vraća na false.

```
public void TakeDamage(float _damage)
{
    if (pState.alive)
    {
        if (!pState.alive) return;
        StopCoroutine("StopTakingDamage");
        audioSource.PlayOneShot(hurtSound);
        Health -= Mathf.RoundToInt(_damage);

        if (Health <= 0)
        {
            Health = 0;
            StartCoroutine(Death());
        }
        else
        {
            StartCoroutine("StopTakingDamage");
        }
    }
}

private IEnumerator StopTakingDamage()
{
        if (!pState.alive) yield break;
    pState.invincible = true;
    GameObject _bloodSpurtParticles = Instantiate(bloodSpurt, transform.position, Quaternion.identity);
    Destroy(_bloodSpurtParticles, 1.5f);

    anim.SetTrigger("TakeDamage");

    yield return new WaitForSeconds(0.25f);

    anim.ResetTrigger("TakeDamage");
    pState.invincible = false;
}
```

Slika 18. Prikaz funkcije za primanje štete

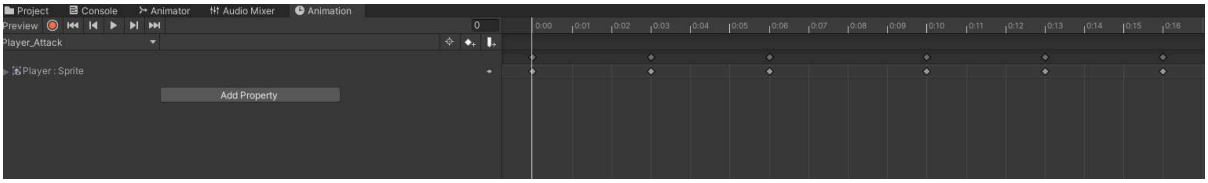
4.4.1. Izrada animacija za glavnog lika

Stvaranje animacija za glavnog lika i ostale objekte u igri predstavlja ključan korak u izgradnji dinamičnog iskustva. Animacije dodaju realizam igri, omogućujući likovima da se kreću, skaču, napadaju i obavljaju druge radnje. U Unity, izrada animacija temelji se na radu sa spriteovima, što je idealno za 2D igre zbog jednostavnog pristupa i interaktivnog sustava.

Prvi korak u izradi animacija uključuje pripremu spriteova, koji su pojedinačne slike koje prikazuju različite faze pokreta lika ili objekta. Na primjer, za animaciju hodanja lika potrebni su spriteovi koji prikazuju različite položaje nogu i tijela tijekom kretanja. Nakon preuzimanja ili izrade spriteova, smještaju se u odgovarajući folder unutar Unity projekta. Ako se koristi sprite sheet, Sprite Editor unutar Unity-ja omogućuje rezanje slike na pojedinačne spriteove.

Ovaj alat omogućava korisnicima da precizno odvoje slike na manje dijelove i koristi ih za izradu animacija.

Nakon pripreme spriteova, slijedi izrada animacija. Izrada animacija je prikazana na slikama 19 i 20. Animator Controller, ključan alat u Unity-u, upravlja animacijama i kreira se desnim klikom unutar Project panela, odabirom opcije Create > Animator Controller. Ovaj objekt omogućuje upravljanje svim animacijama lika. Da bi se započelo s stvaranjem animacija, otvara se Animation prozor putem Window > Animation > Animation. Selektirajući objekt PlayerController u sceni, kreira se nova animacija, poput "Player_Attack", unutar Animation prozora.



Slika 19. Prikaz izrade animacija za napad

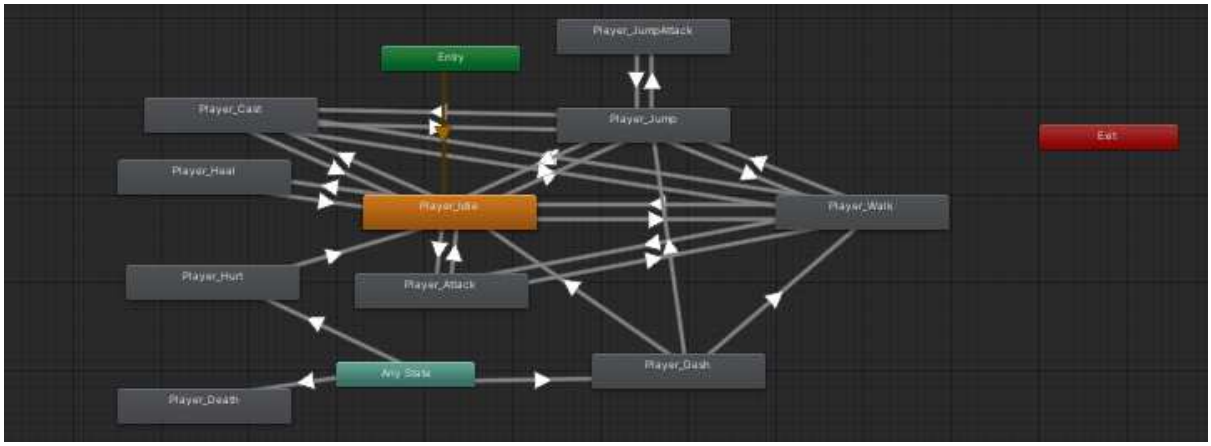
Slika 20. Prikaz završene animacije

Pripremljeni spriteovi povlače se iz foldera u Animation prozor, pri čemu Unity automatski generira ključne frameove za svaku fazu animacije. Brzina animacije se prilagođava promjenom sample rate-a kako bi animacija izgledala glatko, pri čemu se obično koristi između 12 i 24 frameova po sekundi za 2D animacije. Za animacije koje trebaju stalno ponavljanje, kao što su hodanje ili trčanje, u Inspector panelu treba označiti opciju "Loop Time" dok je animacija selektirana.

4.4.2. Animator

Animator Controller u Unity omogućava organiziranje i upravljanje skupom animacijskih isječaka (klipova) i povezanim animacijskim prijelazima za likove ili objekte. Prikaz ne nalazi

na slici 22. Obično se koristi za složenije animacije koje uključuju više stanja i prijelaza između njih. Na primjer, možete automatski prebacivati s animacije hodanja na animaciju skakanja kada pritisnete razmaknicu. Čak i ako imate samo jednu animaciju za vaš GameObject, važno je koristiti Animator Controller. On vam omogućava da postavite i kontrolirate tu animaciju, te upravljate prijelazima između različitih stanja putem parametara. Stanja predstavljaju animacijske klipove, dok prijelazi definiraju kako se prelazi iz jednog stanja u drugo na temelju uvjeta igre. Ovo omogućava fleksibilno i efikasno upravljanje animacijama, kao i jednostavno dodavanje novih animacija u budućnosti. Animator Controller pruža okvir za proširivanje i prilagodbu animacija prema potrebama vaše igre, čime se pojednostavljuje rad sa složenim animacijama (Unity, Animator, 2022).



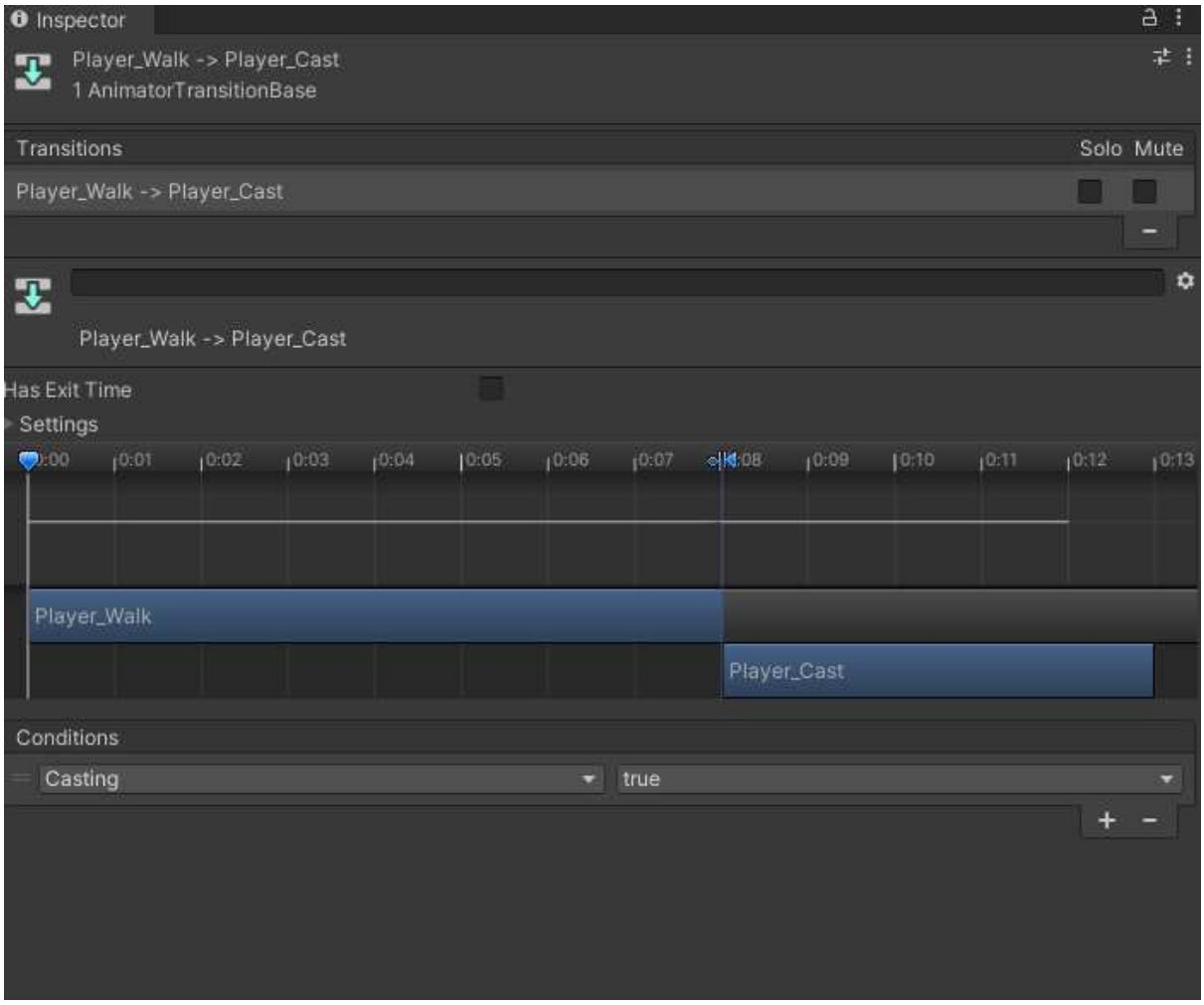
Slika 21. Prikaz animatora za igrača

U svakom animatoru u Unity postoje posebni dijelovi kao što su Any State, Entry i Exit. Svaki od njih ima svoju specifičnu funkciju. Any State omogućava prijelaz između različitih stanja animacije bez obzira na trenutno aktivno stanje. To znači da možete, na primjer, preći s bilo kojeg stanja animacije (kao što su idle, attack, jump) na drugo stanje, poput napada, bez da čekate da se trenutna animacija završi. Entry je početna točka koja označava kada animator ulazi u stanje ili kontroler animacija. Ovdje postavljate početne uvjete i početne animacije. Exit označava kada animator izlazi iz određenog stanja ili kontrolera, što je korisno za čišćenje ili postavljanje novih uvjeta za sljedeće animacije.

Igrač je na početku postavljen u stanje idle, koje je zadano stanje kada igrač stoji na mjestu. Ovo stanje je važno jer predstavlja osnovni, neutralan položaj iz kojeg igrač može započeti bilo koju drugu animaciju. Idle je često povezan s drugim animacijama poput napada (attack), liječenja (heal), bacanja čarolija (cast) ili skakanja (jump), jer se često nakon završetka

specifične animacije želimo vratiti u idle stanje. Ova povezanost omogućava igraču da se brzo vrati u osnovni položaj i tako se pripremi za novu radnju.

Strelice u animatoru predstavljaju prijelaze između različitih stanja animacija i demonstrirane su na slici 23. Koristimo ih kako bismo definirali kako i kada će se animacije mijenjati. Na primjer, ako igrač započne napad, koristit ćemo strijelicu koja će preći iz idle stanja u attack stanje. Strelice se često povezuju s uvjetima, poput pritiska na tipku, promjene varijable ili postizanja određenih uvjeta unutar igre. Kada se ti uvjeti ispune, animator će automatski preći na novo stanje prema definiranom prijelazu (Labz, 2023).



Slika 22. Prikaz tranzicija

Sve ove komponente zajedno omogućuju fluidan i dinamičan prijelaz između različitih animacija, čime se poboljšava iskustvo igre i omogućuje lakše upravljanje animacijama igrača.

4.5. Vještine igrača

Pošto da je „Void Born“ metroidvania, početni set sposobnosti igrača bit će ograničen na osnovne radnje kao što su skakanje, napad i obnavljanje zdravlja kada je ozlijeđen ali kako napreduje kroz svijet igre, otkrit će različite nadogradnje i sposobnosti koje će mu omogućiti pristup novim područjima i poboljšati njegove vještine.

Dok igrač istražuje raznolike lokacije igre, nalaziti će različite vrste unaprijeđenja koja će proširiti njegove mogućnosti istraživanja. Ove nadogradnje mogu uključivati nove sposobnosti, poboljšanja postojećih vještina, ili čak posebne alate koji će mu pomoći da savlada izazove i prepreke koje se nalaze na njegovom putu.

Nadogradnje su ključne za napredovanje u metroidvania igri jer omogućuju igraču da se vrati na prethodna područja s novim sposobnostima i otkriva skrivene tajne ili prolaze do sada nedostupne dijelove svijeta.

Slika 24 prikazuje kod koji upravlja različitim aspektima kontrole igrača tijekom igre. U funkciji Update(), koja se poziva svakog okvira. Ako igrač može nastaviti, kod provjerava različite uvjete i aktivira odgovarajuće funkcije: WallSlide() i WallJump() ako je omogućeno skakanje uz zidove, HandleTeleportation() za teleportaiju u "Void", te StartDash() za dashing. Također, upravlja osnovnim kretanjem igrača, uključujući okretanje Flip(), skakanje Jump(), i kretanje Move().

```
void Update()
{
    if (GameManager.Instance.gameIsPaused) return;
    if(Input.GetKeyDown(KeyCode.O)){
        SaveData.Instance.SavePlayerData();
    }
    if (pState.cutscene) return;
    if (pState.alive)
    {
        if(unlockedWallJump)
        {
        WallSlide();
        WallJump();
        }
        if(unlockedVoid){
            HandleTeleportation();
        }
        if(unlockedDash)
        {
            StartDash();
        }
        if(!isWallJumping)
        {
            Flip();
            Jump();
            Move();
        }

        GetInputs();
        ToggleMap();
        ToggleInventory();
                if (Input.GetKeyDown(KeyCode.Q) && teleportToVoid)
        {
            TeleportToVoid();
        }
```

Slika 23. Prikaz glavne Update funkcije

4.5.1. Dvostruki skok

Dvostruki skok je napredna sposobnost koja omogućava igračima da izvedu dodatni skok dok su još uvijek u zraku, nakon što su već skočili s tla. Ova vještina drastično proširuje mogućnosti kretanja u igri i omogućava veću fleksibilnost pri navigaciji kroz razine i prikazana je na slici 25. Kada igrač aktivira dvostruki skok, može ponoviti skakanje dok se nalazi u zraku, što mu omogućava da prelazi veće razmake između platformi, doseže visoko smještene lokacije ili izbjegne prepreke s većom lakoćom. Na primjer, dvostruki skok može omogućiti igraču da pristupi skrivenim područjima koja su prethodno bila nedostupna ili da prevlada prepreke koje bi bile nemoguće bez dodatnog skoka.

Slika 24. Prikaz dvostrukog skoka

Slika 26 prikazuje funkciju Jump(), koja upravlja skakanjem glavnog lika u igri. Funkcija prvo provjerava je li igrač aktivirao skakanje unutar prihvatljivog vremenskog okvira (jump buffer i coyote time) te ako igrač nije već u skoku (pState. jumping). Ako igrač nije na tlu !Grounded() i još uvijek može izvesti zračni skok (airJumpCounter < maxAirJumps), omogućuje dodatni zračni skok ako je ta opcija omogućena (unlockedVarJump), pri čemu se povećava broj zračnih skokova. Na kraju, ako igrač pusti tipku za skakanje, skakanje se prekida, a brzina u Y osi postavlja na nulu. Os Y je horizontalna pozicija u 2D igricama te je ovdje manipuliramo kako bi omogućili skakanje igraču. Funkcija također ažurira status skakanja u odnosu na stanje tla.

```
void Jump()
{
    if (jumpBufferCounter > 0 && coyoteTimeCounter > 0 && !pState.jumping)
    {
        audioSource.PlayOneShot(jumpSound);
        rb.velocity = new Vector3(rb.velocity.x, jumpForce);
        pState.jumping = true;
    }
    if (!Grounded() && airJumpCounter < maxAirJumps && Input.GetButtonDown("Jump"))
    {
        audioSource.PlayOneShot(jumpSound);
        if (unlockedVarJump)
        {
            pState.jumping = true;
            airJumpCounter++;
            rb.velocity = new Vector3(rb.velocity.x, jumpForce);
        }
    }

    if (Input.GetButtonUp("Jump") && rb.velocity.y > 3)
    {
        pState.jumping = false;
        rb.velocity = new Vector2(rb.velocity.x, 0);
    }

    anim.SetBool("Jumping", !Grounded());
}
```

Slika 25. Prikaz funkcije Jump

4.5.2. Dash

Dash je napredna sposobnost koja omogućava igračima da izvede brzi pokret u odabranom smjer. Ova vještina značajno poboljšava mobilnost u igri i pruža igračima dodatnu fleksibilnost pri kretanju kroz razine. Aktivacijom dash-a, igrač može se naglo pomaknuti u određenom smjeru, što mu omogućava da izbjegne prepreke, preskoči udaljene prostore ili brzo reagira na prijetnje, što je demonstrirano na slici 27. Efikasno korištenje dash zahtijeva pravilan tajming jer igrač mora strateški odabrati kada i kako koristiti ovu vještinu kako bi maksimalno iskoristio njezine prednosti.

Slika 26. Prikaz dash-a

Funkcija Dash(), koja je prikazana na slici 28, upravlja izvedbom brzog kretanja igrača. Na početku funkcije, postavlja se da igrač ne može izvesti novi dash (canDash = false) te se aktivira status dashing i invincibility (pState. dashing = true, pState. invincible = true). Ovo se radi kako bi se osiguralo da ukoliko se igrač prođe kroz neprijatelja da se igrica ne zamrzne. Funkcija zatim pokreće animaciju dash-a i reprodukciju zvuka, dok se gravitacija postavlja na nulu, a brzina u X osi postavlja na određenu vrijednost ovisno o smjeru u kojem igrač gleda. Ukoliko se gravitacija ne postavi na nulu, igrač će biti u not grounded statusu, što neće omogućiti igraču da izađe iz dash-a. Nakon završetka dasha, brzina se postavlja na nulu, a status dashing i invincibility se deaktivira. Funkcija završava vraćanjem statusa na idle i postavljanjem vremena čekanja za dash (dashCooldown), nakon čega igrač ponovno može izvesti dash.

```
IEnumerator Dash()
{
    canDash = false;
    pState.dashing = true;
    pState.invincible = true;
    anim.SetTrigger("Dashing");
    audioSource.PlayOneShot(dashAndAttackSound);
    rb.gravityScale = 0;
    int _dir = pState.lookingRight ? 1 : -1;
    rb.velocity = new Vector2(_dir * 90, 0);

    if (Grounded())
    {
        Instantiate(dashEffect, transform);
    }
    float dashStartTime = Time.time;
    while (Time.time < dashStartTime + dashTime)
    {
        rb.velocity = new Vector2(_dir * 90, rb.velocity.y);
        yield return null;
    }

    rb.gravityScale = gravity;
    rb.velocity = Vector2.zero;
    pState.dashing = false;
    pState.invincible = false;
    anim.SetTrigger("Idle");
```

Slika 27. Prikaz funkcije Dash

4.5.3. Penjanje po zidu

Penjanje po zidu je napredna sposobnost koja omogućava igračima da se penju po vertikalnim površinama, poput zidova i drugih uspravnih objekata u igri. Ova vještina značajno poboljšava kretanje i istraživanje unutar igre, omogućujući igračima pristup visoko

smještenim područjima i dosezanje lokacija koje bi bile nedostupne bez nje. Slika 29 pokazuje igrača kako skače sa zida. Prilaskom do zida, igrač može skočiti i penjati se do novih, prije ne dostupnih lokacija.

Slika 28. Prikaz penjanja po zidu

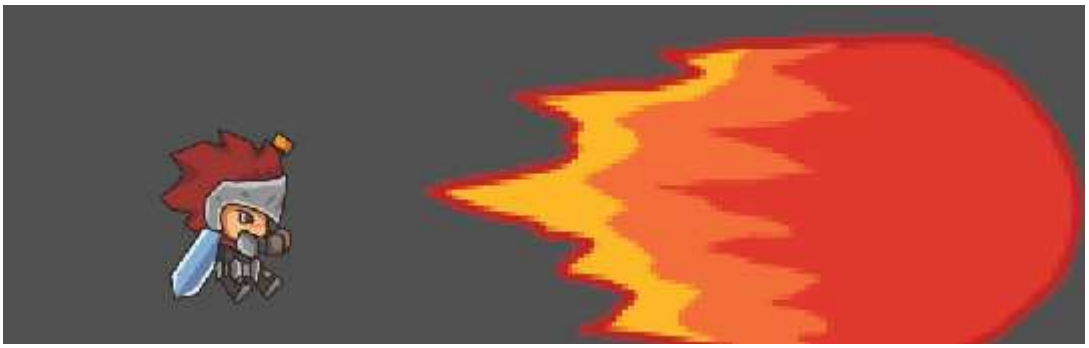
Funkcija WallJump() upravlja skakanjem uz zid. Kao što se vidi na slici 30, ako je igrač na zidu i pritisne tipku za skakanje, igrač će izvesti wall jump u smjeru suprotnom od onoga u kojem gleda. Pri tome se postavlja brzina skoka, poništava prethodni dash i resetira broj zračnih skokova. Također, mijenja se orijentacija lika kako bi gledao u suprotnom smjeru i funkcija se postavlja da se zaustavi nakon određenog vremena.

4.6. Uporaba magije

Osim vještina koje igrač može otključati, u svijetu igre „Void Born“ bit će moguće pronaći razne magije koje će borbe u igrici činiti zanimljivijima. Ove magije predstavljaju posebne sposobnosti koje igrač može koristiti u borbi protiv neprijatelja i za rješavanje različitih izazova unutar igre. Igrač posjeduje mana bar, koji je ključna mehanika za upravljanje magijskim sposobnostima i obnovom života. Mana bar služi kao resurs za korištenje spellova i omogućuje igraču da izvede različite magične napade. Svaki put kada igrač aktivira spell, određena količina mane se troši. Napadanjem neprijatelja, igrač dobiva dio mane nazad što omogućava bitke sa neprijateljima da budu zanimljive jer igrač ne može biti daleko i samo se boriti sa magijom.

4.6.1. Fireball

Fireball je moćan magični napad koji omogućava igraču da napadne neprijatelje iz daljine koristeći vatrenu kuglu prikazanu na slici 30. Ovaj napad je izuzetno koristan zbog svoje sposobnosti da uzrokuje veliku štetu na udaljenosti.



Slika 29. Prikaz fireball spell-a

4.6.2. Upward explosion

Upward Explosion je specijalan magični napad koji omogućava igraču da napadne neprijatelje koji se nalaze iznad njega pomoću snažne eksplozije usmjerene prema gore. Nadap je prikazan na slici 31. Ovaj napad je posebno koristan za borbu protiv neprijatelja koji lete, neprijatelja na višim platformama, ili za obaranje neprijatelja koje se nalaze u visini.

Slika 30. Prikaz upward explosion spell-a

4.6.3. Downward explosion

Downward Explosion je magični napad koji omogućava igraču da napadne neprijatelje sa visine, izazivajući snažnu eksploziju koja se širi od vrha prema dnu, ako što je pokazano na slici 32. Ovaj napad je posebno efikasan za borbu protiv neprijatelja koji se nalaze ispod igrača.

Slika 31. Prikaz downwards explosion spell-a

4.6.4. Funkcije za magiju

Funkcija CastSpell() upravlja procesom bacanja čarolija i ispunjava nekoliko uvjeta kako bi omogućila izvođenje čarolije. Kao što se vidi na slici 33, funkcija provjerava je li igrač pustio tipku za bacanje čarolije, da li je vrijeme od posljednjeg bacanja dostiglo minimalni interval, te da li igrač ima dovoljno mane za bacanje čarolije. Ako su svi uvjeti zadovoljeni, postavlja se stanje za bacanje čarolije i pokreće se CastCoroutine() kako bi se izvršila sama čarolija. Ako uvjeti nisu zadovoljeni, funkcija samo povećava vrijeme od posljednjeg bacanja. Također, ako je igrač na tlu, aktivira ili deaktivira efekt čarolije te, ako je čarolija aktivna, dodaje dodatnu silu prema dolje na igračevu brzinu.

```
void CastSpell()
{
    if (Input.GetButtonUp("Cast/Heal") && castOrHealTimer <= 0.05f && timeSinceCast >= timeBetweenCast && Mana >= manaSpellCost)
    {
        pState.casting = true;
        timeSinceCast = 0;
        StartCoroutine(CastCoroutine());
    }
    else
    {
        timeSinceCast += Time.deltaTime;
    }

    if(Grounded())
    {
        downSpellFireball.SetActive(false);
    }
    if(downSpellFireball.activeInHierarchy)
    {
        rb.velocity += downSpellForce * Vector2.down;
    }
}
```

Slika 32. Prikaz koda za korištenje čarolije

Funkcija CastCoroutine() upravlja procesom bacanja čarolije kroz različite uvjete i animacije, kako je prikazano na slici 34. Funkcija prvo reproducira zvučni efekt bacanja čarolije i, ovisno o trenutnom položaju igrača i odabranoj vrsti čarolije, pokreće različite animacije i efekte. Ako je igrač u poziciji za bočno bacanje (yAxis == 0 ili yAxis < 0 i igrač je na tlu) i ako je bočno

bacanje otključano (unlockedSideCast), pokreće se animacija bacanja, a zatim se stvara bočna vatra. Ako igrač gleda lijevo, vatra se okreće prema lijevo; inače, ostaje okrenuta prema desno. Mana se smanjuje, a funkcija čeka prije nego što završi. Ako igrač gleda prema gore (yAxis > 0) i ako je bacanje prema gore otključano (unlockedUpCast), funkcija pokreće animaciju i stvara eksploziju iznad igrača, dok se brzina igrača postavlja na nulu i mana se smanjuje.

```
IEnumerator CastCoroutine()
{
    audioSource.PlayOneShot(spellCastSound);
    if ((yAxis == 0 || (yAxis < 0 && Grounded())) && unlockedSideCast)
    {
        anim.SetBool("Casting", true);
        yield return new WaitForSeconds(0.15f);
        GameObject _fireBall = Instantiate(sideSpellFireball, SideAttackTransform.position, Quaternion.identity);

        if(pState.lookingRight)
        {
            _fireBall.transform.eulerAngles = Vector3.zero;
        }
        else
        {
            _fireBall.transform.eulerAngles = new Vector2(_fireBall.transform.eulerAngles.x, 180);

        }
        pState.recoilingX = true;
        Mana -= manaSpellCost;
        yield return new WaitForSeconds(0.35f);
    }

    else if( yAxis > 0 && unlockedUpCast)
    {
        anim.SetBool("Casting", true);
        yield return new WaitForSeconds(0.15f);
        Instantiate(upSpellExplosion, transform);
        rb.velocity = Vector2.zero;
        Mana -= manaSpellCost;
        yield return new WaitForSeconds(0.35f);
    }
```

Slika 33. Funkcija bacanja čarolije

4.7. Kontrole

Sve kontrole igrača implementirane su u glavnoj skripti pod nazivom PlayerController. Osnovne funkcionalnosti koje ova skripta pokriva uključuju kretanje lijevo i desno, skakanje, korištenje magije i aktiviranje posebnih vještina, što se vidi na slici 35.

```
void GetInputs()
{
    xAxis = Input.GetAxisRaw("Horizontal");
    yAxis = Input.GetAxisRaw("Vertical");
    attack = Input.GetButtonDown("Attack");
    openMap = Input.GetButtonDown("Map");
    openInventory = Input.GetButtonDown("Inventory");
    if (Input.GetButton("Cast/Heal"))
    {
        castOrHealTimer += Time.deltaTime;
    }
    else
    {
        castOrHealTimer = 0;
    }
```

Slika 34. Prikaz skripte - funkcija GetInputs

Tipke A i D - omogućuju igraču kretanje lijevo i desno po mapi.

Tipka Space - služi za skakanje.

Lijevi klik miša - koristi se za napadanje.

Desni klik miša - aktivira spellove, pri čemu različite kombinacije tipki i klikova omogućuju različite magije:

Desni klik za Fireball.

W + desni klik za Upward Explosion.

S + desni klik za Downward Explosion.

Uz pomoć slike 36, u funkciji Move(), brzina kretanja igrača postavlja se na walkSpeed pomnoženo s xAxis (koji predstavlja horizontalni ulaz, lijevo ili desno kretanje). Ova postavka upravlja brzinom kretanja u vodoravnom smjeru, dok vertikalna brzina (rb. velocity. y) ostaje nepromijenjena. Osim toga, animacija se ažurira postavljanjem parametra "Walking" na true ako je brzina kretanja različita od nule i ako je igrač na tlu, što se provjerava pomoću funkcije Grounded(). Ovaj kod omogućava igraču da se pomiče lijevo ili desno i upravlja animacijom hodanja na temelju trenutnog stanja kretanja i kontakta s tlom.

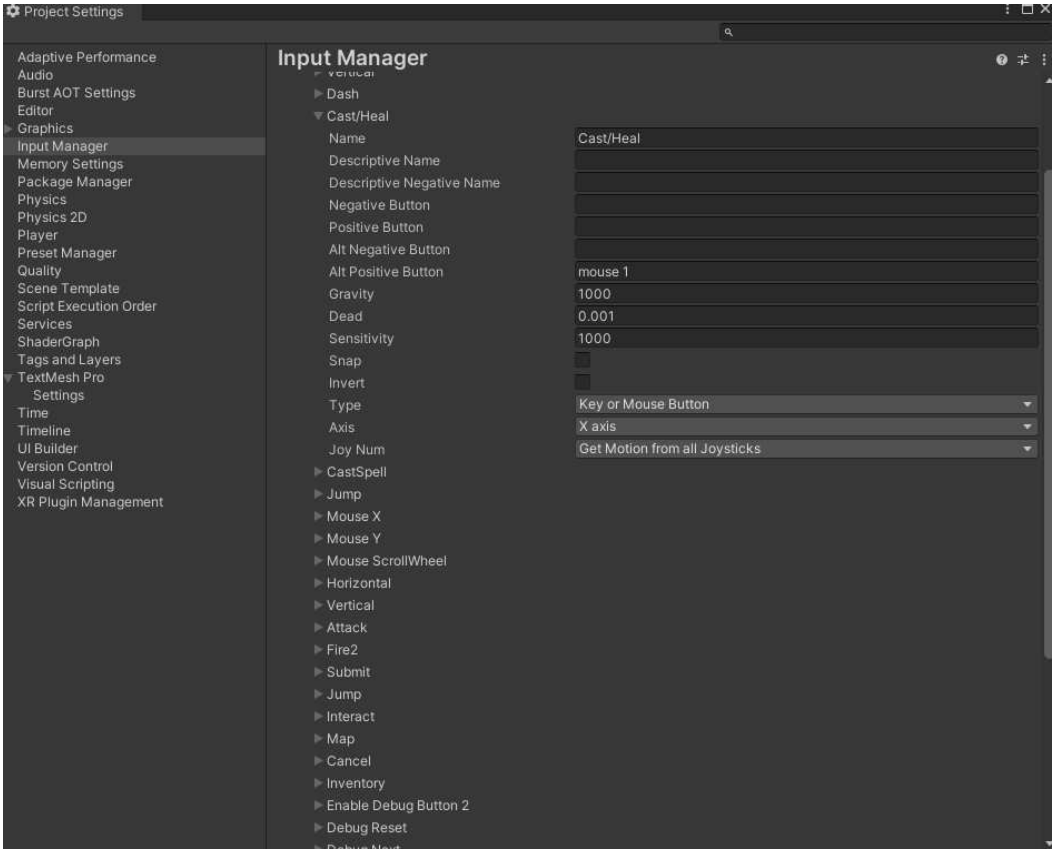
```
315     void Flip()
316     {
317         if (xAxis < 0)
318         {
319             transform.localScale = new Vector2(-4, transform.localScale.y);
320             pState.lookingRight = false;
321         }
322         else if (xAxis > 0)
323         {
324             transform.localScale = new Vector2(4, transform.localScale.y);
325             pState.lookingRight = true;
326         }
327     }
328
329     private void Move()
330     {
331         if (pState.healing) rb.velocity = new Vector2(0, 0);
332         rb.velocity = new Vector2(walkSpeed * xAxis, rb.velocity.y);
333         anim.SetBool("Walking", rb.velocity.x != 0 && Grounded());
334     }
335
336 void StartDash()
337 {
338     if (Input.GetButtonDown("Dash") && canDash && !dashed)
339     {
340         StartCoroutine(Dash());
341         dashed = true;
342     }
343
344     if (Grounded())
345     {
346         dashed = false;
347     }
348 }
349
350 IEnumerator Dash()
351 {
352     canDash = false;
353     pState.dashing = true;
354     pState.invincible = true;
355     anim.SetTrigger("Dashing");
356     audioSource.PlayOneShot(dashAndAttackSound);
357     rb.gravityScale = 0;
358     int _dir = pState.lookingRight ? 1 : -1;
359     rb.velocity = new Vector2(_dir * 90, 0);
360
361     if (Grounded())
362     {
363         Instantiate(dashEffect, transform);
364     }
```

Slika 35. Funkcija za kretanje iz skripte PlayerController

Za upravljanje stanjem igrača koristi se dodatna skripta pod nazivom PlayerStatesList, što demonstrira slika 37. Ova skripta služi za spremanje svih stanja igrača, koja se zatim primjenjuju unutar glavne skripte, čime se osigurava bolja organizacija koda, njegova čitljivost i jednostavnost održavanja.

Slika 36. Prikaz skripte PlayerStateList

Input Manager u Unity služi za konfiguriranje i upravljanje unosom korisnika, uključujući ulaze s tipkovnice, miša i kontrolera. Glavna svrha Input Manager-a je omogućiti definiranje različitih kontrola koje igra koristi. U ovom alatu možete postaviti nazive i vrijednosti za različite ulaze, poput kretanja, skakanja i interakcije, koji se zatim koriste u skriptama za upravljanje ponašanjem igre. Jedna od ključnih funkcionalnosti Input Manager-a je konfiguriranje tipki. Ovdje možete odrediti koje tipke na tipkovnici ili kombinacije tipki odgovaraju određenim akcijama u igri, što omogućuje precizno upravljanje igračevim akcijama. Na primjer, tipka "A" može biti postavljena za kretanje naprijed, dok "D" može pomaknuti igrača unatrag, što je prikazano na slici 38. Osim toga, Input Manager omogućava upravljanje ulazima s različitih uređaja, uključujući miš, tipkovnicu i kontrolere, što je korisno za igre koje se igraju na različitim platformama. Ako dođe do promjena u načinu na koji igra reagira na ulaze, te promjene se mogu lako prilagoditi u Input Manager-u, čime se izbjegava potreba za izmjenom skripti koje koriste te ulaze (Pekar, 2022).



Slika 37. Prikaz input managera

4.8. Izrada neprijatelja

U igrama, neprijatelji igraju ključnu ulogu u oblikovanju iskustva igre i izazovu za igrače. Ovi neprijatelji nisu samo prepreke koje treba savladati, već i elementi koji doprinose istraživačkom i progresivnom aspektu igre. U Metroidvanijama, neprijatelji su često dizajnirani da predstavljaju različite vrste prijetnji i izazova, a njihovo ponašanje i sposobnosti mogu varirati od jednostavnih napadača do kompleksnih protivnika sa specifičnim strategijama i obrascima ponašanja. Na taj način, neprijatelji ne samo da povećavaju težinu igre, već i doprinose naraciji i atmosferi, pomažući igračima da se osjećaju kao da su u živom i dinamičnom svijetu. Neprijatelji su dizajnirani da budu ključni elementi u igri, dodajući dubinu i kompleksnost. Svaki neprijatelj ima jedinstvene karakteristike i ponašanje koje je pažljivo usklađeno s temom i dizajnom igre (Julia, 2019).

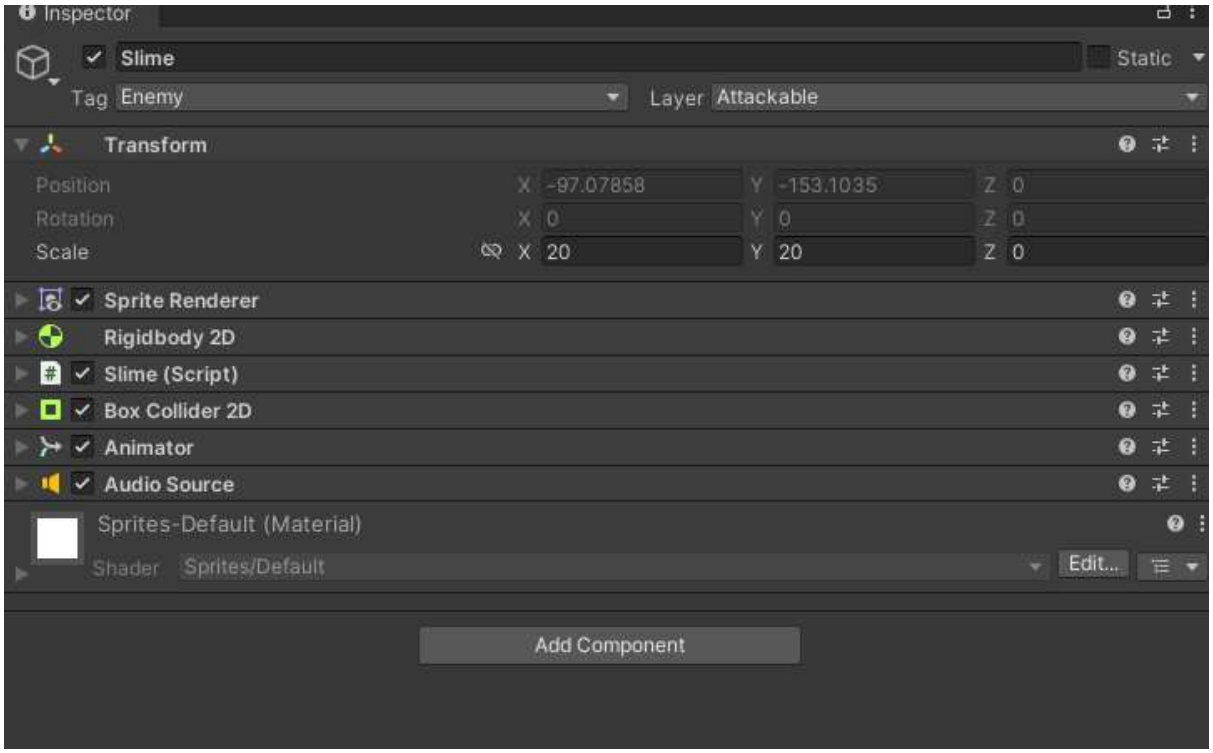
Dizajn i kreiranje neprijatelja u igri „Void Born“ započinje definiranjem njihovog vizualnog izgleda koristeći prethodno definirane spriteove. Na primjeru prikazanom na slici 39, može se vidjeti sprite neprijatelja pod nazivom noćni šišmiš. Nakon što su spriteovi uvezeni u Unity, stvara se novi GameObject unutar scene, kojemu se dodaje Sprite Renderer komponenta. Ova

komponenta omogućava postavljanje vizualnog prikaza neprijatelja na scenu, čineći ga vidljivim u svijetu igre.

Slika 38. Dodavanje neprijateljskoga sprite-a

Nakon definiranja vizualnog izgleda neprijatelja, sljedeći korak uključuje dodavanje potrebnih komponenata za upravljanje interakcijama. Dodavanjem Collider komponente, poput Box Collidera ili Circle Collidera, omogućuje se detekcija sudara neprijatelja s igračem ili drugim objektima unutar igre. U slučaju da neprijatelj treba reagirati na fiziku igre, dodaje se i Rigidbody2D komponenta, koja omogućava fizičke interakcije, poput gravitacije ili sudara s okolinom. Zatim se implementira ponašanje neprijatelja putem skripti koje definiraju osnovne funkcije kao što su kretanje, napadanje, praćenje igrača te reakcije na udarce. Ove skripte omogućuju neprijatelju dinamičnu interakciju unutar igre, koristeći metode poput Update() za kontinuirane radnje ili OnCollisionEnter() za reakcije na sudare. Ove funkcionalnosti

prikazane su na slici 40, gdje se može vidjeti kako neprijatelj reagira na različite situacije u stvarnom vremenu.



Slika 39. Prikaz svih komponenti koji neprijatelji sadrže

4.8.1. Obični neprijatelji

Obični neprijatelji u igri predstavljaju osnovne izazove koje će igrač susretati dok prolazi kroz različite dijelove igre. Svaki od ovih neprijatelja ima jedinstvene karakteristike koje uključuju posebne napade, animacije, te različit broj života i štete koju uzimaju igraču prilikom udarca. Ovi neprijatelji su dizajnirani s različitim AI obrascima i ponašanjima, što doprinosi njihovoj raznolikosti i kompleksnosti.

Njihova uloga u igri nije samo u pružanju izazova, već i u pripremi igrača za nadolazeće borbe sa šefovima. Kroz interakciju s različitim neprijateljima, igrači imaju priliku testirati svoje vještine i strategije, što im pomaže u razumijevanju mehanike igre i pripremi za teže borbe koje će se pojaviti u toj istoj okolini. Ovi neprijatelji igraju ključnu ulogu u postepenom povećanju težine igre i omogućuju igračuu da se bolje upoznaju s pravilima i dinamikom borbe unutar igre.

Skripta Enemy u Unityju definira osnovne karakteristike i ponašanje neprijatelja u igri. Na početku, skripta koristi različite varijable za postavljanje osnovnih atributa neprijatelja, uključujući zdravlje, brzinu kretanja, štetu koju neprijatelj nanosi igraču i dužinu te faktor

reakcije na udarce (recoil), što je prikazano na slici 42. Ove varijable omogućuju prilagodbu i balansiranje neprijatelja prema specifičnostima igre. Također, skripta sadrži različite zvukove i animacije koje se koriste za različita stanja neprijatelja, poput napada, umiranja i hodanja. Kada neprijatelj primi štetu kroz funkciju EnemyHit(), njegovo zdravlje se smanjuje i, ako nije u fazi reakcije, neprijatelj će reagirati na udarac tako što će emitirati zvuk i odgurnuti se u smjeru udarca.

```
82      protected EnemyStates currentEnemyState;
83      protected virtual void Start()
84      {
85          rb = GetComponent<Rigidbody2D>();
86          sr = GetComponent<SpriteRenderer>();
87          audioSource = GetComponent<AudioSource>();
88      }
89      protected virtual void Update()
90      {
91          if (GameManager.Instance.gameIsPaused) return;
92
93          if (isRecoiling)
94          {
95              if (recoilTimer < recoilLength)
96              {
97                  recoilTimer += Time.deltaTime;
98              }
99              else
100             {
101                 isRecoiling = false;
102                 recoilTimer = 0;
103             }
104         }
105         else
106         {
107             UpdateEnemyStates();
108         }
109     }
110
111     public virtual void EnemyHit(float _damageDone, Vector2 _hitDirection, float _hitForce)
112     {
113         health -= _damageDone;
114         if (!isRecoiling)
115         {
116             audioSource.PlayOneShot(hurtSound);
117             rb.velocity = _hitForce * recoilFactor * _hitDirection;
118             isRecoiling = true;
119         }
120
121         if (health <= 0)
122         {
123             Die();
124         }
125     }
```

Slika 40. Skripta za neprijatelje

Svaki neprijatelj koristi ovu skriptu kao kostur svojega ponašanje. Ponašanje neprijatelja se dodatno još definira sa skriptom koja je napisana za posebnoga neprijatelja. Svaki neprijatelj ima svoju skriptu koju koristi kako bi se ponašao posebno i razlikovao od drugoga neprijatelja. Slika 43 prikazuje kako izgleda skripta za neprijateljsku klasu koja se zove bandit.

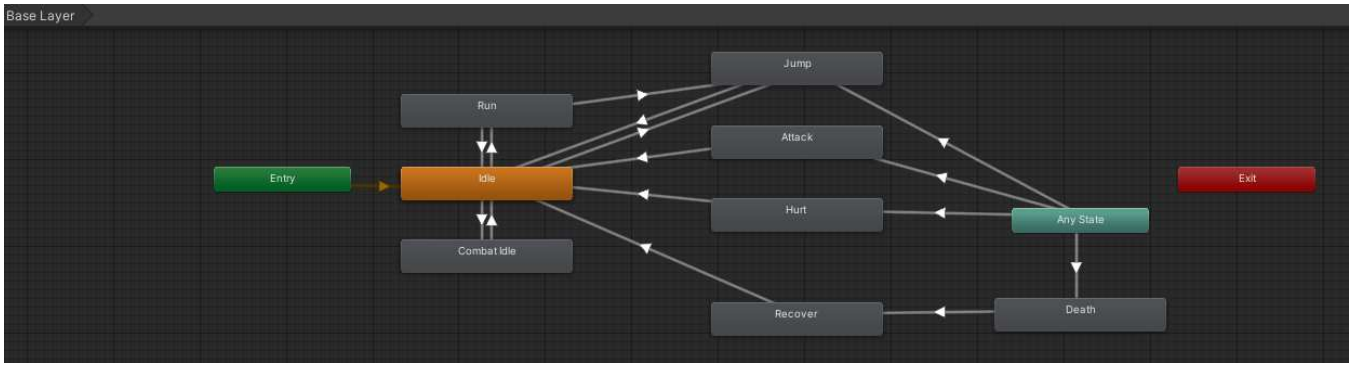
```
public class Bandit : Enemy
{
    [SerializeField] private float chaseDistance;
    [SerializeField] private float attackDistance;
    [SerializeField] private float patrolSpeed;
    [SerializeField] private float chaseSpeed;
    [SerializeField] private float attackCooldown;
    [SerializeField] private float flipWaitTime;
    [SerializeField] private float ledgeCheckX;
    [SerializeField] private float ledgeCheckY;
    [SerializeField] private LayerMask whatIsGround;
    [SerializeField] private float chaseDuration;
    [SerializeField] private BoxCollider2D attackRangeCollider; // Collider for attack range

    private Animator m_animator;
    private float attackTimer;
    private float flipTimer;
    private EnemyStates previousState;
    private bool isAttacking;
    private float chaseEndTime;
    private bool m_grounded;

    private const int AnimState_Idle = 1;
    private const int AnimState_Run = 2;

    private const string Trigger_Attack = "Attack";
    private const string Trigger_Hurt = "Hurt";
    private const string Trigger_Death = "Death";

    protected override void Start()
    {
        base.Start();
        m_animator = GetComponent<Animator>();
        attackTimer = attackCooldown;
        ChangeState(EnemyStates.Bandit_Idle);
        attackRangeCollider.enabled = false;
    }

    protected override void UpdateEnemyStates()
    {
        float distanceToPlayer = Vector2.Distance(transform.position, PlayerController.Instance.transform.position);
        m_grounded = Physics2D.Raycast(transform.position, Vector2.down, ledgeCheckY + 0.1f, whatIsGround);

        if (health <= 0 && currentEnemyState != EnemyStates.Bandit_Death)
        {
            ChangeState(EnemyStates.Bandit_Death);
            StartCoroutine(BanditHandleDeath());
            return;
        }

        switch (currentEnemyState)
        {
            case EnemyStates.Bandit_Idle:
```

Slika 41. Skripta bandit

Bandit je neprijatelj koji patrolira, prati i napada igrača ukoliko su uvjeti ispunjeni. Bandit je u mirnom stanju dok ne započne bitku sa igračem. UpdateEnemyStates() se koristi kako bi definirali što bandit radi u kojemu trenutku. Ukoliko je u mirnom stanju, bandit će se kretati sam. Ukoliko igrač dođe u blizinu bandit će se promijeniti sa funkcijom UpdateEnemyStates() u stanje vijanja i početi će se boriti sa igračem. Prilaskom do igrača, napada ga i ukoliko ga uspješno udari, oduzima mu jedan život. Kada bandit ostane bez života, poziva se funkcija Destoy GameObject() koja ga uklanja iz scene.

4.8.2. Animacije neprijatelja

Svaki neprijatelj u igri koristi vlastite, jedinstvene animacije kako bi prikazao svoje ponašanje i radnje. Animacije igraju ključnu ulogu u tome da neprijatelji izgledaju prirodno u svijetu igre i jasno prenose igraču što se događa na ekranu. To na prikazuje slika 44.



Slika 42. Animator za bandita

Bandit u igri koristi Animator komponentu kako bi upravljao svojim animacijama i ponašanjem. Na početku, Bandit se postavlja na Idle stanje, gdje čeka dok se ne dogodi neka promjena u igri. Animator komponenta omogućava Banditu da koristi različite animacije za različita stanja, kao što su hodanje, napadanje, ozljede i smrt. Skripta povezana s Animatorom koristi informacije iz Animator Controller-a da odredi koje stanje i animacija treba biti aktivna u svakom trenutku. Animator Controller ima posebno konfiguriran Any State dio, koji omogućava Banditu da brzo reagira na ključne događaje kao što su smrt, ozljede, napad ili skakanje. Kada se dogodi jedan od tih događaja, Animator prelazi na odgovarajuće stanje, aktivirajući potrebne animacije. Nakon što se specifična akcija završi, kao što je smrt ili napad, Bandit se vraća na Idle stanje kako bi nastavio s normalnim ponašanjem ili patroliranjem, ovisno o okolnostima u igri.

4.8.3. Šefovi

Šefovske borbe odavno su temelj dizajna videoigara, stvarajući neke od najupečatljivijih i najintenzivnijih trenutaka za igrače. Ovi sukobi često definiraju cijelo iskustvo igranja, potičući rasprave o tome koje šefovske borbe zauzimaju mjesto među najboljima svih vremena. Od epskih dvoboja do neočekivanih preokreta, svatko ima svoj popis omiljenih borbi, a rasprave o njima uvijek su strastvene i duboko subjektivne. Dizajniranje ovih sukoba zahtijeva ravnotežu između mehanike, narativnog utjecaja i angažmana igrača, uz osiguravanje da borba bude svježa i nezaboravna. Šefovske borbe često su mjesto gdje srž dizajna igre i kreativnost zaista dolaze do izražaja (Stout, 2010).

U „Void Born“, svaka zona sadrži jedinstvenog šefa koji igraču postavlja različite izazove. U mahovinskoj šumi, veliki koruptirani šišmiš uči igrača osnovama borbe, postavljajući prvi ozbiljni test. Špilja skriva vukodlaka koji neprestano napada i testira igračevu snalažljivost. U dvorcu, heroj brani prijelaz u drugu dimenziju, dok u koruptiranoj šumi zli čarobnjak može prizivati kosture i napadati iz daljine. Svaki od ovih šefova predstavlja poseban izazov i dodatno obogaćuje igračko iskustvo. Svaki od ovih šefova može biti poražen na različite načine i na igraču je da otkrije najefikasniju strategiju za svakoga od njih. Najteži izazov se nalazi na kraju gdje će se igrač morati suočiti sa prazninom i poraziti void knight-a u finalnoj areni u drugoj dimenziji gdje će se testirati sve što je igrač naučio.

4.9. Priča

Metroidvania igre ističu se ne samo kroz svoje istraživanje i igrivosti, već i kroz jedinstven način na koji pričaju svoje priče. Za razliku od drugih žanrova, ove igre često koriste okruženje, skrivenu predaju i samu dinamiku napredovanja kako bi oblikovale narativ. U Metroidvania igrama, priča često ostaje u pozadini, otkrivajući se postupno kroz istraživanje i interakciju s okolinom. Ova metoda pričanja priče obogaćuje iskustvo istraživanja, pretvarajući svako otkriće u važan dio veće slagalice.

Priča u igri „Void Born“ odvija se kroz interakcije s raznovrsnim sporednim likovima koje igrač susreće tijekom svoje pustolovine. Radnja igre započinje kada moćni kralj, opsjednut željom za neograničenom moći, priziva prazninu, tajanstvenu dimenziju ispunjenu hororom, u svoj svijet. Ovaj čin uzrokuje katastrofalan spoj dva svijeta – onog ljudskog i onog izvan realnog. Posljedice su dramatične: ljudski svijet je pogođen kaosom i razaranjem, stvorenja koja žive u tim mirnim područjima se mijenjaju i postaju agresivna, te neki ljudi jednostavno budu prebačeni u void bez svoje volje ili pristanka

Igrač se budi usred ove katastrofe i suočen je s posljedicama kraljeve odluke. Tijekom svoje avanture, igrač se oslanja na pomoć NPC-eva, koji kroz svoje priče i zadatke pružaju ključne informacije o tome što se dogodilo i kako se nositi s prijetnjama. Svaki NPC donosi jedinstvenu perspektivu, omogućujući igraču da složi sliku događaja koji su doveli do trenutne situacije, a istovremeno pruža tragove o tome kako se može suprotstaviti prijetnji i vratiti ravnotežu između dvaju svjetova.

4.10. Izrada sporednih likova

NPC (non-player character) su sporedni likovi koji služe kao važni elementi u priči igre. Slika 45 prikazuje plavog čarobnjaka, kojeg igrač može upoznati rano u igri. Ovaj sporedni lik je

kreiran unutar scene i opremljen ključnim komponentama: Sprite Renderer za prikaz slike, 2D Collider za interakciju, te skriptom koja upravlja njegovim ponašanjem. Skripta omogućava igraču interakciju s NPC-om i čitanje dijaloga kada se približi i vidi se na slici 46. Glavne funkcionalnosti uključuju prikaz dijaloške ploče, prikaz teksta u obliku dijaloga, te omogućavanje igraču da napreduje kroz dijaloge pritiskom na tipku.

Slika 43. Prikaz NPC plavog čarobnjaka

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.F) && playerIsClose)
    {
        if (dialoguePanel.activeInHierarchy)
        {
            ResetDialogue();
        }
        else
        {
            ShowDialoguePanel();
        }
    }
    if (dialogueText.text == dialogue[currentDialogueIndex])
    {
        contButton.SetActive(true);
    }
}
private void ResetDialogue()
{
    if (typingCoroutine != null)
    {
        StopCoroutine(typingCoroutine);
        typingCoroutine = null;
    }
    dialogueText.text = string.Empty;
    currentDialogueIndex = 0;
    dialoguePanel.SetActive(false);
}
private void ShowDialoguePanel()
{
    dialoguePanel.SetActive(true);
    typingCoroutine = StartCoroutine(TypeDialogue());
}
private IEnumerator TypeDialogue()
{
    if (isTyping) yield break;
    isTyping = true;
    dialogueText.text = string.Empty;
    foreach (char letter in dialogue[currentDialogueIndex])
    {
        dialogueText.text += letter;
        yield return new WaitForSeconds(wordSpeed);
    }
    isTyping = false;
}
```

Slika 44. Skripta za NPC - dijalog i interakciju

Kada igrač uđe u zonu interakcije s NPC-jem (detektirano pomoću metode OnTriggerEnter2D), postavlja se zastavica da je igrač blizu. Zatim, pritiskom na tipku F, ako je igrač u dometu NPC-a, otvara se dijaloška ploča i započinje ispis teksta. Tekst se prikazuje postupno, simulirajući efekt tipkanja pomoću korutine-a u metodi TypeDialogue(), gdje se slova postepeno dodaju svakih nekoliko milisekundi, kontrolirano varijablom wordSpeed. Ako je dijalog već aktivan, igrač može zatvoriti dijalog ili nastaviti do sljedeće linije koristeći tipku NextLine. Kad igrač napusti zonu NPC-a , dijaloška ploča se automatski zatvara, a dijalog se resetira, vraćajući sve na početnu točku. Time se osigurava da dijalog uvijek počinje od početka kada se igrač ponovno približi NPC-u. Skripta također uključuje gumb za nastavak (contButton) koji se pojavljuje nakon što je trenutna linija dijaloga potpuno ispisana, omogućujući igraču da napreduje kroz dijalog ili ga zatvori kada završi.

4.11. Grafičko korisničko sučelje

Korisničko sučelje (UI) u video igrama odnosi se na vizualne komponente koje igračima omogućuju interakciju s igrom, uključujući mehaniku, postavke i značajke. UI obuhvaća razne elemente poput izbornika, gumba, ikona, teksta, karata i drugih grafičkih prikaza na ekranu. Dizajn korisničkog sučelja igra ključnu ulogu u razvoju igara jer značajno utječe na ukupno korisničko iskustvo i igru. Dobro osmišljeno korisničko sučelje može poboljšati navigaciju, pojačati uranjanje u igru i povećati angažman igrača (polydin, 2023). Kvalitetan dizajn korisničkog sučelja od presudne važnosti za dizajnere i programere igara.

4.11.1. Glavni izbornik

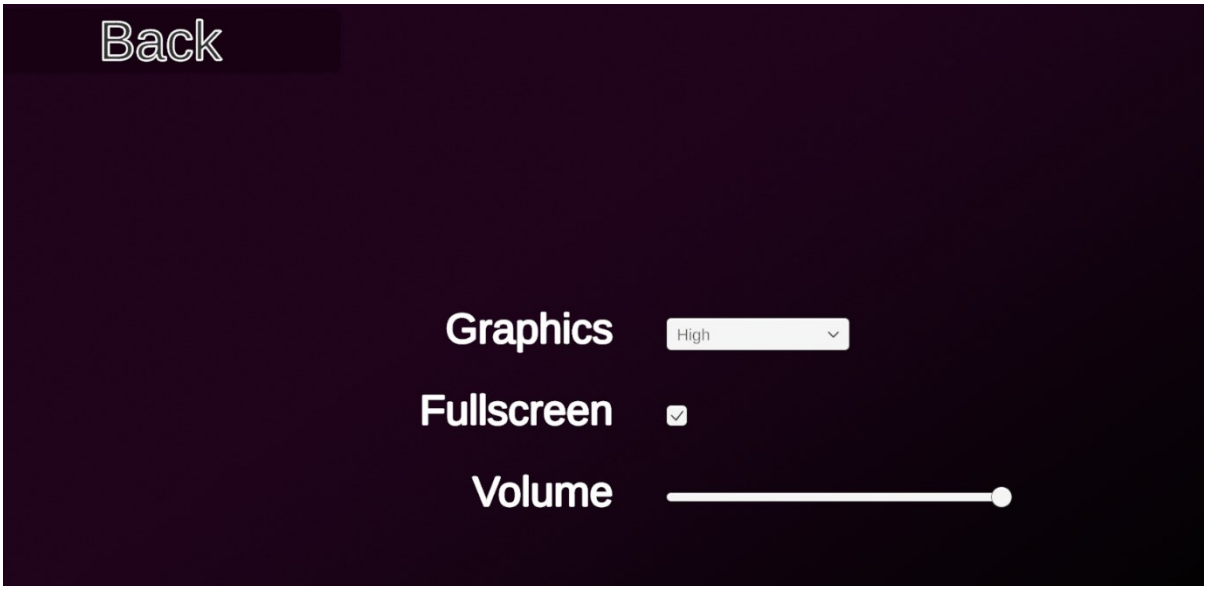
Izbornik igre, prikazan na slici 47, dizajniran je s ljubičastom pozadinom. Na vrhu izbornika nalazi se veliki naslov igre, ističući glavnu temu i tonu igre.

Slika 45. Prikaz glavnog izbornika

Izbornik sadrži četiri ključne opcije koje omogućuju igračima jednostavan pristup različitim funkcijama i postavkama igre:

1. Start - Ova opcija pokreće igru i vodi igrača na početak igre ili u najnoviji spremljeni napredak. Klikom na "Start", igrač ulazi u igru i započinje avanturu.

2. Options - Ova opcija omogućuje igračima da prilagode postavke igre prema svojim preferencijama, što je prikazano u slici 48. U "Options" izborniku, igrači mogu mijenjati grafičke i zvučne postavke i odabrati da li žele igrati preko cijeloga ekrana ili ne žele.

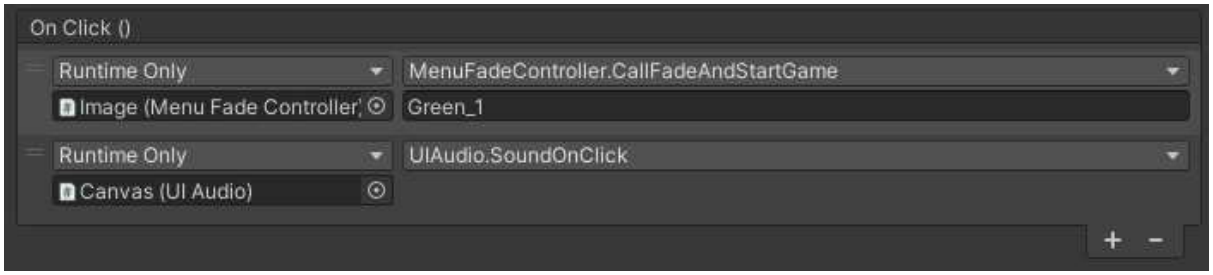


Slika 46. Prikaz opcija

3. Credits Ovdje igrač može vidjeti tko je napravio igricu. Klikom na "Credits", igrač će biti upućen na zaslon s informacijama o projektu.
4. Quit Ova opcija omogućuje igračima da izađu iz igre i vrate se na radnu površinu.

4.11.2. Mijenjanje izbornika i interakcija gumbova

U „Void Born“, mijenjanje izbornika pomoću gumba funkcionira putem interakcije korisnika s elementima korisničkog sučelja. Slika 49 prikazuje početak igre. Kada igrač klikne na određeni gumb, on pokreće specifičnu akciju koja je povezana s tim gumbom, kao što je prelazak na novi ekran, otvaranje podizbornika ili izvršavanje određene funkcije. U Unityju, ovi gumbi obično koriste UI elemente kao što su Button komponente, a logika za upravljanje tim gumbima može biti definirana putem skripti.



Slika 47. Funkcije za i prilagodba gubma

U ovom primjeru, početak igre upravlja se kroz dvije skripte. Prva skripta, Menu Fade Controller, odgovorna je za animaciju prelaska između izbornika i same igre, odnosno za vizualni efekt pri promjeni scena. Ova skripta koristi se kako bi se kontrolirao prijelaz iz glavnog izbornika u igru, stvarajući glatki fade efekt koji poboljšava korisničko iskustvo.

Kada igrač pritisne gumb "Start Game", Menu Fade Controller poziva funkciju koja inicira prelazak na odabranu razinu koja je definirana u izborniku što je prikazano u slici 50. Funkcija koja započinje igru može biti fleksibilna, omogućujući programeru da specificira bilo koji nivo kao početnu točku. Ova fleksibilnost znači da, umjesto da se uvijek pokrene isti nivo, igra može početi s različitim scenama, ovisno o uvjetima ili izborima igrača u izborniku. Na taj način, osim glatkog prijelaza, skripta omogućuje dinamično pokretanje igre s različitih točaka unutar igre.

```
Assets > Script > MenuFadeController.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuFadeController : MonoBehaviour
{
    private FadeUI fadeUI;
    [SerializeField] private float fadeTime;

    void Start()
    {
        fadeUI = GetComponent<FadeUI>();

        if (fadeUI == null)
        {
            Debug.LogError("FadeUI component not found on the GameObject. Please ensure it is attached.");
            return;
        }

        Debug.Log("FadeUI component found. Starting fade out.");
        fadeUI.FadeUIOut(fadeTime);
    }

    public void CallFadeAndStartGame(string sceneToLoad)
    {
        if (fadeUI == null)
        {
            Debug.LogError("FadeUI component not found. Please ensure it is assigned before calling this method.");
            return;
        }

        StartCoroutine(FadeAndStartGame(sceneToLoad));
    }

    private IEnumerator FadeAndStartGame(string sceneToLoad)
    {
        fadeUI.FadeUIOut(fadeTime);
        yield return new WaitForSecondsRealtime(fadeTime);
        SceneManager.LoadScene(sceneToLoad);
    }
```

Slika 48. Kod za početak igrice

Svaki gumb može biti povezan s funkcijom koja se izvršava kada igrač klikne na njega što se vidi na slici 51. Na primjer, u skripti možete koristiti funkciju poput OnClick() da registrirate kada je gumb pritisnut. Ovisno o funkciji gumba, skripta može mijenjati aktivne scene, pokazivati različite UI elemente ili mijenjati postavke igre. Pritisak na gumb "Start" pokreće igru i prebaci igrača s glavnog izbornika na scenu igre.

Slika 49. Funkcije za prilagođavanja gumba u opcijama

Izbornik s opcijama u igri ima drugačiju strukturu i funkcionira drugačije od glavnog izbornika. Kada igrač odluči ući u opcije (postavke), otvara se novi podizbornik u kojem može mijenjati različite postavke igre, poput glasnoće, grafike i prilagodbi veličine ekrana. Ovaj dio igre često nudi dublje mogućnosti podešavanja kako bi igrač mogao personalizirati svoje iskustvo. Kako bi se osigurala jednostavna navigacija, unutar izbornika s opcijama uvijek postoji mogućnost povratka na glavni izbornik. To se postiže putem back buttona (gumba za povratak), koji omogućava igraču da se jednostavno vrati na prethodni izbornik bez potrebe za ponovnim pokretanjem igre ili zatvaranjem aplikacije. Kada igrač pritisne gumb za povratak, vraća se na početni ekran izbornika, čime se osigurava neprekinuta i intuitivna navigacija između opcija i glavnog izbornika.

4.11.3 Kanvas

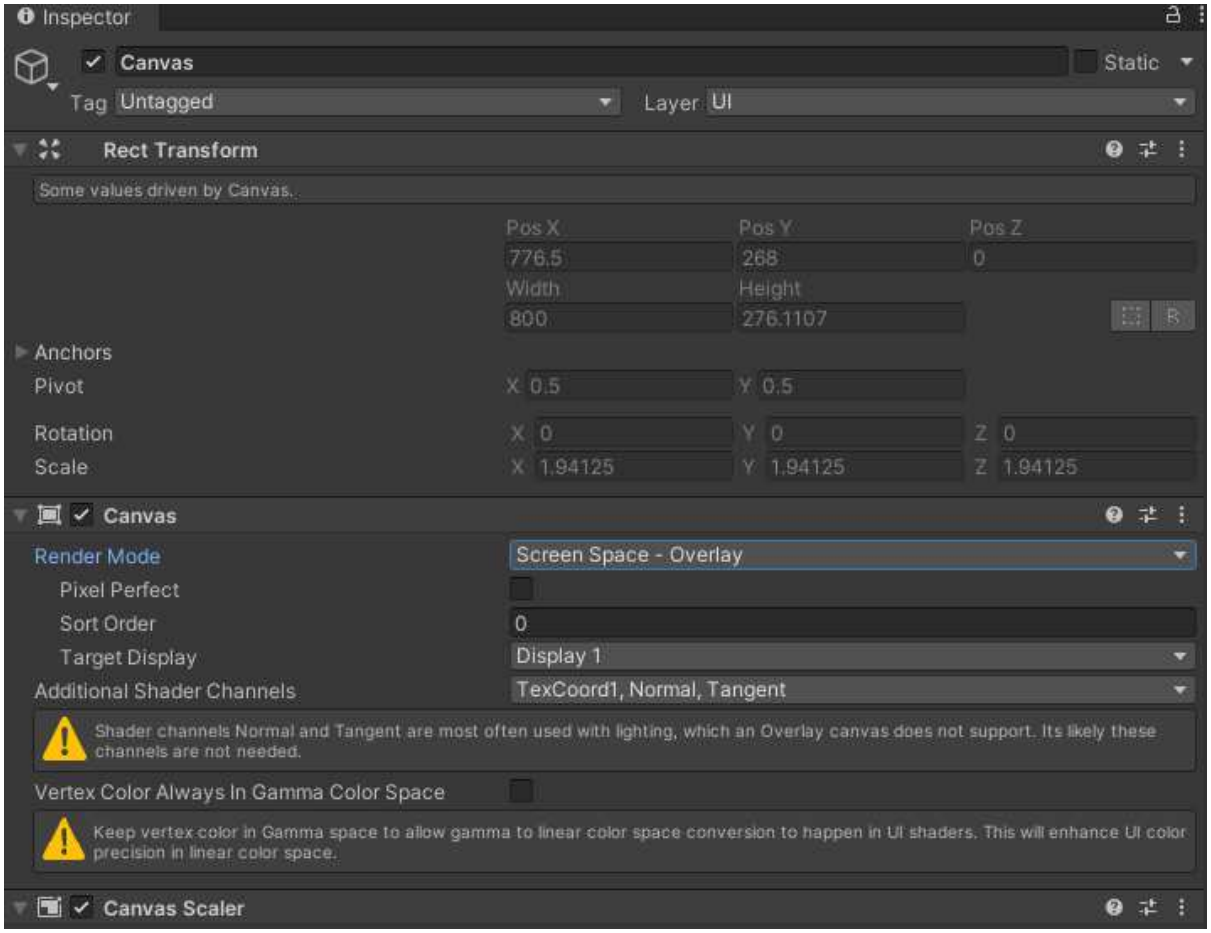
U Unityj-u, kanvas je ključni UI (korisnički sučelje) element koji služi kao kontejner za sve ostale UI komponente kao što su tekstovi, gumbi, slike, i drugi grafički elementi. On određuje kako će se ti elementi prikazivati na ekranu i kako će reagirati na promjene veličine ekrana ili različite rezolucije. Kanvas je virtualni prostor koji Unity koristi za crtanje UI elemenata u odnosu na ekran, a omogućuje i prikazivanje tih elemenata iznad same igre. Svaki UI element koji dodate mora biti unutar kanvasa kako bi bio vidljiv na ekranu što se vidi na slici 52. Postoji nekoliko načina na koje se kanvas može konfigurirati (Sergry, 2021):

Screen Space - Overlay: UI elementi su prikazani na vrhu igre, neovisno o kameri. Ova postavka koristi čitav ekran i svi UI elementi automatski se prilagođavaju rezoluciji ekrana.

Screen Space - Camera: UI elementi su prikazani u odnosu na poziciju kamere, omogućujući efekat dubine i interakciju s 3D elementima.

World Space: UI elementi postaju 3D objekti u sceni i mogu se postaviti bilo gdje u svijetu igre. To omogućuje stvaranje elemenata poput 3D tekstova ili interaktivnih elemenata unutar igre.

Kanvas također uključuje mehanizme za prilagođavanje veličine UI elemenata kako bi ostali proporcionalni bez obzira na rezoluciju ili omjer slike. (Datta, 20223)

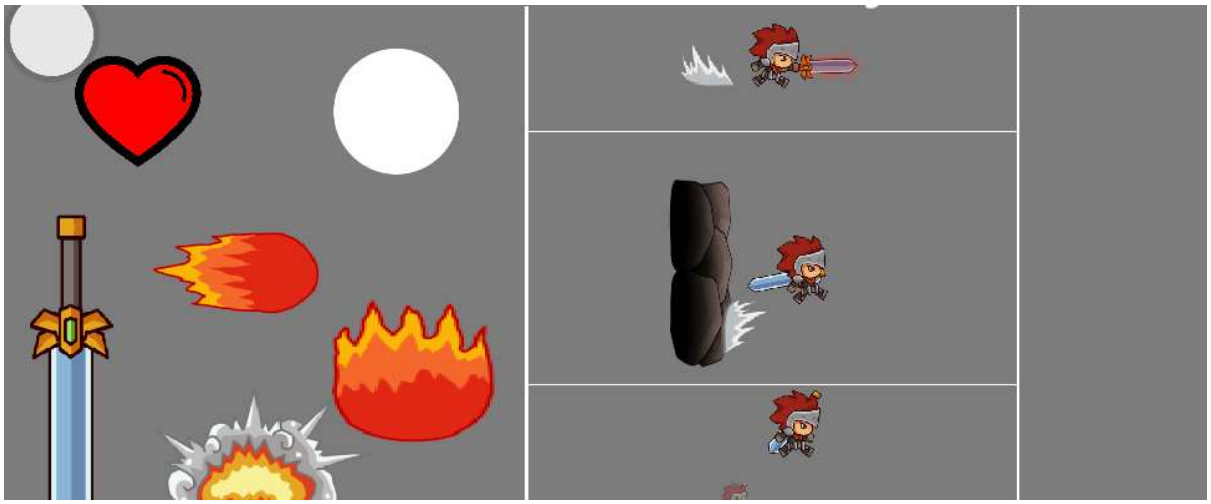


Slika 50. Prikaz kanvasa u inspektoru

4.11.4. Sučelje u igrici

Kanvas igra središnju ulogu kao srce korisničkog sučelja igre, pružajući sve potrebne vizualne elemente s kojima će igrač moći integrirati. Svaki aspekt igre koji je povezan s korisnikovim iskustvom prikazan je putem kanvasa, čineći ga ključnim dijelom igre koji upravlja načinom na koji igrači primaju informacije i donose odluke. Jedna od osnovnih funkcionalnosti kanvas je prikazivanje zdravlja igrača u obliku srca i mane, koja se koristi za obnavljanje zdravlja i izvođenje magičnih sposobnosti. Srce simbolizira preostale živote igrača, a kako igrač prima štetu, srca nestaju, dajući jasan vizualni indikator preostalog zdravlja. Mana omogućuje izvođenje posebnih napada i regeneraciju zdravlja, a njena razina se kontinuirano prati i prikazuje unutar UI-a. Oba ova elementa povezana su s Player Controller skriptom, koja upravlja ponašanjem igrača u igri. Zahvaljujući toj povezanosti, kanvas dinamički ažurira stanje srca i mane, omogućujući igraču da u svakom trenutku zna koliko resursa mu je preostalo i kada je blizu poraza.

Pored prikaza života i mane, Canvas također sadrži inventory sustav koji se otvara pritiskom na tipku "I" što se vidi na slici 53. Inventory omogućava igraču da pregleda sve magije i sposobnosti koje je pronašao i otključao tijekom igranja. Magije koje igrač otkrije pohranjuju se u inventory i mogu se pregledati ili aktivirati kad je potrebno. Ovo omogućuje igraču lak pristup svim resursima i sposobnostima koje je sakupio, pomažući mu da razvije vlastitu strategiju u borbi protiv neprijatelja.



Slika 51. Prikaz inventory-a u kanvasu

Konačno, Canvas također upravlja prijelazom između života i smrti unutar igre. Kada igrač izgubi sva srca, Canvas automatski aktivira fade efekt, zatamnjujući ekran, što jasno signalizira poraz igrača. Nakon toga, igrač se ponovno stvara u svijetu igre, omogućujući mu nastavak igre bez potrebe za ponovnim pokretanjem. Ovaj sustav pomaže u održavanju fluidnog igranja, bez predugih prekida, i omogućuje igraču da se brzo vrati u akciju.

4.11.5. Izgled trgovine

U „Void Born“-u, trgovina služi kao ključno mjesto gdje igrači mogu trošiti souls, valutu koju dobivaju porazom neprijatelja, kako bi kupili trajna unapređenja koja značajno poboljšavaju njihovu izvedbu u igri. Ova unapređenja igračima omogućuju napredak kroz igru i uspješnije

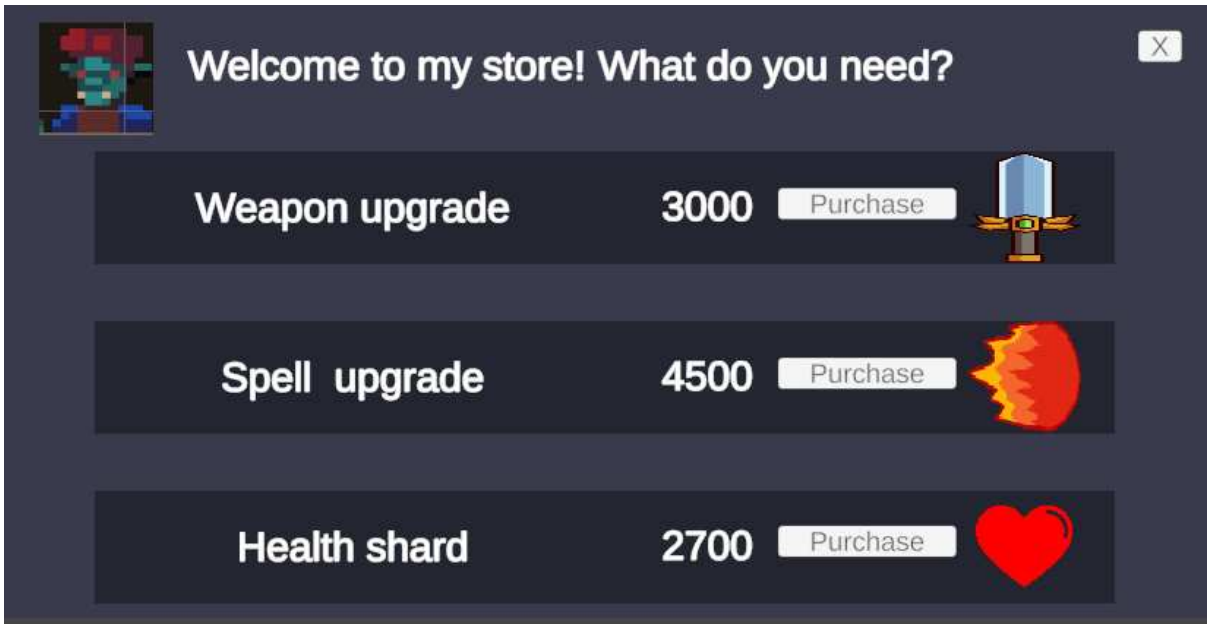
suočavanje s izazovima, pružajući im veću snagu i izdržljivost. Trgovina nudi tri glavna tipa unapređenja:

Unapređenje oružja – Kupnjom ovog unapređenja, igrač povećava štetu koju njegovo oružje nanosi neprijateljima, čineći borbu učinkovitijom. Cijena ovog unapređenja je 3000 souls.

Unapređenje magije – Ova opcija omogućava igračima da poboljšaju štetu svojih magijskih napada, povećavajući njihovu učinkovitost u borbi protiv snažnijih protivnika. Cijena za unapređenje spell-a je 4500 souls.

Health shard – Ovaj predmet povećava maksimalno zdravlje igrača, omogućujući mu da preživi dulje i pretrpi više štete prije nego što bude poražen. Cijena za health shard je 2700 souls.

Trgovina tako postaje ključna mehanika igre, omogućujući igračima da strateški troše svoje resurse kako bi se optimalno pripremili za buduće izazove i poboljšali svoje šanse za preživljavanje i uspjeh u borbi. Slika 54 prikazuje kako trgovina izgleda u igri.



Slika 52. Prikaz trgovine

U igri, sustav trgovine koji je implementiran pomoću dviju skripti koje, prikazane na slici 55 i 56, omogućuje igraču da kupi različita unapređenja kada se nalazi unutar trgovine. Trgovina koristi mehanizam okidača za prepoznavanje kada se igrač približi ili udalji od područja trgovine, dok korisničko sučelje trgovine (UI) omogućuje igraču da vrši kupnje i zatvara trgovinu.

Assets > Shop_Trigger.cs

```
using UnityEngine;

public class ShopTrigger : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D collider)
    {
        if (collider.CompareTag("Player"))
        {
            var uiShop = FindObjectOfType<UI_Shop>();
            if (uiShop != null)
            {
                uiShop.playerInRange = true;
            }
            else
            {
                Debug.LogWarning("UI_Shop instance not found in OnTriggerEnter2D.");
            }
        }
    }

    private void OnTriggerExit2D(Collider2D collider)
    {
        if (collider.CompareTag("Player"))
        {
            var uiShop = FindObjectOfType<UI_Shop>();
            if (uiShop != null)
            {
                uiShop.playerInRange = false;

                // Ensure the shop is closed if it's still open
                if (uiShop.gameObject.activeSelf)
                {
                    uiShop.CloseShop();
                }
            }
            else
            {
                Debug.LogWarning("UI_Shop instance not found in OnTriggerExit2D.");
            }
        }
    }
}
```

Slika 53. Opis skripte Shop_Trigger

Ova skripta koristi okidač područja da otkrije kada se igrač nalazi unutar područja trgovine. Kada igrač uđe u tu zonu, trgovina postaje dostupna za interakciju, a igrač može otvoriti sučelje trgovine pritiskom na tipku F. Ako igrač napusti područje trgovine, trgovina se automatski zatvara, a sučelje trgovine postaje nedostupno. Također, ako trgovina slučajno ostane otvorena kada igrač izađe iz zone, skripta osigurava da se trgovina zatvori kako bi se spriječile neželjene interakcije.

Skripta za korisničko sučelje trgovine upravlja svim aspektima vizualnog prikaza trgovine i kupovine predmeta. To je demonstrirano na slici 56. Kada igrač pritisne F dok je unutar trgovine, sučelje trgovine se otvara, prikazujući sve dostupne opcije kupnje. Trgovina nudi tri glavna unapređenja:

Assets > UI_Shop.cs

```
public class UI_Shop : MonoBehaviour
{
    private Transform Shop;
    private Transform ShopItemTemplate;
    public bool playerInRange;

    public int weaponUpgradeCost = 3000;
    public int spellDamageUpgradeCost = 4500;
    public int healthShardUpgradeCost = 2700;

    [SerializeField] private Button closeButton;

    private void Awake()
    {
        Debug.Log("UI_Shop script initialized.");
        Shop = transform.Find("Shop");
        ShopItemTemplate = Shop.Find("ShopItemTemplate");
        ShopItemTemplate.gameObject.SetActive(false);
        Shop.gameObject.SetActive(false);

        if (closeButton != null)
        {
            closeButton.onClick.AddListener(CloseShop);
        }
        else
        {
            Debug.LogWarning("Close button is not assigned in the inspector!");
        }
    }

    private void Update()
    {
        if (playerInRange && Input.GetKeyDown(KeyCode.F))
        {
            Debug.Log("F key pressed and player in range.");
            OpenShop();
        }

        if (Shop.gameObject.activeSelf && Input.GetKeyDown(KeyCode.Escape))
        {
            Debug.Log("ESC key pressed. Closing shop.");
            CloseShop();
        }
    }

    private void OpenShop()
    {
        Shop.gameObject.SetActive(true);
        ActivateAllChildren(Shop);
    }

    public void CloseShop()
    {
        Shop.gameObject.SetActive(false);
    }
```

Slika 54. Opis skripte UI_Shop

Unapređenje oružja,

Unapređenje magijskih napada (spells),

Health shard koji povećava zdravlje igrača.

Svaka od ovih opcija može se kupiti ako igrač ima dovoljno souls-a. Kupnja unapređenja oduzima određenu količinu souls iz igračevog salda, a unapređenje se primjenjuje na igrača –

povećava se šteta oružja ili magije, ili se povećava maksimalno zdravlje igrača. Ako igrač nema dovoljno souls, trgovina prikazuje poruku kako nema dovoljno sredstava za kupnju.

Sučelje trgovine također omogućava igraču da zatvori trgovinu pritiskom na ESC ili klikom na gumb Close, čime trgovina postaje neaktivna, ali je ponovno dostupna čim igrač ponovno uđe u trgovinsku zonu.

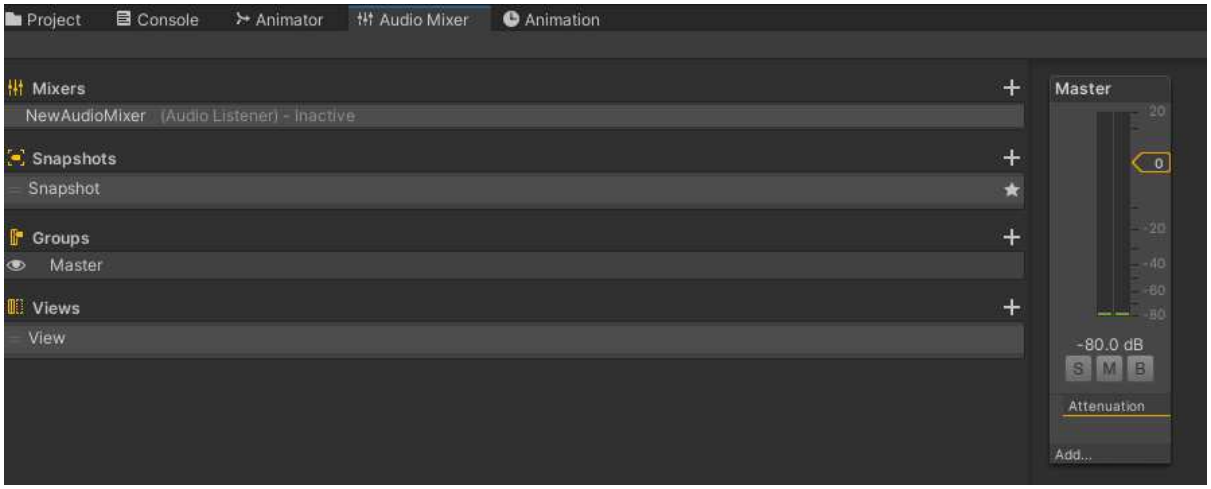
4.12. Audio

Zvuk igra ključnu ulogu u poboljšanju cjelokupnog iskustva igranja, a Unity je moćna platforma koja olakšava ugradnju zvuka u vaše projekte. Bilo da želite dodati pozadinsku glazbu, zvučne efekte ili glasovnu glumu, Unity ima sve što vam je potrebno za početak.

Unityjev audio sustav podržava širok raspon audio formata, uključujući AIFF, WAV, MP3 i OGG, tako da možete odabrati format koji najbolje odgovara vašim potrebama. Također možete koristiti ugrađeni audio mikser za prilagodbu glasnoće, visine i drugih svojstava audio materijala (Dursun, 2023).

4.12.1 Glavni zvukovni mikser

U izborniku igre, implementiran je glavni audio komponent koji upravlja distribucijom zvuka kroz cijelu igru. Ovaj sustav omogućuje da svi zvučni efekti, glazba i drugi zvučni elementi prolaze kroz centralizirani kontroler što prikazuje slika 57. To znači da kada igrač promijeni postavke zvuka u izborniku Options, te postavke automatski utječu na cijelu igru, bez potrebe za dodatnim prilagodbama u različitim dijelovima igre.



Slika 55. Prikaz zvuka

Promjene u glasnoći ili zvuku primjenjuju se globalno, pa igrači mogu, primjerice, smanjiti glazbu ili pojačati zvučne efekte kako bi stvorili iskustvo prilagođeno njihovim preferencijama.

Ova postavka pojednostavljuje upravljanje zvukom u igri i osigurava dosljednost bez obzira na to gdje se igrač nalazi u igri.

4.12.2 Skripta za zvuk

Skripta za zvuk se zove UIAudio i prikazana je na slici 58. Koristi se za upravljanje zvučnim efektima unutar korisničkog sučelja (UI) igre. Ona omogućuje reproduciranje različitih zvukova prilikom interakcije s UI elementima, kao što su zvukovi pri prelasku miša (hover) preko gumba ili klikom na gumbe. Također, skripta osigurava reprodukciju glavne glazbene teme izbornika i brine o njenom kontinuiranom ponavljanju dok je izbornik aktivan.

Skripta koristi AudioSource komponentu za upravljanje zvukovima. Kada igra započne, u metodi Start() provjerava se postoji li postavljena glazba za glavni izbornik (mainMenuMusic). Ako je prisutna, i ako je pridruženi AudioSource ispravno postavljen, glazba se automatski reproducira u petlji. Ova metoda omogućuje kontinuiranu pozadinsku glazbu dok igrač koristi izbornik, što poboljšava atmosferu i iskustvo unutar igre.

Dodatno, skripta nudi funkcije SoundOnHover() i SoundOnClick(), koje se pozivaju kada igrač prelazi mišem preko UI elemenata ili klikne na njih. Ove funkcije reproduciraju odgovarajuće zvučne efekte pomoću metode PlayOneShot(), koja omogućava trenutnu reprodukciju zvuka bez ometanja pozadinske glazbe. Skripta također osigurava pravilno zaustavljanje glazbe prilikom uništavanja objekta pomoću metode OnDestroy(), čime se osigurava da glazba ne nastavi svirati kada više nije potrebna.

Assets > Script > UIAudio.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class UIAudio : MonoBehaviour
{
    [SerializeField] private AudioClip hover;
    [SerializeField] private AudioClip click;
    [SerializeField] private AudioClip mainMenuMusic;

    private AudioSource audioSource;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();

        if (mainMenuMusic != null && audioSource != null)
        {
            audioSource.clip = mainMenuMusic;
            audioSource.loop = true;
            audioSource.Play();
        }
    }

    public void SoundOnHover()
    {
        if (hover != null && audioSource != null)
        {
            audioSource.PlayOneShot(hover);
        }
    }

    public void SoundOnClick()
    {
        if (click != null && audioSource != null)
        {
            audioSource.PlayOneShot(click);
        }
    }

    private void OnDestroy()
    {
        if (audioSource != null)
        {
            audioSource.Stop();
        }
    }
}
```

Slika 56. Skripta za zvuk

5. Zaključak

Zaključak ovog rada pruža sveobuhvatan prikaz ključnih koraka u razvoju igre „Void Born“ u Unity okruženju. Projekt je pokazao složenost razvoja 2D igara, s posebnim naglaskom na žanr metroidvanije, gdje je nužno uskladiti različite mehanike igre, uključujući borbene sustave, napredovanje igrača, interakciju s dinamičnom okolinom te narativne elemente.

Iako je postignuta zadovoljavajuća razina razvoja, ograničeni vremenski okvir i resursi utjecali su na opseg i složenost konačnog proizvoda. S daljnjim razvojem, igra bi se mogla dodatno unaprijediti uvođenjem novih okruženja, različitih tipova neprijatelja te implementacijom dodatnih mehanika i sustava, čime bi se obogatilo cjelokupno iskustvo igranja. Proširenja bi omogućila dublje istraživanje različitih aspekata igre, pružajući igračima dinamičnije i raznovrsnije iskustvo, uz veću razinu izazova i interakcije. Uz proširenje gameplaya, fokus na optimizaciju performansi i prilagodbu za različite platforme također bi mogao doprinijeti većoj dostupnosti i boljem iskustvu korisnika.

Korištenje alata kao što je Unity značajno je olakšalo proces razvoja ove igre. Bez sličnog alata, mnogi aspekti igre zahtijevali bi ručno programiranje od početka, što bi usporilo i otežalo razvojni proces. Na primjer, sustavi poput fizičke simulacije i upravljanja animacijama automatski su dostupni putem Unityja, što je omogućilo bržu i učinkovitiju implementaciju mehanika poput detekcije sudara i složenih prijelaza između animacija. Nadalje, sustav komponenti u Unityju omogućio je jednostavnu integraciju skripti s objektima u igri, čime je osigurana brza i fleksibilna prilagodba funkcionalnosti unutar igre. Također, korištenje Unity trgovine olakšalo je pristup resursima kao što su 2D spriteovi i zvukovi, što je dodatno ubrzalo proces razvoja i omogućilo veći fokus na dizajnerske aspekte igre.

Ovaj projekt pružio je dragocjeno iskustvo u razvoju 2D računalnih igara, pogotovo u kontekstu integracije različitih tehničkih i dizajnerskih aspekata unutar kompleksnog sustava. Iako postoje mogućnosti za daljnje proširenje igre, projekt predstavlja čvrstu osnovu za budući razvoj i dodatna unaprjeđenja koja bi igračima omogućila još bogatije i zadovoljavajuće iskustvo igranja. Iskustva stečena kroz ovaj projekt mogu poslužiti kao temelj za buduće projekte u industriji videoigara.

GitHub – https://github. com/M4T30-772/Voidborn

Literatura

Anderson, N. (14. november 2023). *The Popularity of Video Games: Entertainment and Interactive Storytelling*. (medium) Dohvaćeno iz https://medium.com/@noah.anderson_84365/the-popularity-of-video-games-entertainment-and-interactive-storytelling-0c1671365797

ansimuz. (1. January 2022). *itch.io*. (itch.io) Dohvaćeno iz https://ansimuz.itch.io/gothicvania-junk-wasteland-environment

brullov. (1. January 2018). *itch.io*. (itch.io) Dohvaćeno iz https://brullov.itch.io/2d-platformer-asset-pack-castle-of-despair

Burke, M. (18. December 2023). *In History: The first ever video game console, 50 years on*. (BBC) Dohvaćeno iz https://www.bbc.com/culture/article/20231213-in-history-the-first-ever-video-game-console-50-years-on

Cherry, T. (2017). *Hollow knight.* Team cherry.

Club, M. (12. April 2022). *How Video Games Are Made*. (Medium) Dohvaćeno iz https://medium.com/@microclub_usthb/how-video-games-are-made-608cacced8f

corwin-zx. (1. January 2022). *itch.io*. (itch.io) Dohvaćeno iz https://corwin-zx.itch.io/the-pale-moonlight

Datta, S. (25. January 20223). *Develop 2D Applications using Unity UI – Canvas and Scalability*. (cloud that) Dohvaćeno iz https://www.cloudthat.com/resources/blog/develop-2d-applications-using-unity-ui-canvas-and-scalability

Dursun, F. E. (1. Febuary 2023). *How to Optimize Audio in Unity: Best Practices Vol.1*. (Swishswoosh) Dohvaćeno iz https://www.swish-swoosh.com/blogs/in-tune/audio-in-unity?srsltid=AfmBOopeV3Qz9T2cPWEKZSH-wjSpott2AzXuuIuzm3W5JnQ1Pnpq_RcW

editors, H. (31. August 2017). *Video Game History*. (HISTORY) Dohvaćeno iz https://www.history.com/topics/inventions/history-of-video-games

Games, R. S. (6. October 2023). *What Are Assets in Game Design?* (Retro Style Games) Dohvaćeno iz https://retrostylegames.com/blog/what-are-assets-in-game-design/

Giri, P. (8. April 2022). *Unity's Sprite Settings Explained*. (yarsalabs) Dohvaćeno iz https://blog.yarsalabs.com/unity-sprite-settings-full-guide/

Halpern, J. (11. December 2018). *The What and Why of Game Engines*. (medium) Dohvaćeno iz https://medium.com/@jaredehalpern/the-what-and-why-of-game-engines-f2b89a46d01f

Howarth, J. (11. June 2024). *https://explodingtopics.com/blog/number-of-gamers*. Dohvaćeno iz https://explodingtopics.com/: https://explodingtopics.com/blog/number-of-gamers

Julia. (19. January 2019). *Game Enemy Design Starter Guide*. (GameDevFriends) Dohvaćeno iz https://gamedevfriends.com/enemy-design-starter-guide/

Labz, V. (28. August 2023). *How To Make Smooth Transition Between Scenes In Unity.* (Linkedin) Dohvaćeno iz https://www.linkedin.com/pulse/how-make-smooth-transition-between-scenes-unity-vector-labz-fze-llc

Low, A. (1. April 2024). *What is Sprite?* (codeandweb) Dohvaćeno iz https://www.codeandweb.com/knowledgebase/what-is-a-sprite

Luke, C. (1. March 2024). *Castlevania games in order: Story and release orders explained*. (RadioTimes.com) Dohvaćeno iz https://www.radiotimes.com/technology/gaming/castlevania-games-in-order/

Maaot. (1. January 2019). *Itch.io*. (Itch.io) Dohvaćeno iz https://maaot.itch.io/mossy-cavern

Maaot. (1. January 2020). *itch.io*. (itch.io) Dohvaćeno iz https://maaot.itch.io/2d-browncave-assets?ac=txbHXzLDXvG

Ng, K. (6. September 2017). *Knight Sprite Sheet*. (Asset Store) Dohvaćeno iz https://assetstore.unity.com/packages/2d/characters/knight-sprite-sheet-free-93897

Pekar, M. (8. September 2022). *Building a third-person controller in Unity with the new input system*. (LogRocket) Dohvaćeno iz https://blog.logrocket.com/building-third-person-controller-unity-new-input-system/

play, t. -n. (1. January 2007). *museumofplay*. (thestrong) Dohvaćeno iz https://www.museumofplay.org/toys/atari-2600-game-system/

polydin. (27. April 2023). *What is UI in Games? | All about User Interface*. (polydin) Dohvaćeno iz https://polydin.com/ui-in-games/

Sergry. (4. April 2021). *Unity meets Canvas*. (stfalcon) Dohvaćeno iz https://stfalcon.com/uk/blog/post/unity-meets-canvas

Shahbazi, N. (27. June 2024). *What Are the Different Kinds of Video Game Genres?* (Pixune) Dohvaćeno iz https://pixune.com/blog/video-game-genres/

Stegner, B. (19. October 2021). *What Are Metroidvania Video Games?* (makeuseof) Dohvaćeno iz https://www.makeuseof.com/what-are-metroidvania-video-games/

Stout, M. (15. September 2010). *Game Developer*. (Boss Battle Design and Structure) Dohvaćeno iz https://www.gamedeveloper.com/design/boss-battle-design-and-structure

Tyrell, V. (11. November 2020). *Ori and the Will of the Wisps*. (IGN) Dohvaćeno iz https://adria.ign.com/ori-dva

Unity. (4. March 2022). *Animator*. (Unity) Dohvaćeno iz https://docs.unity3d.com/Manual/class-AnimatorController.html

Unity. (1. March 2024). *Unity*. (Unity) Dohvaćeno iz https://unity.com/download

Unity. (1. January 2024). *Unity Documentation*. (Unity) Dohvaćeno iz https://docs.unity.com/

Popis slika