Izrada vertikalnog platformera u programskom alatu Unity

Horvat, Mario

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:300168

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mario Horvat

IZRADA 2D VERTIKALNOG PLATFORMERA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mario Horvat

Matični broj: 0016149278

Studij: Informacijski i poslovni sustavi – Umreženi sustavi i računalne igre

IZRADA 2D VERTIKALNOG PLATFORMERA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. Sc. Mladen Konecki

Varaždin, studeni 2024.

Mario Horvat

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog završnog rada je izrada 2D vertikalnog platformera u programskom alatu Unity. Pisani dio se sastoji od opisa korištenih alata konkretno Unity te Visual Studio 2022, te opisa same igre. Nakon toga je opisana izrada igre, što uključuje mehanike, animacije, zvukove i sve korišteno u izradi. Kreiranje samog svjeta u kojem se igra odvija, svih likova i njihovih animacija skakanja, pucanja, hodanja, te svih sustava korištenih u logičkom dijelu igre. Glavni lik je igrač koji prelazi razinu, na putu susreće neprijatelje, te mora doći do kraja razine bez da izgubi sve životne bodove, na svom putu mora pokupiti određene napitke koji mu povećavaju snagu skoka, kako bi mogao dosegnuti određene dijelove razine. Igrač na svom putu također mora skupljati novčiće, te ukoliko je prikupio dovoljan broj mu se otvara mogućnost prolaska kroz portal, gdje ga nakon toga čeka glavni neprijatelj kojeg kada porazi je igra gotova. Igrač je uspješno završio igricu ako je ubio glavnog neprijatelja, a izgubio ako su mu životni bodovi dostigli 0. Praktički dio završnog rada je izrada prototipa igre u programskom alatu Unity.

Ključne riječi: Unity, 2D platformer, vertikalni platformer, mehanike, igrač , neprijatelj, životni bodovi, razine, projektil

Sadržaj

1. Uvod

Videoigre danas su jedna od najpopularnijih formi zabave, s tržištem koje se rapidno razvija i raste iz godine u godinu. S napretkom tehnologije napreduju i mogućnosti za kreiranje razlonikih žanrova igara. Platformerske igre, posebice 2D platformeri su jedan od najstarijih žanrova te kao takvi su među najprepoznatijim, a mnogi su odrasli uz 2D platformer klasike kao što su Super Mario ili Sonic the Hedgehog.

Razvoj videoigara nije samo zabava već i jedan složen proces koji zahtjeva solidno poznavanje programskih jezika, kao i alata za kreiranje grafike i zvuka, te alata i same logike igre. Kroz razvoj vlastitog prototipa 2D platformera imao sam priliku praktično primjeniti znanja stečena tokom studija, koristeći moderne alate poput Unity-a i Vusial Studio 2022.

Ovaj završni rad ima za cilj prikazati kako je izrađen 2D vertikalni platformer, od inicijalnog planiranja, preko tehničke implementacije mehanika do završne verzije igre. Motivacija za odabir ove teme proizašla je iz mog interesa za videoigre i želje da stvorim nešto svoje. Vlastiti projekt koji će obuhvatiti sve faze razvoja videoigre. Od ideje do funkcionalnog prototipa.

U radu će biti objašnjeni korišteni alati, tehnike programiranja, animacije i zvučni efekti kao i proces kreiranja razina i neprijatelja u igri. Na kraju, ovaj rad predstavlja rezultat dugotrajnog učenja i praktičnog rada, a izrada igre dala mi je uvid u složenost i kreativnost koja stoji iza razvoja ovakvih a i puno većih i opsežnijih projekata videoigara.

2. Metode i tehnike rada

Proces izrade ovog projekta uključivao je temeljito istraživanje tržišta videoigara koje dijele sličan žanr ili mehanike igranja sa onima koje sam ja zamislio implementirati. Primarna platforma za istraživanje bila mi je YouTube, budući da postoji mnogo developera koji detaljno prikazuju kako određene mehanike igre i kodovi funkcioniraju. Ova platforma omogućila mi je da razumijem osnovne koncepte i ideje koje su mi pomogle prilikom razvoja vlastite igre. Pregledom videozapisa o sličnim projektima stekao sam uvid u najbolje prakse za razvoj sličnih žanrova igara.

Za sami razvoj igre korišten je Unity [1], koji je sveobuhvatan alat za izradu igara, omogućuje integraciju različitih fizičkih komponenti, animacija i UI elemenata. Za implementaciju koda i logike igre koristio sam Microsoft Visual Studio 2022 [2], integrirano razvojno okruženje (IDE) koje se pokazalo iznimno korisnim za pisanje i debugiranje C# skripti. Osim alata za razvoj koda i mehanike, izrada vizualnih elemenata također je igrala ključnu ulogu u stvaranju igre. Photopea [3], online alat za uređivanje slika, korišten je za popravak i prilagodbu sprite-ova, koji su činili osnovne vizualne elemente igre. Ovaj alat, koji je jednostavan za upotrebu i podržava različite formate slika, omogućio je lako uređivanje postojećih resursa preuzetih s različitih platformi kao što su Unity Asset Store [4], Itch.io [5]

2.1. Unity

Unity je jedan od najpopularnijih alata za razvoj video igara, poznat po svojoj svestranosti i moćnim mogućnostima za izradu 2D i 3D igara. Razvijen od strane Unity Technologies, prvi put je predstavljen 2005. godine, a s vremenom je postao ključna platforma za stvaranje igara zbog svoje jednostavnosti i pristupačnosti. Njegova glavna prednost je što omogućuje razvoj igara za različite platforme, uključujući mobilne uređaje, računala i konzole.

Ono što Unity čini posebno privlačnim jest njegov bogat ekosustav alata (eng. tools) i resursa (eng assets). Unity Asset Store [4] pruža pristup velikom broju vizualnih i zvučnih resursa (eng. assets), koji se mogu jednostavno integrirati u projekte. To značajno ubrzava proces izrade igara, jer se resursi poput grafičkih elemenata, zvučnih efekata i predložaka mogu preuzeti i koristiti bez potrebe za vlastitom izradom. Ovo mi je znatno olakšalo izradu projekta jer sam i sam koristio potpuno besplatne resurse u ovaj projekt.

Unity također podržava različite tehnologije poput proširene (AR) i virtualne stvarnosti (VR). Razvijanje AR i VR aplikacija unutar Unity-ja omogućeno je putem specijaliziranih paketa, koji pružaju podršku za razne uređaje, uključujući VR headsete i mobilne telefone.

Jedna od ključnih prednosti Unityja je jednostavna struktura rada. Projekti su podijeljeni u scene (eng. Scenes), gdje se svaki objekt unutar scene može pojedinačno manipulirati. Na ovaj način, razvojne aktivnosti su jasno organizirane i olakšavaju implementaciju novih funkcionalnosti. Unity Scripting API nudi direktan pristup različitim funkcijama i metodama, što omogućuje programerima da kreiraju složene mehanike bez potrebe za dubljim poznavanjem tehničkih detalja.

Uz to, Unity podržava rad na više platformi, što znači da se igra razvijena u ovom alatu može izvoziti na različite operativne sustave i uređaje bez značajnih izmjena u kodu. Ova sposobnost minimalizira potrebne prilagodbe između verzija igre na različitim platformama, što uvelike olakšava proces distribucije i testiranja igre.

Jedan od glavnih razloga zbog kojeg je Unity postao izbor mnogih programera (eng. developera) je njegova sposobnost da podrži različite stilove grafike i različite platforme. Kao alat za razvoj igara, Unity nudi robusne alate za rad s 2D i 3D grafikom, uključujući napredne alate za animaciju, prilagodbu osvjetljenja i stvaranje složenih shader efekata putem Shader Grapha. U kontekstu ovog rada, koristio sam Unity za izradu 2D vertikalnog platformera, što uključuje rad s jednostavnim 2D grafičkim elementima i implementaciju različitih mehanika igre.

U konačnici, Unity se pokazao kao iznimno moćan alat za izradu mojeg 2D vertikalnog platformera, nudeći sve što je potrebno za stvaranje dinamične i vizualno privlačne igre uz podršku za različite platforme i jednostavno upravljanje resursima.

2.2. Visual Studio 2022

Visual Studio 2022 je napredna integrirana razvojna okolina (IDE) koja se koristi za razvoj aplikacija i igara, a u mom slučaju, za razvoj igre u Unityju. Ovaj alat razvijen od strane Microsoft-a nudi širok spektar funkcionalnosti koje olakšavaju pisanje, testiranje i debugiranje koda.

Jedna od glavnih prednosti Visual Studio 2022 je njegova podrška za različite programske jezike i okvire. U mom slučaju, koristio sam C# za razvoj skripti unutar Unityja. Visual Studio 2022 nudi bogat set alata za rad s C#, uključujući IntelliSense, koji pruža automatske prijedloge za kod, funkcije, varijable i klase, čime se znatno ubrzava proces

pisanja koda i smanjuje mogućnost grešaka. IntelliSense također pomaže u otkrivanju grešaka u kodu i predlaže ispravke, što je bilo izuzetno korisno pri pisanju složenih skripti za igru. Na primjer u nekim slučajevima mi je znatno ubrzao pisanje koda kada sam deklarirao više istih varijabli, IntelliSense sam predloži većinu toga što mi je znatno pomoglo pri pisanju skripti.

Osim IntelliSense-a, Visual Studio 2022 nudi napredne mogućnosti za debugiranje. Ove funkcionalnosti uključuju praćenje izvršenja koda, postavljanje točaka prekida, praćenje varijabli i inspekciju objekata u stvarnom vremenu.

Visual Studio 2022 također podržava rad s Git repozitorijima direktno iz IDE-a. Ovo omogućava lakšu kontrolu verzija i kolaboraciju, što je važno za održavanje organiziranog i učinkovitog razvojnog procesa. Povezivanje s Git-om važno je zbog praćenja promjena u kodu, jednostavno vraćanje na prethodne verzije i pregled promjena, što bi bilo korisno za praćenje napretka i povratak na radne verzije u slučaju potrebe. U mom slučaju ovu opciju nisam morao koristiti jer nije bilo potrebe, no vidim važnost u ovome te mislim da idući puta kada budem razvijao igricu ću sigurno prevagnuti na korištenje Git-a.

Za potrebe ovog završnog rada, Visual Studio 2022 je bio neprocjenjiv alat koji mi je omogućio efikasno pisanje i upravljanje kodom, testiranje funkcionalnosti i rješavanje problema. Svojim naprednim mogućnostima i integracijama, ovaj IDE je značajno doprinio uspješnom razvoju mog 2D vertikalnog platformera u Unityju.

2.3. Unity Asset Store

Unity Asset Store[3] je službena online platforma unutar Unity ekosustava koja programerima i dizajnerima omogućuje preuzimanje različitih resursa (engl. assets) koji se mogu koristiti pri razvoju videoigara i drugih interaktivnih sadržaja.Trgovina je pokrenuta 2010. godine, postala je jedno od najvažnijih središta za pristup gotovim resursima i alatima za Unity developere. Asset Store sadrži širok raspon sadržaja poput 3D modela, 2D spriteova, zvučnih efekata, glazbe, skripti, alata za animaciju, tekstura i shader-a te mnogih drugih korisnih alata za razvoj igara. Velika prednost kod Unity Asset Storea je što omogućava razvojnim programerima, posebno onima s ograničenim resursima ili znanjem, da pristupe profesionalnim materijalima bez potrebe da ih sami izrađuju. Ovo značajno smanjuje vrijeme potrebno za izradu igre, jer programeri mogu preuzeti već gotove materijale i prilagoditi ih potrebama svog projekta. Na primjer, 3D modeli i animacije likova, pozadine, UI elementi i čestice mogu se preuzeti i lako integrirati u igru.

Asset Store također omogućava zajednici developera da dijele svoje resurse, bilo besplatno ili uz naknadu. To stvara dinamičnu zajednicu u kojoj programeri mogu međusobno surađivati, dijeliti resurse i podržavati jedni druge kroz razvojne procese. Mnogi besplatni resursi su kvalitetni, a programeri mogu kupiti i komercijalne resurse koje su izradili profesionalci .

Osim resursa, Asset Store nudi i različite alate za optimizaciju igara, poput skripti za upravljanje performansama, sustava za čuvanje podataka, naprednih shadera te alata za proširenu i virtualnu stvarnost (AR/VR). Ovi alati pomažu programerima da poboljšaju učinkovitost i vizualnu privlačnost svojih igara, bez potrebe za detaljnim tehničkim znanjem o svakom aspektu razvoja .

2.4. Photopea

Photopea[4] je napredni alat za uređivanje slika koji omogućuje korisnicima rad s različitim vrstama grafičkih datoteka, uključujući PSD, JPEG, PNG, i mnoge druge. Ovaj alat je posebno koristan za razvoj video igara zbog svoje sposobnosti da kreira i prilagođava sprite-ove, koji su ključni za vizualni dio igre. U velikoj mjeri mi je pomogao sa prilagodbom određenih sprite-ova, te sam tako na kraju i implementirao svoj sprite u igru.

Jedna od glavnih prednosti Photopea-e je njegova sličnost s Adobe Photoshopom, što omogućava korisnicima koji su već upoznati s Photoshopom da se lako prilagode ovom besplatnom alatu. S obzirom na to da sam radio s velikim brojem sprite-ova za svoju igru, Photopea mi je omogućio detaljnu kontrolu nad svakim aspektom grafike.

Isto tako bitno je naglasiti da je najveća prednost ovog alata što je besplatan te kao takav bio je idealan odabir za moj projekt.

3. Razrada teme

Kroz ovo poglavlje objasnit ću nešto više o samom žanru platformera, nadalje ću se pozabaviti sučeljem programskog alata korištenog za izradu ovog projekta, te na posljetku ću objasniti sve u vezi projekta što uključuje scene, likove, animacije, skripte te dodatne materijale koji poboljšavaju iskustvo igranja.

3.1. Opis žanra

Platformer je žanr videoigara u kojem igrač upravlja likom kroz razine koje obuhvaćaju skakanje između platformi, izbjegavanje prepreka i poraz neprijatelja. Osnovne mehanike platformera uključuju trčanje, skakanje i istraživanje, a cilj igre obično je dosegnuti kraj razine ili ostvariti specifične zadatke kao što su prikupljanje predmeta ili poraz neprijatelja.

Žanr platformera seže u rane 1980-te, kada su igre poput Donkey Kong popularizirale koncept skakanja između platformi, a igre su bile prikazane na jednom ekranu. Razvojem tehnologije došlo je do uvođenja „scrollinga“, gdje se okolina pomiče dok se igrač kreće, omogućujući dinamičnije i složenije okruženje za istraživanje.

Platformeri mogu biti dvodimenzionalni (2D) ili trodimenzionalni (3D), no kako je tema ovog rada 2D platformer mi ćemo se fokusirati na dvodimenzionalnost. U 2D platformerima, igrači kontroliraju lika iz bočne perspektive, s ograničenim kretanjem unutar dvije dimenzije – lijevo-desno i gore-dolje. Ključna mehanika ovog žanra jest skakanje, koje igra glavnu ulogu u izbjegavanju prepreka i prelasku razina. Razine su dizajnirane s izazovima u vidu rupa, neprijatelja i zagonetki koje igrač mora savladati kako bi napredovao. Za razliku od 3D platformera, gdje igrači imaju slobodu kretanja u tri osi, 2D platformeri obično ograničavaju kretanje na horizontalnu i vertikalnu dimenziju. To donosi jednostavnije kontrole, ali i veći naglasak na preciznost skokova i manevriranja kroz izazovne dijelove razine. Usprkos jednostavnijem dizajnu, 2D platformeri mogu biti vrlo izazovni, zahtijevajući od igrača brzo razmišljanje, dobre reflekse i koordinaciju.

3.1.1. Podjela platformera

Platformeri se mogu podijeliti prema tematici i mehanikama. Tematske podjele uključuju platformere s rješavanjem zagonetki, akcijske platformere, avanturističke platformere i trči-pucaj platformere. Platformeri s rješavanjem zagonetki često kombiniraju elemente

logičkih igara s tradicionalnim skakanjem i istraživanjem, dok akcijski platformeri naglašavaju borbu s neprijateljima i dinamične razine.

Platformeri se također dijele prema broju dimenzija. 2D platformeri, poput onih na kojima se temelji ovaj rad, koriste ravnu perspektivu s fiksnim gledištem, dok 3D platformeri omogućuju istraživanje u sve tri dimenzije. Unutar 2D prostora, igrači imaju ograničene mogućnosti kretanja, no to donosi specifične izazove kao što su precizno tempiranje skokova i izbjegavanje prepreka.

3.1.2. Prednosti i nedostaci 2D platformera

2D platformeri donose nekoliko ključnih prednosti. Razvoj 2D igara je obično brži i zahtijeva manje tehničkih resursa, zbog čega su popularni na mobilnim platformama i za manje timove. Također, kontrole su jednostavnije, što omogućava većem broju igrača da lako uđe u igru. No, ograničenje 2D prostora donosi i određene nedostatke – svijet igre je ravniji, a mehanike često jednostavnije u usporedbi s 3D platformerima.

Unatoč tome, mnogi 2D platformeri stekli su kultni status zbog svoje jednostavne, ali duboko izazovne mehanike. Klasici poput Super Mario Bros., Sonic the Hedgehog, ali i noviji naslovi poput Celeste pokazali su da jednostavna prezentacija može biti jednako izazovna i nagrađujuća kao i suvremeni 3D naslovi.

4. Koncept igre

U ovom poglavlju ću opisati koncept igre kako sam ju ja zamislio, koje funkcije i animacije sam zamislio.

Na početku treba naglasiti da igra prati glavnog lika kroz put prema vrhu razine. Igra je zamišljena tako da je primarna funkcionalnost skakanje, te tako igrač mora na putu pokupiti Napitak koji mu daje jačinu skoka kako bi došao do određenog dijela razine. Isto tako igrač mora pronaći sve Novčiće koji su postavljeni po razini kako bi mu se otvorio portal koji ga vodi do glavnog neprijatelja kojeg on mora poraziti da bi prešao igricu. Igrač se može kretati lijevo, desno i skakati. Držati se za zid te će tako lagano kliziti prema dolje, isto tako dok je na zidu može skočiti. Igrač može pucati projektile pritiskom na lijevu tipku miša. Na putu do vrha susreće neprijatelje koji se kreću te kada vide igrača počinju ga pucati projektilima. Naravno u igru je implementiran sustav životnih bodova pa kada igrač izgubi sve bodove otvara mu se Pause Menu te iz tog menija ima mogućnost vratiti se na početak razine ili izaći iz igre. Svi neprijatelji imaju 3 životna boda isto tako i glavni igrač ima 3 životna boda, dok glavni neprijatelj ima 10 životnih bodova i povećanu brzinu projektila.

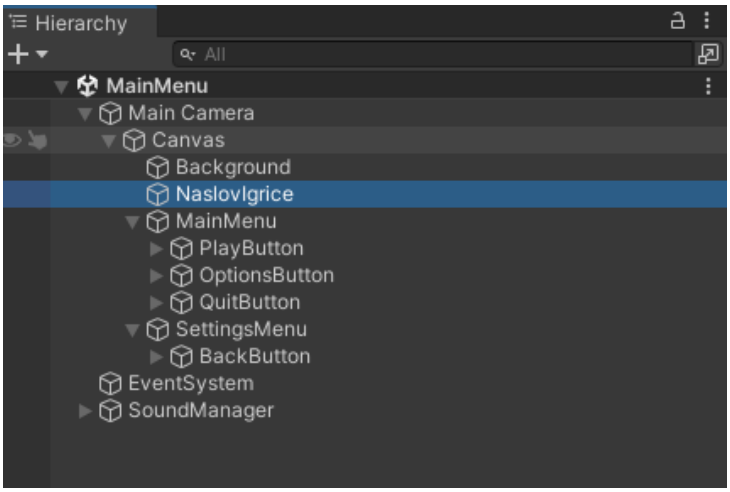
5. Kreiranje scena

5.1. Početna scena

Početna scena sastoji se od objekta Canvas u kojemu su implementirani dugmići (eng. buttons) te klikom na gumb Play igrač pokreće igru. Na vrhu scene je Naslov igrice „Jumpy Land“, te ispod 3 gumbića. Play, Settings, te Quit, trenutno se ovdje preklapaju gumbi Back i Quit iz razloga što je ovo Scene View, no kada se pokrene Game View gumbi se ne preklapaju. To se događa radi toga što je implementirana opcija Settings koja kada se klikne maknu se svi gumbi i pojavi se gumb Back. U ovoj sceni je zamišljeno da budu postavke igre kao što je npr. glasnoća muzike.

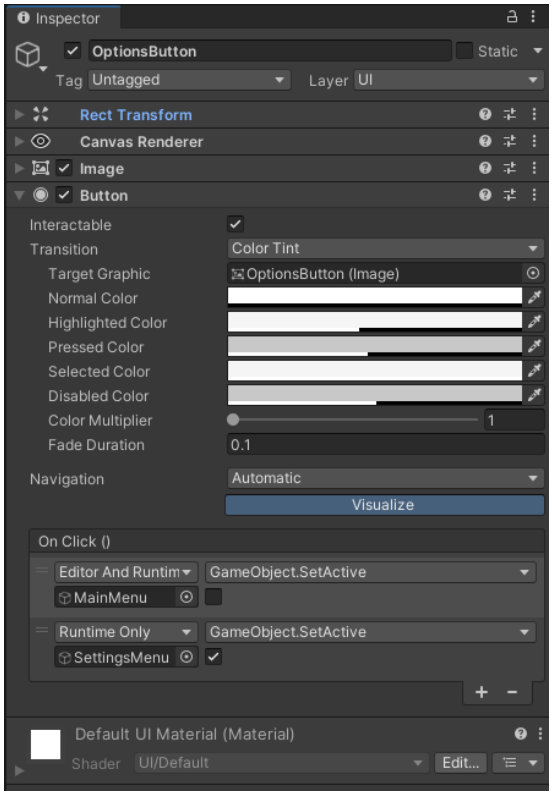
Slika 1 Početna scena igre

Na slici [2] je prikazana hijerarhija scene na kojoj se jasno vidi da objekt MainMenu ima 3 gumba Play, Options koji se u igri prikazuje kao Settings i Quit. A objekt SettingsMenu ima samo jedan gumb Back koji kada se klikne vraća se na početnu scenu.



Slika 2 Hijerarhija početne scene

Nadalje bitno je objasniti kako gumbi rade, svaki gumb ima komponentu Button koja ima funkciju On Click(), u koju se postavi objekt na kojemu se nalaze gumbi te se može manipulirati što će se desiti kada se klikne gumb. Na slici [3] vidljivo je kako SettingsButton na funkciji On Click() ima postavljena 2 objekta MainMenu i SettingsMenu. Te opciju GameObject.SetActive i prozorčić koji kada se klikne omogućuje se prikaz te scene.

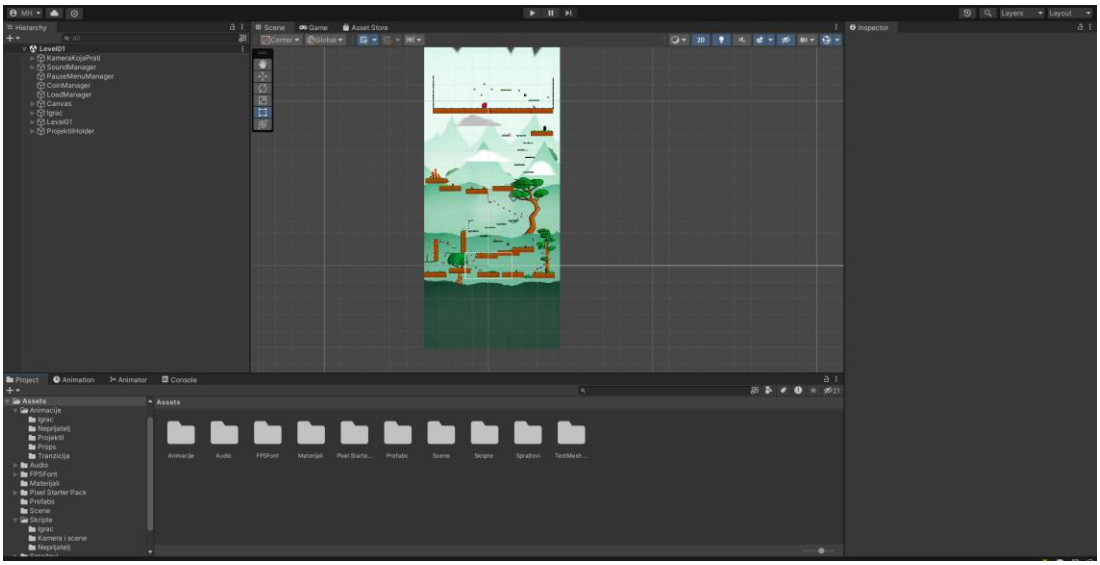


Slika 3 Inspector Settings gumba

Isto tako kako je postavljen ovaj gumb postavljeni su i ostali no to ćemo više prokomentirati kada dođemo do skripte koja upravlja ovim dijelom igre.

5.2. Scena razine

Scena Level01 je ključna scena u ovom projektu, koja predstavlja jednu razinu igre. U ovoj sceni odvija se glavna radnja igre, uključujući sve elemente koji čine razinu funkcionalnom i zanimljivom za igrače. Na Level01 sceni su implementirani svi bitni dijelovi igre koji su potrebni za igru, uključujući objekte, neprijatelje, prepreke i druge interaktivne elemente.



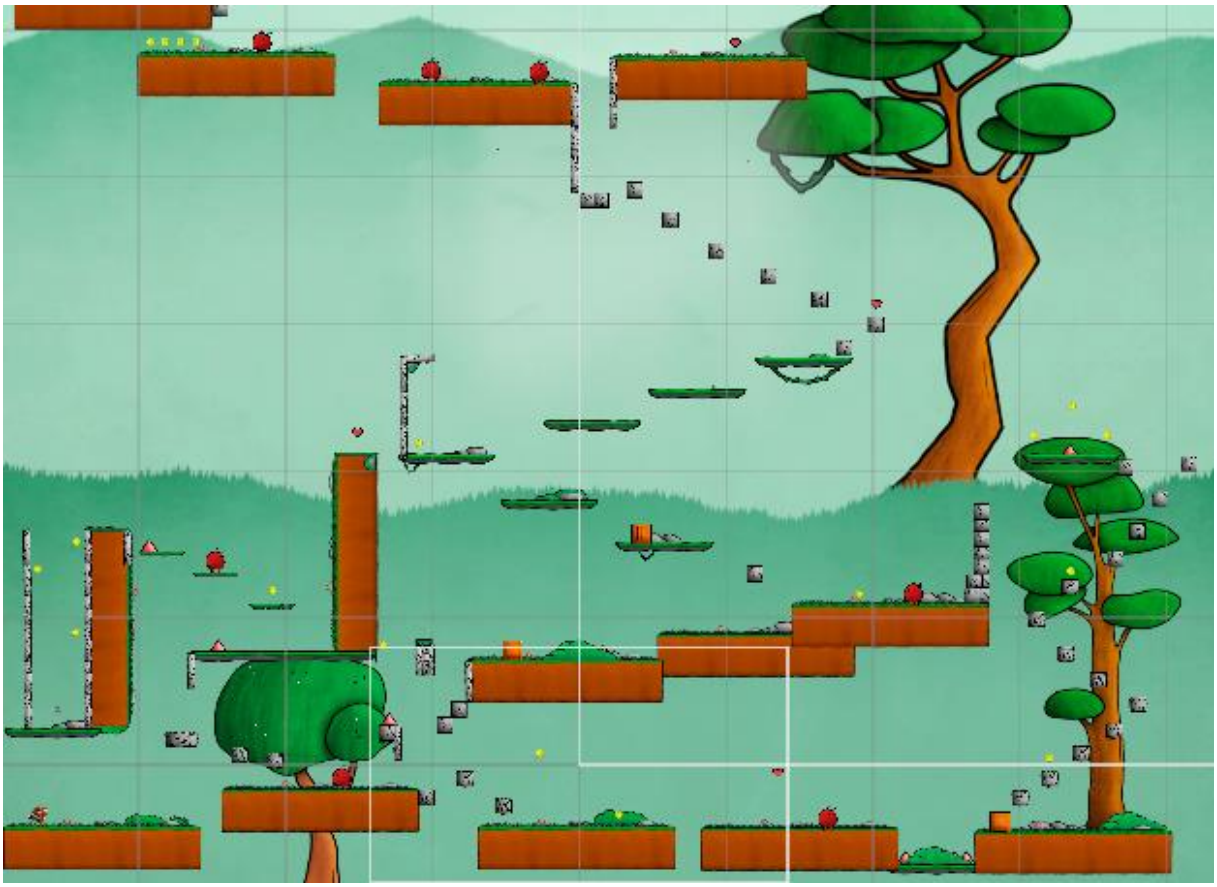
Slika 4 Prikaz sučelja nad scenom razine

Na slici [4] vidi se sučelje kada je otvorena scena Level01. Ovo su ključne komponente sučelja koje možete vidjeti:

- Prozor Hierarchy: Smješten s lijeve strane, ovaj prozor prikazuje hijerarhiju svih objekata prisutnih na sceni. Svaki objekt koji se nalazi na sceni, uključujući likove, neprijatelje, prepreke, i sve druge elemente, može se vidjeti i upravljati preko ovog prozora. Hijerarhija objekata omogućuje lakše organiziranje i upravljanje različitim komponentama i njihovim odnosima u sceni.
- Prozor Project: Smješten pri dnu sučelja, ovaj prozor prikazuje strukturu mapa i datoteka unutar projekta. Ovdje su organizirani svi resursi, uključujući skripte, teksture, zvučne datoteke i druge važne komponente koje se koriste u razvoju igre.

Ovaj prozor omogućuje pristup i upravljanje svim datotekama koje su korištene u projektu.

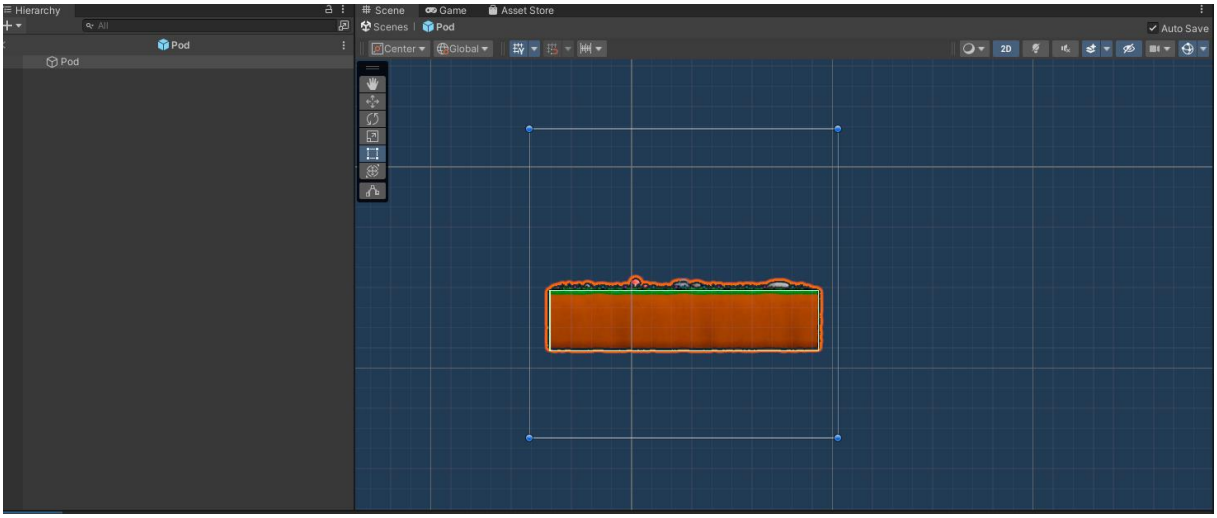
- Scene View: Smješten u sredini ekrana, Scene View pruža vizualni prikaz trenutne scene. Ovdje se mogu pregledavati i uređivati pozicije objekata, mijenjati njihove atribute i testirati raspored elemenata u sceni. Scene View omogućuje dinamičko prilagođavanje razine, dodavanje novih objekata i pregledavanje njihovih interakcija u stvarnom vremenu.



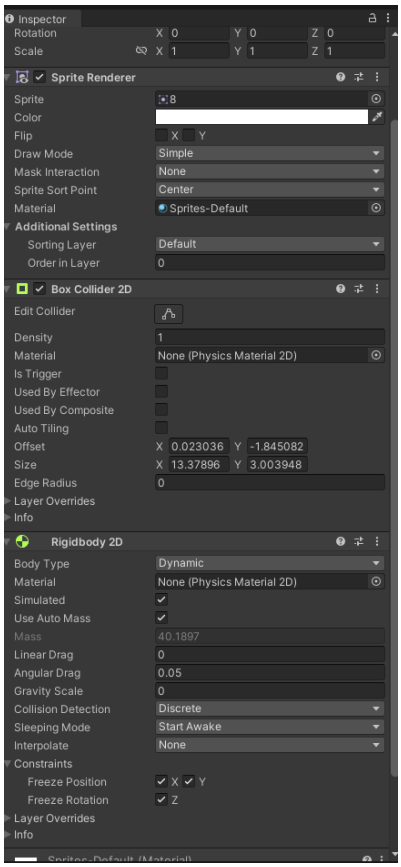
Slika 5 Svijet igre

Svijet je kreiran platformi i zidova. Pod platforme spadaju platforma poda i kamenje, te sa ovih platformi može skakati i pucati. Dok su zivodi sve platforme koje su vertikalno okrenute te se za njih igrač može uhvatiti i skočiti.

Prva platforma je platforma Pod na kojoj se igrač stvara. Na slici [6] se vidi prefab platforme, te na slici [7] komponente. Najbitnija komponenta je Rigidbody2D koja dodaje fiziku platformi te omogućava kretanje po njoj, kao što se vidi na slici [7] gravitacija je stavljena na 0 te su Constraints uključeni, kako se platforma nebi mogla micati i kako bi ostala na istom mjestu od početka do kraja igre.



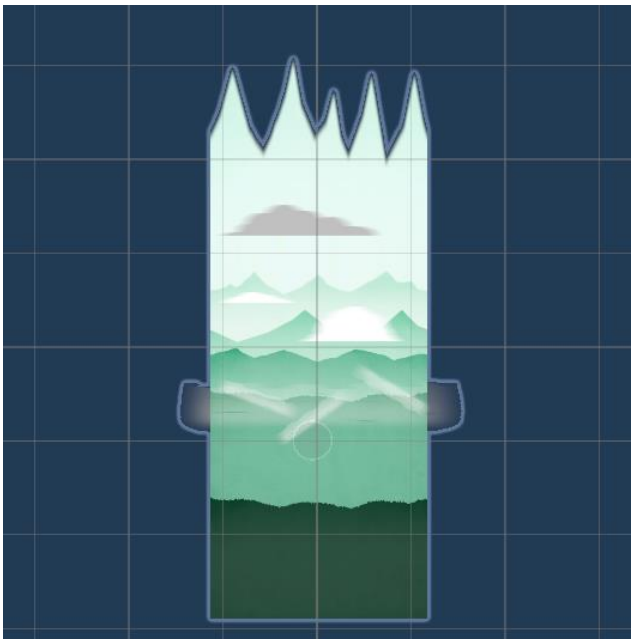
Slika 6 Prefab platforme



Slika 7 Komponente platforme

Platforma Zid je u suštini ista kao i platforma za pod, jedina razlika je u Tagu, no ovo ću objasniti kada budem objašnjavao kod igre.

Pozadina igre je implementirana na način da se sastoji od 6 slika koje su svojstvom Order In Layer postavljene na određene vrijednosti tako da se ne preklapaju i da ona najbliža pozadina ima najveći broj u svojstvu Order In Layer a ona najdalja ima najmanji broj. Kao što se vidi na slici [8] pozadinu čine još i oblaci i magla.



Slika 8 Pozadina razine

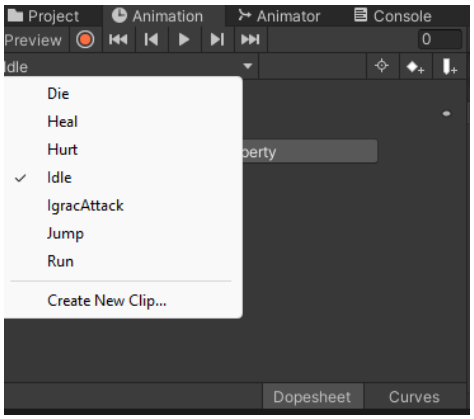
6. Glavni lik

Glavni lik igre je čovjek sa glavom zmaja i krilima. Glavni cilj mu je doći do vrha i poraziti glavnog neprijatelja koji ga čeka. U ovom poglavlju bit će objašnjeno sve vezano uz glavnog lika igrice kao što su animacije, skripte za kretanje pucanje, projektili, sustav životnih bodova i prikupljanje novčića i napitaka za pojačavanje jačine skoka.

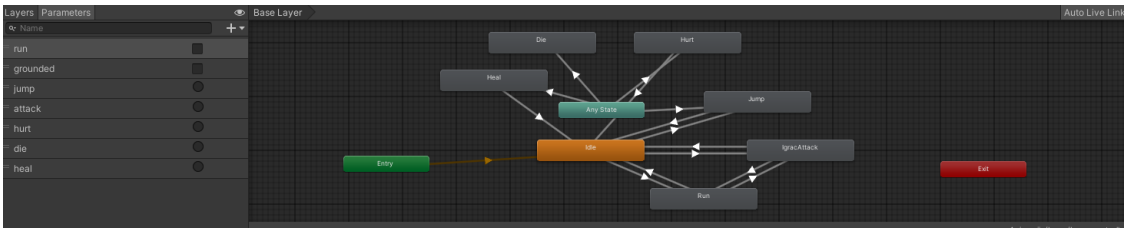
Slika 9 Glavni lik

6.1. Animacije glavnog lika

Animacije glavnog lika napravljene su tako što je na objekt igrača dodana komponenta Animator, na koju se povezuje Controller koji upravlja animacijama našeg glavnog lika. Na slici [10] prikazan je prozor Animation u kojem se stvaraju animacije. Kako je resurs glavnog lika uvežen sa Unity Asset Store-a, on dolazi sa već predodređenim animacijama, koje se ručno moraju postaviti. Kada se stvori nova animacija mora se pokrenuti snimanje klikom na crvenu točku unutar Animation prozora, te se onda pomicanjem po vremenskoj traci animira lik. U mom slučaju ja sam za svaku od animacija prikazanih na slici [10] imao sve Sprite-ove koji su mi bili potrebni pa sam samo u razmaku od 0:05 sekundi postavio drugu sliku lika. Na kraju kada se završe animacije one se nekako moraju i povezati što je napravljeno pomoću Animator prozora.

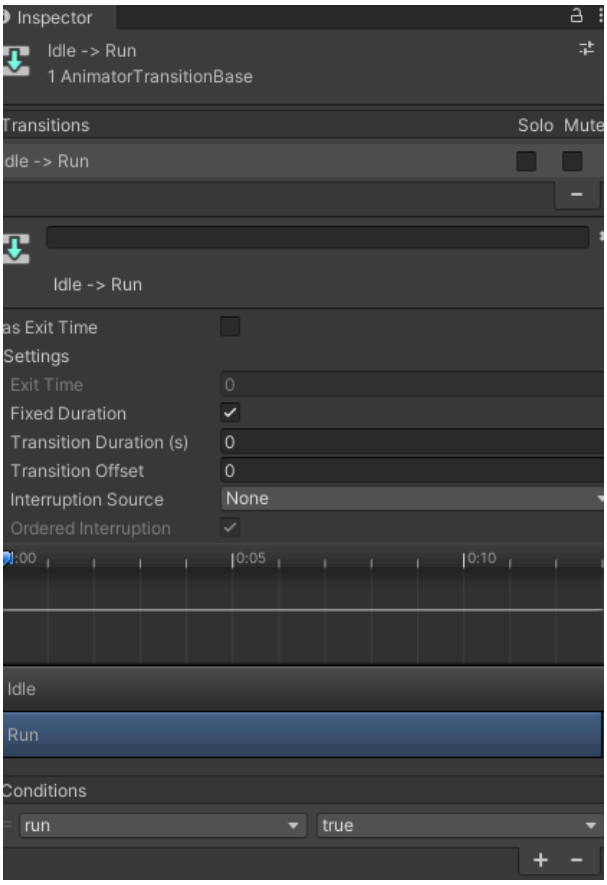


Slika 10 Animacije glavnog lika



Slika 11 Animator glavnog lika

U prozoru Animator vide se sve animacije koje sam kreirao za glavnog lika. Animacija kreće od Entry tj. od ulaza, gdje se on spaja na onu animaciju koja će se prva prikazivati, u mom slučaju to je animacija Idle koja se odvija kada lik stoji na mjestu i ne kreće se. Od te animacije može se napraviti logika da se prijeđe u sve ostale animacije. Ja ću konkretno pokazati samo kako sam naporavio da se iz animacije Idle prelazi u animaciju Run. Na slici [12] se vide postavke prijelaza iz Idle u Run. Postavljen je Condition ako je run true, onda će se prijeći u animaciju Run, isto tako za povratak na animaciju Idle je postavljen Condition ako je run false. Taj condition se postavlja preko koda, o ovome ću više objašnjavati poslje

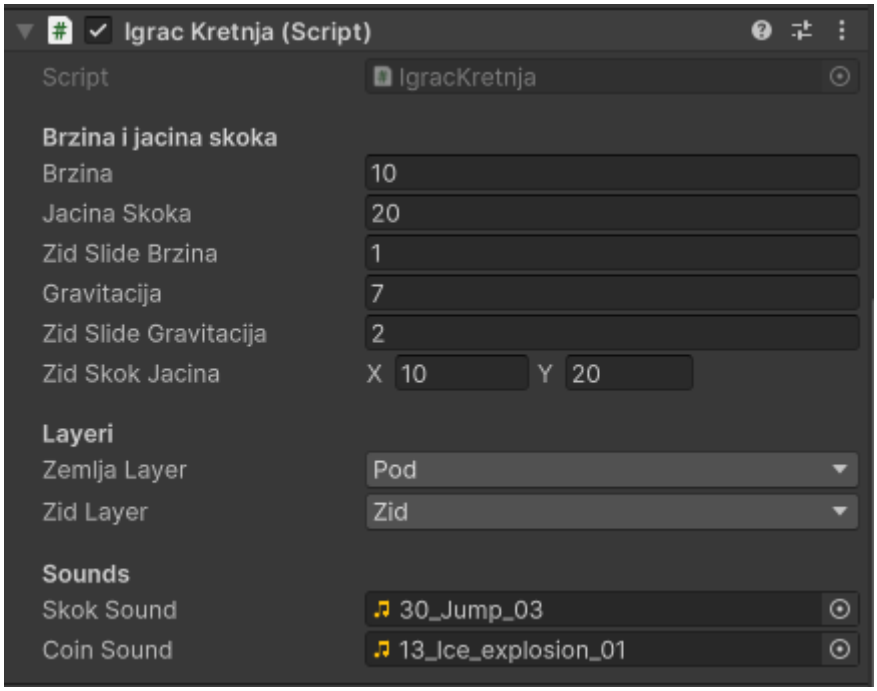


Slika 12 Prijelaz animacije Idle u Run

6.2. Kretanje igrača

Na objektu igrača primijenjena je skripta IgracKretnja koja upravlja svim aspektima kretanja igrača u igri. Ova skripta omogućuje igraču da se kreće lijevo-desno, skače, penje se uz zidove, i interagira s drugim objektima u igri. U nastavku je opis glavnih komponenti i funkcionalnosti ove skripte. Varijable brzina i jacinaSkoka određuju brzinu kretanja igrača i visinu skoka. Varijabla zidSlideBrzina kontrolira brzinu klizanja po zidu, dok gravitacija i zidSlideGravitacija definiraju jačinu gravitacije u različitim situacijama. Varijabla zidSkokJacina određuje jačinu skoka kada igrač skače sa zida. Skripta koristi različite komponente kao što su Rigidbody2D, Animator, BoxCollider2D, i Health. Komponenta CoinManager koristi se za upravljanje brojem skupljenih novčića. U metodi Start() skripta inicijalizira sve potrebne komponente i provjerava postoji li instanca CoinManager-a. Metoda Update() upravlja horizontalnim kretanjem igrača i animacijama. Igrač se kreće lijevo-desno na temelju unosa tipke. Ako igrač pritisne razmaknicu (Space), a trenutno je na zemlji ili uz zid, izvršit će se skakanje ili zidni skok, ovisno o situaciji. Metoda WallSlide() omogućava igraču da klizi niz zid ako nije na zemlji. Metoda TrcanjePoPodu() upravlja horizontalnim kretanjem igrača kada nije

uz zid. Metoda Uzemljen() provjerava je li igrač na zemlji koristeći Raycast, te metoda naZidu() provjerava je li igrač uz zid koristeći Raycast. Na kraju metoda OnTriggerEnter2D upravlja interakcijama igrača s različitim objektima poput novčića, napitaka, portala i prepreka. Ovdje se dodaju bodovi za novčiće, povećava jačina skoka za napitke, ili aktivira portal i šteta od bodlji i nevidljivih zidova.



Slika 13 Varijable kretnje igrača

```
public class IgracKretnja : MonoBehaviour
{
    [Header("Brzina i jacina skFoka")]
    [SerializeField] private float brzina = 10f;
    [SerializeField] private float jacinaSkoka = 20f;
    [SerializeField] private float zidSlideBrzina = 1f;
    [SerializeField] private float gravitacija = 7f;
    [SerializeField] private float zidSlideGravitacija = 2f;
    [SerializeField] private Vector2 zidSkokJacina = new Vector2(10f, 20f);

    private float horizontalnoKretanje;

    private Rigidbody2D rb2D;
    private Animator animator;
    private BoxCollider2D boxCollider;
```

```
    private Health health;

    private CoinManager coinMan;

    [Header("Layeri")]
    [SerializeField] private LayerMask zemljaLayer;
    [SerializeField] private LayerMask zidLayer;

    [Header("Sounds")]
    [SerializeField] private AudioClip skokSound;
    [SerializeField] private AudioClip coinSound;

    void Start()
    {
        rb2D = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
        boxCollider = GetComponent<BoxCollider2D>();
        health = GetComponent<Health>();


        if (CoinManager.instance != null)
        {
            coinMan = CoinManager.instance;
        }
        else
        {
            Debug.LogError("CoinManager instance is missing.");
        }
    }

    void Update()
    {
        horizontalnoKretanje = Input.GetAxisRaw("Horizontal");

        animator.SetBool("run", Mathf.Abs(horizontalnoKretanje) > 0.01f);
        animator.SetBool("grounded", Uzemljen());

        if (horizontalnoKretanje != 0)
        {
            transform.localScale = new
Vector3(Mathf.Sign(horizontalnoKretanje), 1, 1);
```

```
        }

        if (Input.GetKeyDown(KeyCode.Space))
        {
            if (Uzemljen())
            {
                SoundManager.instance.PlayAudio(skokSound);
                Skok();
            }
            else if (naZidu() && horizontalnoKretanje ==
Mathf.Sign(transform.localScale.x))
            {
                SoundManager.instance.PlayAudio(skokSound);
                ZidSkok();
            }
        }
    }

    private void FixedUpdate()
    {
        if (naZidu() && !Uzemljen() && horizontalnoKretanje ==
Mathf.Sign(transform.localScale.x))
        {
            WallSlide();
        }
        else
        {
            TrcanjePoPodu();
        }
    }

    private void WallSlide()
    {
        rb2D.gravityScale = zidSlideGravitacija;

        if (rb2D.velocity.y < 0)
        {
            rb2D.velocity = new Vector2(rb2D.velocity.x, -zidSlideBrzina);
        }
    }
```

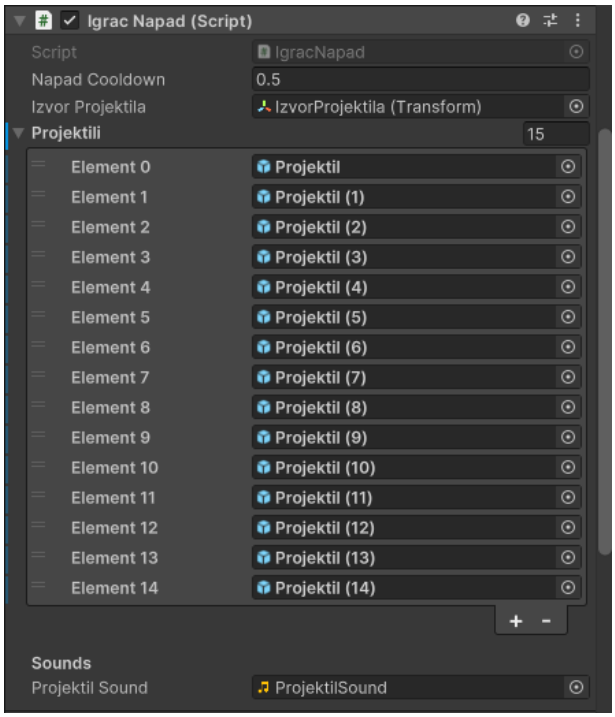
```
    private void TrcanjePoPodu()
    {
        rb2D.gravityScale = gravitacija;
        rb2D.velocity = new Vector2(horizontalnoKretanje * brzina,
rb2D.velocity.y);
    }

    private void Skok()
    {
        rb2D.velocity = new Vector2(rb2D.velocity.x, jacinaSkoka);
        animator.SetTrigger("jump");
    }

    private void ZidSkok()
    {
        float smjerSkoka = -Mathf.Sign(transform.localScale.x);
        rb2D.velocity = new Vector2(smjerSkoka * zidSkokJacina.x,
zidSkokJacina.y);
        animator.SetTrigger("jump");
    }

    private bool Uzemljen()
    {
        RaycastHit2D raycastHit =
Physics2D.Raycast(boxCollider.bounds.center, Vector2.down,
boxCollider.bounds.extents.y + 0.1f, zemljaLayer);
        return raycastHit.collider != null;
    }

    private bool naZidu()
    {
        RaycastHit2D raycastHit =
Physics2D.Raycast(boxCollider.bounds.center, new
Vector2(transform.localScale.x, 0), boxCollider.bounds.extents.x + 0.1f,
zidLayer);
        return raycastHit.collider != null;
    }

    public bool mozeNapadati()
    {
        return Uzemljen() && !naZidu();
    }
```

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Coin"))
        {
            SoundManager.instance.PlayAudio(coinSound);
            Destroy(collision.gameObject);
            coinMan.addCoins(1);
        }
        if (collision.gameObject.CompareTag("Napitak"))
        {
            Destroy(collision.gameObject);
            jacinaSkoka = jacinaSkoka + 2f;
        }
        if (collision.gameObject.CompareTag("Portal"))
        {
            if (coinMan.coinBrojac > 20) {
                Vector2 vector2 = new Vector2(transform.position.x,
transform.position.y + 20f);
                rb2D.position = vector2;
            }
        }
        if (collision.gameObject.CompareTag("NevidljiviZidDolje"))
        {
            health.health = 0;
        }
        if (collision.gameObject.CompareTag("Bodlja"))
        {
            health.health = 0;
        }
    }

}
```

6.3. Pucanje

Skripta IgracNapad upravlja napadom, suština ove skripte je da kada igrač pritisne tipku miša da se pozove funkcija Napad(). Ova funkcija ispaljuje projektil tako što postavlja njegovu poziciju na izvorProjektila i aktivira ga. Također pokreće animaciju napada i postavlja zvučni efekt za ispaljivanje projektila. Vrijeme čekanja za sljedeći napad se resetira. Projektili

su implementirani pomoću object pooling metode. To znači da je unutar scene već kreirano veći broj projektila, te su oni deaktivirani. Kada se pozove ova funkcija oni se postave na određeno mjesto na ispaljivanje te se aktiviraju. Isto tako je implementirano da projektili imaju zivotni vijek da nebi mogli putovati beskonačno u jednom smjeru ako ne udare ni u što.



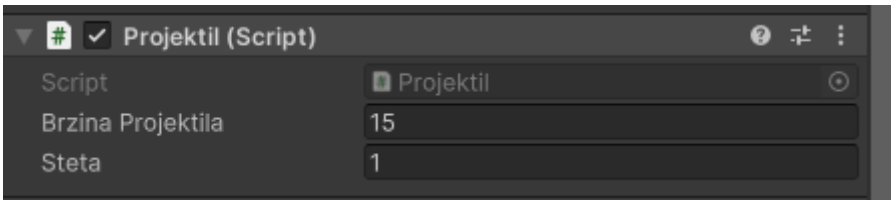
Slika 14 Varijable pucanja igrača

```
public class IgracNapad : MonoBehaviour
{
    private Animator animator;
    private IgracKretnja igracKretnja;
    [SerializeField] private float napadCooldown;
    private float napadCooldownVrijeme = 10;

    [SerializeField] private Transform izvorProjektila;
    [SerializeField] private GameObject[] projektili;


    [Header("Sounds")]
    [SerializeField] private AudioClip projektilSound;

    // Start is called before the first frame update
    void Start()
```

```
    {
        animator = GetComponent<Animator>();
        igracKretnja = GetComponent<IgracKretnja>();
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButton(0) && napadCooldownVrijeme > napadCooldown
&& igracKretnja.mozeNapadati()) Napad();
        napadCooldownVrijeme += Time.deltaTime;
    }

    private void Napad()
    {
        SoundManager.instance.PlayAudio(projektilSound);
        animator.SetTrigger("attack");
        napadCooldownVrijeme = 0;

        projektili[PronadiProjektil()].transform.position =
izvorProjektila.position;

projektili[PronadiProjektil()].GetComponent<Projektil>().PostaviSmjer(Mathf
.Sign(transform.localScale.x));
    }

    private int PronadiProjektil()
    {
        for (int i = 0;i < projektili.Length;i++)
        {
            if (!projektili[i].activeInHierarchy)
            {
                return i;
            }
        }
        return 0;
    }
}
```

Iduća na redu je skripta koja je zadužena za upravljanje projektilom u igri, zadužena je za njegovu interakciju s drugim objektima u igri, te za animacije i deaktivaciju projektila koju

sam već objasnio u prijašnjem odlomku. Metoda OnTriggerEnter2D() se poziva kada projektil dođe u kontakt s drugim objektima u igri. Različiti tagovi objekata uzrokuju različite reakcije. Ako projektil udari u objekt s tagom "Platforma" ili "Zid", označava da je projektil udario (udar je true), deaktivira collider i pokreće animaciju eksplozije. Ako udari u neprijatelja (Neprijatelj), također pokreće animaciju eksplozije, te nanosi štetu neprijatelju koristeći PrimiStetu metodu. Ako udari u objekt s tagom "Kutija", pokreće animaciju eksplozije, uništava kutiju i deaktivira collider. Metoda PostaviSmjer(float _smjer) postavlja smjer u kojem projektil putuje i resetira trajanje. Aktivira projektil i postavlja njegovu lokalnu skalaciju tako da se uskladi s smjerom. Te na kraju metoda Deaktiviraj() koja deaktivira projektil tako što ga postavlja na neaktivan SetActive(false).



Slika 15 Varijable projektila

```
public class Projektil : MonoBehaviour
{

    [SerializeField] private float brzinaProjektila;
    private bool udar;
    private float smjer;
    private float trajanje;


    private Animator animator;
    private CapsuleCollider2D capsuleCollider;

    [SerializeField] private int steta = 1;



    // Start is called before the first frame update
    void Start()
    {
        animator = GetComponent<Animator>();
```

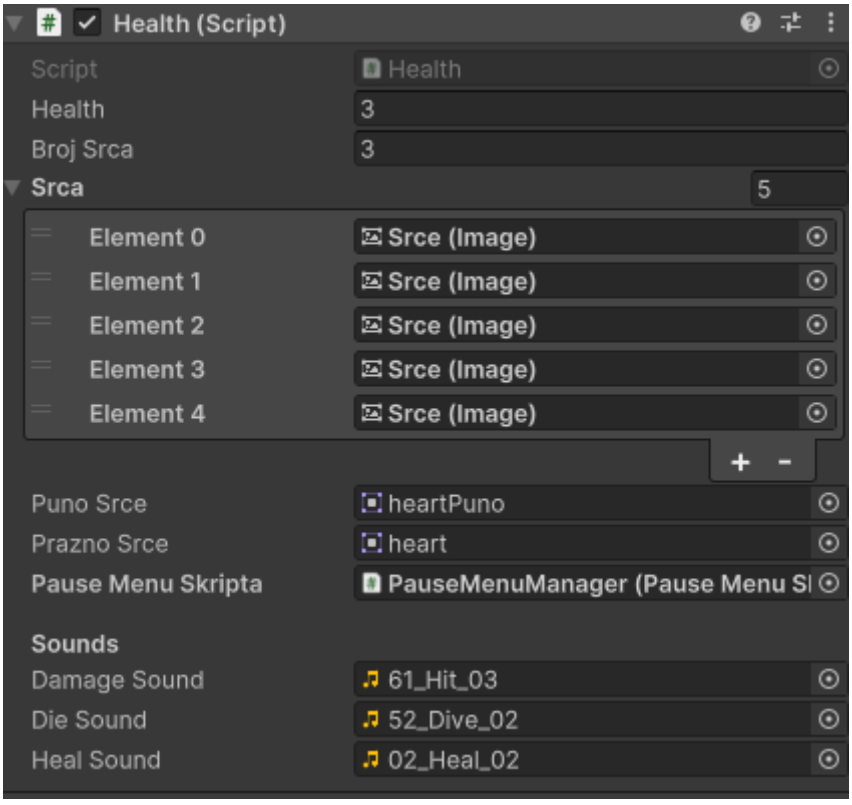
```
        capsuleCollider = GetComponent<CapsuleCollider2D>();

    }

    // Update is called once per frame
    void Update()
    {
        if (udar) return;
        float brzinaKretnje = brzinaProjektila * Time.deltaTime * smjer;
        transform.Translate(brzinaKretnje, 0, 0);

        trajanje += Time.deltaTime;
        if (trajanje > 10 ) gameObject.SetActive(false);

    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (udar) return;

        if (collision.CompareTag("Platforma") ||
collision.CompareTag("Zid"))
        {
            udar = true;
            capsuleCollider.enabled = false;
            animator.SetTrigger("eksplozija");
        }

        else if (collision.CompareTag("Neprijatelj"))
        {
            udar = true;
            capsuleCollider.enabled = false;
            animator.SetTrigger("eksplozija");

            NeprijateljHealth neprijateljHealth =
collision.GetComponent<NeprijateljHealth>();

            if (neprijateljHealth != null )
            {
                neprijateljHealth.PrimiStetu(1);
            }
```

```
        }

        else if (collision.CompareTag("Kutija"))
        {
            udar = true;
            capsuleCollider.enabled = false;
            animator.SetTrigger("eksplozija");
            Destroy(collision.gameObject);
        }

    }

    public void PostaviSmjer(float _smjer)
    {
        trajanje = 0;
        smjer = _smjer;
        gameObject.SetActive(true);
        udar = false;
        capsuleCollider.enabled = true;

        float localScaleX = transform.localScale.x;
        if (Mathf.Sign(localScaleX) != _smjer) localScaleX = -localScaleX;

        transform.localScale = new Vector3(localScaleX,
transform.localScale.y, transform.localScale.z);
    }

    private void Deaktiviraj()
    {
        gameObject.SetActive(false);
    }

}
```

6.4. Životni bodovi

Skripta Health upravlja životnim bodovima igrača u igri, kao i njihovim vizualnim prikazom i interakcijama s drugim objektima. Metoda Start() inicijalizira animator i igracKretnja

komponente. Metoda Update() provjerava je li igrač mrtav (health <= 0). Ako jest, postavlja jeMrtav na true, deaktivira igračev GameObject, postavlja stanje igre na "game over" i poziva metodu gameOver() iz pauseMenuSkripta, osigurava da broj životnih bodova ne prelazi maksimalni broj srca, te ažurira prikaz srca u UI na temelju trenutnog broja životnih bodova. Metoda PrimiStetu() Smanjuje broj životnih bodova igrača za 1, ako igrač još uvijek ima životne bodove, reproducira zvuk štete i pokreće animaciju povrede te ako su životni bodovi nula ili manji od nula, reproducira zvuk smrti, pokreće animaciju smrti, zaustavlja vrijeme igre (Time.timeScale = 0f), deaktivira igračevu skriptu za kretanje i poziva gameOver() metodu iz pauseMenuSkripta. Nadalje je implementirana metoda DodajSrce() koja dodaje igraču srce ako on u trenutku prikupljanja srca ima manje životnih bodova od maksimalnog broja životnih bodova. Te metoda OnTriggerEnter2D() poziva metodu DodajSrce() kada igrač dođe u kontakt s objektom koji ima tag „Srce“, te isto tako uništava taj objekt unutar igre.



Slika 16 Varijable životnih bodova igrača

```
public class Health : MonoBehaviour
{
    public int health;
    public int brojSrca;
```

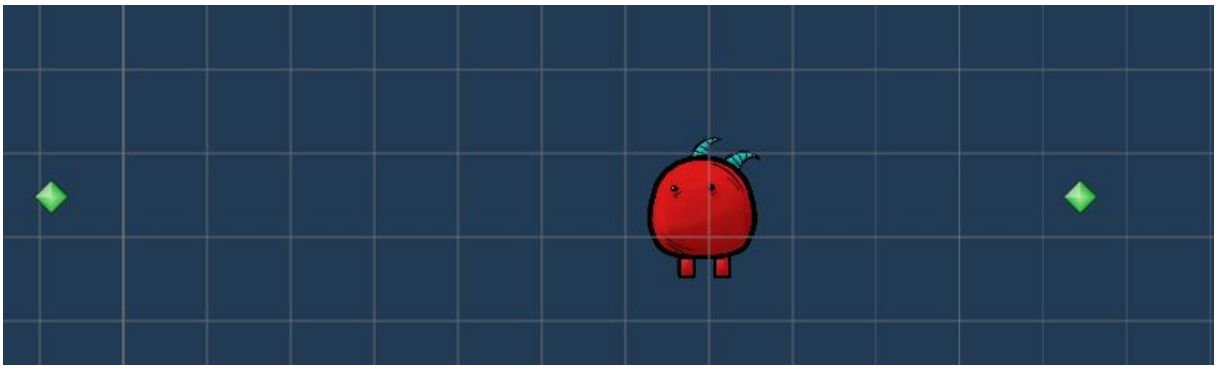
```
public Image[] srca;
public Sprite punoSrce;
public Sprite praznoSrce;
private Animator animator;
private IgracKretnja igracKretnja;
public PauseMenuSkripta pauseMenuSkripta;

private bool jeMrtav;


[Header("Sounds")]
[SerializeField] private AudioClip damageSound;
[SerializeField] private AudioClip dieSound;
[SerializeField] private AudioClip healSound;

// Start is called before the first frame update
void Start()
{
    animator = GetComponent<Animator>();
    igracKretnja = GetComponent<IgracKretnja>();

}

// Update is called once per frame
void Update()
{

    if (health <= 0 && !jeMrtav)
    {
        jeMrtav = true;
        gameObject.SetActive(false);
        pauseMenuSkripta.jeMrtav = true;
        pauseMenuSkripta.gameOver();
        Debug.Log("Mrtav");
    }

    if ( health > brojSrca)
    {
        health = brojSrca;
    }
```

```
    for (int i = 0; i < srca.Length; i++)
    {
        if (i < health)
        {
            srca[i].sprite = punoSrce;
        }
        else
        {
            srca[i].sprite = praznoSrce;
        }

        if (i < brojSrca)
        {
            srca[i].enabled = true;
        }
        else
        {
            srca[i].enabled = false;
        }
    }

}

public void PrimiStetu()
{
    health--;

    if (health > 0)
    {
        SoundManager.instance.PlayAudio(damageSound);
        animator.SetTrigger("hurt");
    }
    if (health <= 0)
    {
        SoundManager.instance.PlayAudio(dieSound);
        animator.SetTrigger("die");
        Time.timeScale = 0f;
        igracKretnja.enabled = false;
        pauseMenuSkripta.jeMrtav = true;
```

```
            pauseMenuSkripta.gameOver();
        }
    }

    public void DodajSrce()
    {
        if (health < brojSrca)
        {
            health++;
            SoundManager.instance.PlayAudio(healSound);
            animator.SetTrigger("heal");
        }
    }


    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Srce"))
        {
            DodajSrce();
            Destroy(collision.gameObject);
        }
    }
}
```

7. Neprijatelji

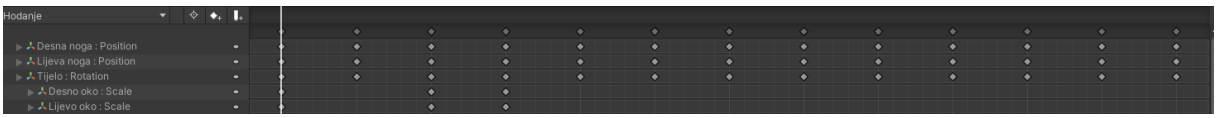
Objekt neprijatelja je zamišljen kao kamen, koji spriječava igrača da dosegne vrh razine i porazi glavnog neprijatelja. Neprijatelji se kreću lijevo desno te pucaju igrača kada se on pronađe ispred njih. U ovom poglavlju ću opisati sve skripte koje su potrebne kako bi neprijatelji radili te na kraju poglavlja ću se dotaknuti glavnog neprijatelja i njegove posebne skripte. Na slici [13] se vidi sprite neprijatelja te 2 zelene točke u prostoru koje označavaju točke do kojih se neprijatelj kreće, kada dostigne do jedne točke okreće se te se krene kretati prema drugoj točki.



Slika 17 Neprijatelj

7.1. Animacije neprijatelja

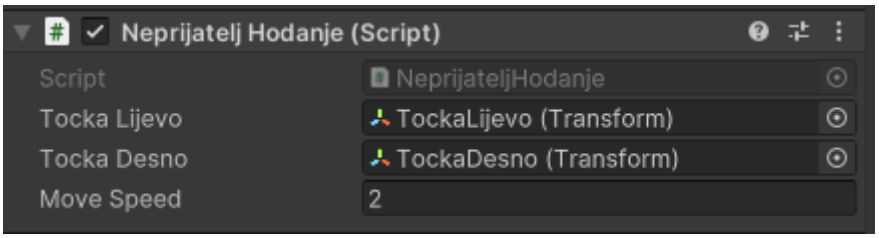
Animacije neprijatelja napravljene su isto kao i za igrača, no međutim ovi resursi nisu došli sa više slika nego samo jednom, pa sam ručno morao postavljati parametre svake animacije, npr. tijela i glave, i svake noge zasebno. Na slici [14] se vidi kako sam na svakih 0:05 sekundi promijenio poziciju, rotaciju ili veličinu određenih elemenata kako bi se na kraju dobila cjelokupna animacija hodanja neprijatelja.



Slika 18 Vremenska traka animacije neprijatelja

7.2. Neprijatelji kretanje

Skripta NeprijateljHodanje omogućuje neprijatelju da se automatski kreće lijevo i desno između dvije zadane točke. Na početku igre, neprijatelj se postavlja da kreće prema lijevoj točki (tockaLijevo). Svaki put kad neprijatelj dosegne ovu točku, skripta ga preusmjerava prema desnoj točki (tockaDesno) i obrnuto. U Update metodi, neprijatelj se pomiče prema trenutnoj ciljnoj točki (targetPoint) koristeći metodu Vector2.MoveTowards, čija brzina ovisi o varijabli moveSpeed i vremenskoj jedinici (Time.deltaTime). Kada neprijatelj dosegne vrlo malu udaljenost od ciljne točke (manju od 0.1 jedinice), skripta mijenja ciljnu točku na suprotnu točku i poziva metodu Flip kako bi preokrenula neprijatelja u smjeru kretanja. Metoda Flip postiže ovo tako što mijenja znak x komponente localScale transformacije, što uzrokuje da neprijatelj izgleda kao da se okreće u suprotnom smjeru.



Slika 19 Varijable neprijatelj kretanje

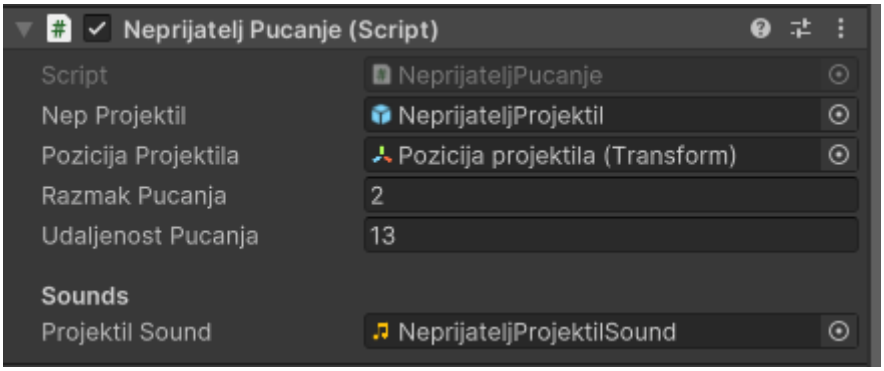
```
public class NeprijateljHodanje : MonoBehaviour
{
    public Transform tockaLijevo;
    public Transform tockaDesno;
    public float moveSpeed = 2f;
    private Transform targetPoint;

    private void Start()
    {
        targetPoint = tockaLijevo;
    }

    private void Update()
    {
        transform.position = Vector2.MoveTowards(transform.position,
targetPoint.position, moveSpeed * Time.deltaTime);
```

```
        if (Vector2.Distance(transform.position, targetPoint.position) <
0.1f)
        {
            targetPoint = targetPoint == tockaLijevo ? tockaDesno :
tockaLijevo;
            Flip();
        }
    }

    private void Flip()
    {
        Vector3 localScale = transform.localScale;
        localScale.x *= -1;
        transform.localScale = localScale;
    }
}
```

7.3. Neprijatelj pucanje

Pucanje neprijatelja implementirano je na sličan način kao kod igrača, no ovaj put nisam koristio object pooling za projektile nego svaki puta kada neprijatelj puca stvara se novi objekt na sceni te se uništava kada se sudari ili mu istekne životni vijek. Sada ću objasniti dvije skripte koje su zadužene za pucanje neprijatelja.

Skripta NeprijateljPucanje upravlja pucanjem neprijatelja u igri, omogućujući mu da puca na igrača kad se nađe u određenom radijusu. Na početku igre, skripta pronalazi igrača u sceni pomoću oznake "Igrac" i dohvaća komponentu Animator sa svog objekta. U Update metodi, skripta prvo izračunava udaljenost između neprijatelja i igrača. Ako je udaljenost manja od zadane vrijednosti (udaljenostPucanja) i ako neprijatelj gleda prema igraču, skripta broji vrijeme koje prolazi. Kada vrijeme premaši interval između pucanja (razmakPucanja), neprijatelj ispaljuje projektil koristeći metodu NepPucaj. Metoda DaliGledaUIgraca provjerava smjer u kojem neprijatelj gleda u odnosu na smjer prema igraču. Ako su smjerovi različiti, vraća true, što znači da neprijatelj gleda prema igraču. Ako su smjerovi isti, vraća false, što znači da neprijatelj ne gleda prema igraču i stoga ne bi trebao pucati. Metoda NepPucaj odgovorna je za ispaljivanje projektila. Ona pokreće zvučni efekt pucanja, aktivira animaciju pucanja pomoću SetTrigger, te instancira novi projektil na poziciji zadanoj varijablom pozicijaProjektila.



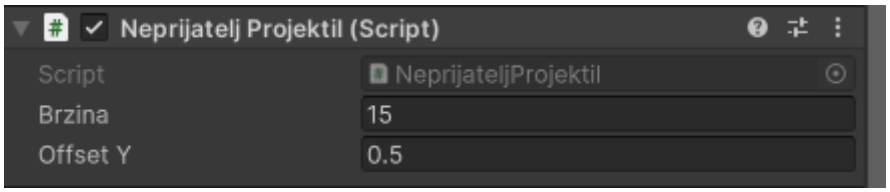
Slika 20 Varijable neprijatelj pucanje

```
public class NeprijateljPucanje : MonoBehaviour
{
    public GameObject nepProjektil;
    public Transform pozicijaProjektila;

    [SerializeField] private float razmakPucanja;
    [SerializeField] private float udaljenostPucanja;

    private float vrijeme;
    private GameObject igrac;
    private Animator animator;


    [Header("Sounds")]
    [SerializeField] private AudioClip projektilSound;

    // Start is called before the first frame update
    void Start()
    {
        igrac = GameObject.FindGameObjectWithTag("Igrac");
        animator = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        float udaljenost = Vector2.Distance(transform.position,
igrac.transform.position);
```

```
        Vector3 smjer = igrac.transform.position - transform.position;

        if (udaljenost < udaljenostPucanja && DaliGledaUIgraca(smjer))
        {
            vrijeme += Time.deltaTime;

            if (vrijeme > razmakPucanja)
            {
                vrijeme = 0;
                NepPucaj();
            }
        }
    }

    private bool DaliGledaUIgraca(Vector3 smjer)
    {
        float smjerNeprijatelja = Mathf.Sign(transform.localScale.x);
        float smjerKaIgracu = Mathf.Sign(smjer.x);

        return smjerNeprijatelja != smjerKaIgracu;
    }

    public void NepPucaj()
    {
        SoundManager.instance.PlayAudio(projektilSound);
        animator.SetTrigger("Pucanje");
        Instantiate(nepProjektil, pozicijaProjektila.position,
Quaternion.identity);
    }

}
```

Skripta NeprijateljProjektil upravlja ponašanjem neprijateljskih projektila u igri. Na početku, skripta pronalazi igrača u sceni koristeći oznaku "Igrac" i dohvaća komponentu Health sa igračeva objekta. Također, dohvaća komponentu Rigidbody2D za upravljanje fizikom projektila. U Start metodi, skripta izračunava smjer prema igraču uzimajući u obzir offsetY, koji služi za prilagodbu pozicije ispaljivanja projektila kako bi se osiguralo da projektil cilja prema igraču ispravno. Izračunava se brzina projektila pomoću rb2D.velocity i smjer rotacije se postavlja tako da projektil bude usmjeren prema igraču. Projektil se rotira za 180 stupnjeva

kako bi izgledao kao da je usmjeren prema cilju. U Update metodi, skripta prati vrijeme i nakon 10 sekundi uništava projektil kako bi se izbjeglo nakupljanje neaktivnih objekata u sceni. Metoda OnTriggerEnter2D upravlja sudarima projektila. Ako projektil udari u objekt označen kao "Igrac", poziva PrimiStetu metodu sa Health komponente igrača kako bi mu nanio štetu, a zatim uništava projektil. Ako projektil udari u objekte označene kao "Zid" ili "Platforma", također se uništava.



Slika 21 Varijablje neprijatelj projektil

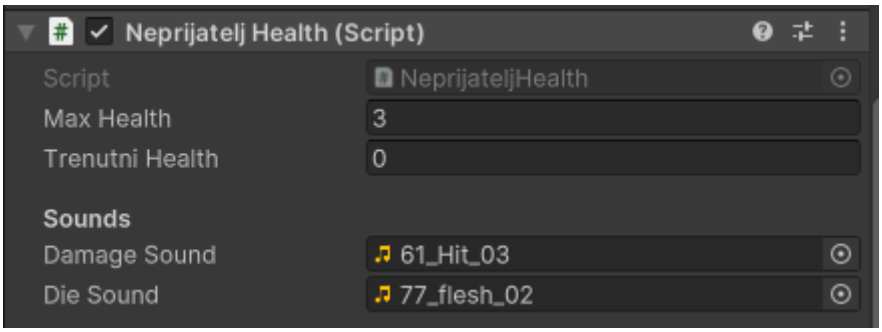
```
public class NeprijateljProjektil : MonoBehaviour
{
    private GameObject igrac;
    private Rigidbody2D rb2D;
    private float vrijeme;
    [SerializeField] private float brzina;
    [SerializeField] private float offsetY = 0.5f;

    private Health igracHealth;

    // Start is called before the first frame update
    void Start()
    {
        igrac = GameObject.FindGameObjectWithTag("Igrac");
        igracHealth =
GameObject.FindGameObjectWithTag("Igrac").GetComponent<Health>();
        rb2D = GetComponent<Rigidbody2D>();

        // Koristi offsetY varijablu za kontrolisanje pozicije gađanja
        Vector3 smjer = (igrac.transform.position - new Vector3(0, offsetY,
0)) - transform.position;
        rb2D.velocity = new Vector2(smjer.x, smjer.y).normalized * brzina;

        float rotacija = Mathf.Atan2(-smjer.y, -smjer.x) * Mathf.Rad2Deg;
        transform.rotation = Quaternion.Euler(0, 0, rotacija + 180);
    }
```

```
    // Update is called once per frame
    void Update()
    {
        vrijeme += Time.deltaTime;

        if (vrijeme > 10)
        {
            Destroy(gameObject);
        }
    }

    public void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("Igrac"))
        {

            if (igracHealth != null)
            {
                igracHealth.PrimiStetu();
            }
            Destroy(gameObject);
        }
        else if (collision.gameObject.CompareTag("Zid") ||
collision.gameObject.CompareTag("Platforma"))
        {
            Destroy(gameObject);
        }
    }

}
```

7.4. Neprijatelj životni bodovi

Skripta NeprijateljHealth upravlja životnim bodovima neprijatelja u igri. Na početku, skripta inicijalizira životne bodove neprijatelja postavljanjem trenutnog broja životnih bodova (trenutniHealth) na maksimalan broj (maxHealth) i dohvaća komponentu Animator za animiranje neprijatelja. Metoda PrimiStetu se poziva kada neprijatelj primi štetu. Ova metoda smanjuje trenutne životne bodove za zadani iznos štete i pokreće animaciju za primanje štete uz reproduciranje zvučnog efekta štete. Ako trenutni broj životnih bodova padne na nulu,

aktivira se animacija smrti neprijatelja. Metoda Smrt se koristi za reprodukciju zvuka smrti neprijatelja i uništavanje neprijateljskog objekta kada njegovi životni bodovi postanu nula. Ovo omogućava pravilno uklanjanje neprijatelja iz igre i pruža povratne informacije o smrti neprijatelja kroz zvuk.



Slika 22 Varijable neptijatelj životni bodovi

```
public class NeprijateljHealth : MonoBehaviour
{
    public int maxHealth = 3;
    public int trenutniHealth;
    private Animator animator;


    [Header("Sounds")]
    [SerializeField] private AudioClip damageSound;
    [SerializeField] private AudioClip dieSound;

    // Start is called before the first frame update
    void Start()
    {
        trenutniHealth = maxHealth;
        animator = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {

    }
```

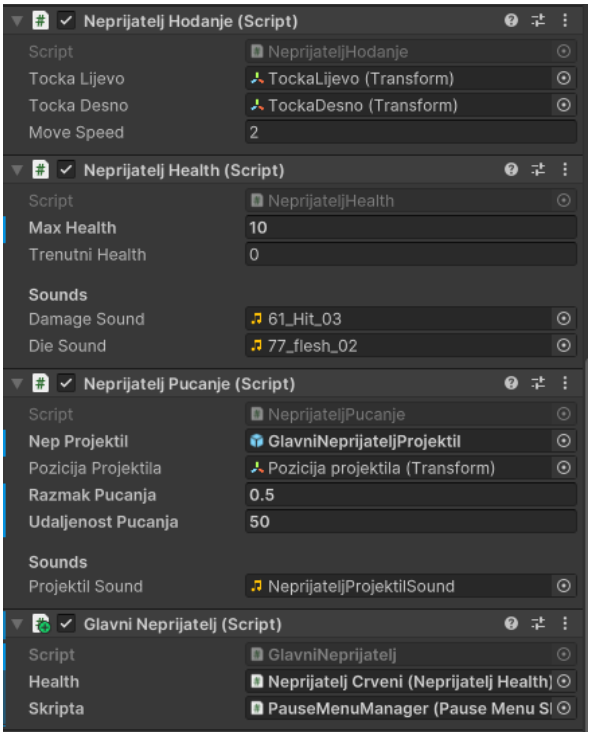
```
    public void PrimiStetu(int iznosStete)
    {
        trenutniHealth -= iznosStete;
        animator.SetTrigger("PrimiStetu");
        SoundManager.instance.PlayAudio(damageSound);

        if (trenutniHealth == 0)
        {
            animator.SetTrigger("Die");
        }
    }

    private void Smrt()
    {
        SoundManager.instance.PlayAudio(dieSound);
        Destroy(gameObject);
    }

}
```

7.5. Glavni neprijatelj

Glavni neprijatelj radi na istom principu kao i svi ostali normalni neprijatelji, samo što su mu varijable za pucanje pojačane, brzina mu je postavljena na 0.5, i radijus pucanja na 50. Isto tako glavni neprijatelj ima 10 životnih bodova, i u igri je 3 puta veći nego svi ostali neprijatelji. Također ima i skriptu koja upravlja što će se desiti kada glavni neprijatelj umre. Jako jednostavna skripta koja ima poveznicu na životne bodove neprijatelja te kada oni padnu na 0 pozove se funkcija gameOverUI() iz skripte za PauseMenu, koju ću objasniti u nastavku rada.

```
public class GlavniNeprijatelj : MonoBehaviour
{

    public NeprijateljHealth health;
    public PauseMenuSkripta skripta;
```

```
    private void Start()
    {
        health = GetComponent<NeprijateljHealth>();
    }


    private void Update()
    {
        if (health.trenutniHealth <= 0)
        {
            skripta.gameOver();
        }
    }
}
```

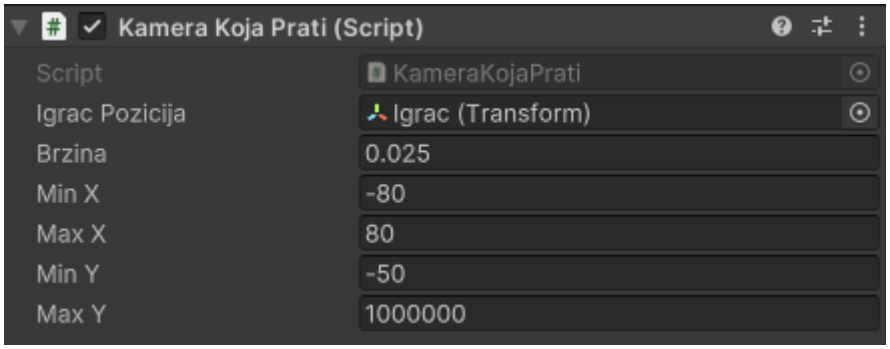


Slika 23 Varijable glavni neprijatelj

8. Implementacija ostalih skripti

8.1. Kamera koja prati igrača

Skripta KameraKojaPrati omogućava kameri da prati poziciju igrača unutar određenih granica. Na početku igre, u metodi Start, pozicija kamere se postavlja na trenutnu poziciju igrača (igracPozicija). U metodi Update, skripta ažurira poziciju kamere svakim frame-om igre kako bi pratila kretanje igrača. Kamera se pomiče prema poziciji igrača koristeći funkciju Vector2.Lerp, koja omogućava glatko pomicanje kamere s obzirom na zadanu brzinu (brzina). Ova funkcija interpolira između trenutne pozicije kamere i ciljne pozicije igrača, čime se postiže glatko praćenje. Da bi se osiguralo da kamera ne izađe iz okvira igre, koristi se funkcija Mathf.Clamp koja ograničava poziciju kamere unutar definiranih granica (minX, maxX, minY, maxY). Na taj način, iako kamera prati igrača, ona ostaje unutar zadatih granica, sprječavajući neželjene prikaze područja izvan igrivog područja.



Slika 24 Kamera koja prati igrača

```
public class KameraKojaPrati : MonoBehaviour
{
    public Transform igracPozicija;
    public float brzina;

    public float minX;
    public float maxX;
    public float minY;
    public float maxY;
```

```
    // Start is called before the first frame update
    void Start()
    {
        transform.position = igracPozicija.position;
    }

    // Update is called once per frame
    void Update()
    {
        if (igracPozicija != null)
        {
            float ograniceniX = Mathf.Clamp(igracPozicija.position.x,
                                                                    minX,
maxX);
            float ograniceniY = Mathf.Clamp(igracPozicija.position.y,
                                                                    minY,
maxY);
            transform.position = Vector2.Lerp(transform.position,
                                new Vector2(ograniceniX, ograniceniY),
brzina);
        }

    }
}
```

8.2. Coin Manager

Skripta CoinManager upravlja brojem novčića u igri i osigurava da se stanje novčića očuva između različitih scena. CoinManager koristi Singleton obrazac kako bi osigurao da postoji samo jedna instanca ove skripte u igri. U metodi Awake, skripta provjerava je li instanca već postavljena. Ako nije, postavlja trenutnu instancu kao jedinu i osigurava da se ne uništi prilikom učitavanja novih scena pomoću DontDestroyOnLoad. Ako već postoji instanca CoinManager, nova instanca se uništava. Metoda UpdateCoinText ažurira prikaz broja novčića na ekranu. Ako je coinText referenca na TextMeshProUGUI komponentu postavljena, tekst se ažurira s trenutnim brojem novčića. Metoda AddCoins omogućava dodavanje određenog broja

novčića. Nakon dodavanja novčića, metoda UpdateCoinText se poziva kako bi se osvježio prikaz broja novčića na ekranu.

```
public class CoinManager : MonoBehaviour
{
    public static CoinManager instance;

    public int coinBrojac;
    public TextMeshProUGUI coinText;

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }

    private void Start()
    {
    }

    public void UpdateCoinText()
    {
        if (coinText != null)
        {
            coinText.text = ": " + coinBrojac.ToString();
        }
    }

    public void AddCoins(int amount)
    {
        coinBrojac += amount;
        UpdateCoinText();
```

```
    }
}
```

8.3. Main Menu

Skripta MainMenu upravlja prikazom glavnog izbornika i izbornika opcija u igri te omogućava osnovne funkcionalnosti poput pokretanja igre i izlaska iz aplikacije. U metodi Start, koja se poziva kada se scena učita, skripta postavlja vidljivost glavnog izbornika na true, dok se izbornik opcija skriva. To se postiže pozivom metode ShowMainMenu. Metoda PlayGame omogućava igraču da započne igru tako što učitava sljedeću scenu u redoslijedu prema indexu trenutne scene. To se ostvaruje putem SceneManager.LoadScene, koji koristi indeks trenutne scene povećan za jedan, čime se prelazi na sljedeću scenu u build listi. Metoda QuitGame omogućava igraču da izađe iz igre. Ovo se postiže pozivom Application.Quit(), koja zatvara aplikaciju kada se igra pokrene izvan Unity editora, dok u editoru ovu funkciju možete testirati koristeći Unity Editor opcije. Metode ShowMainMenu i ShowOptionsMenu upravljaju vidljivošću korisničkog sučelja. ShowMainMenu aktivira glavni izbornik i deaktivira izbornik opcija, dok ShowOptionsMenu čini suprotno – aktivira izbornik opcija i skriva glavni izbornik. Main Menu slika u pozadini je kreirana pomoću ChatGPT-a. Unešena je tema igrice te je on kreirao sliku na zahtjev.

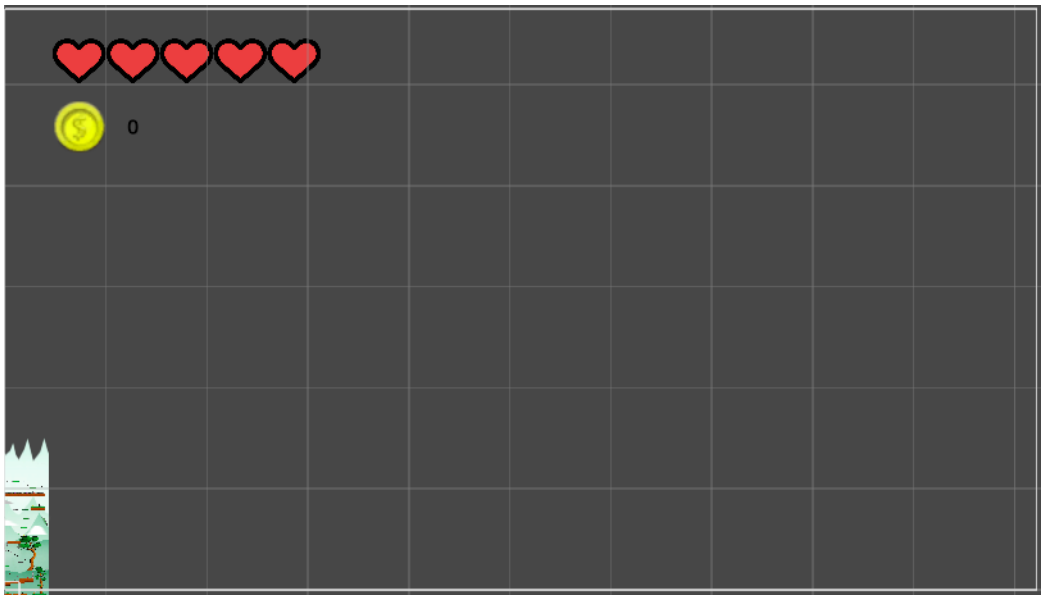
```
public class MainMenu : MonoBehaviour
{
    public GameObject mainMenuButtons;
    public GameObject optionsMenu;

    void Start()
    {
        ShowMainMenu();
    }

    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex+1);
    }

    public void QuitGame()
    {
```

```
        Application.Quit();
    }

    public void ShowMainMenu()
    {
        mainMenuButtons.SetActive(true);
        optionsMenu.SetActive(false);
    }

    public void ShowOptionsMenu()
    {
        mainMenuButtons.SetActive(false);
        optionsMenu.SetActive(true);
    }
}
```

8.4. Pause Menu

Skripta PauseMenuSkripta upravlja pauziranjem igre, prikazom završnog ekrana i opcijama za ponovno pokretanje igre, povratak na glavni izbornik ili izlazak iz igre. Aktivira se kada igrač pritisne tipku Escape. Ako je igra već pauzirana (jePauziran je true), metoda ResumeGame se poziva kako bi se igra nastavila. Ako igra nije pauzirana, metoda PauseGame se poziva da bi se igra pauzirala. Dok je igra pauzirana, gameOverUI se aktivira i vrijeme igre se zaustavlja postavljanjem Time.timeScale na 0. Ako je igrač mrtav (jeMrtav je true), dugme za nastavak igre (resumeButton) se onemogućava, inače ostaje aktivno. U metodi Update, skripta kontrolira stanje kursora. Kada je gameOverUI aktivan, kursor postaje vidljiv i slobodan je za kretanje. Kada gameOverUI nije aktivan, kursor je sakriven i zaključan u sredini ekrana. Metoda gameOver aktivira gameOverUI i zaustavlja igru, a stanje dugmeta za nastavak igre postavlja u skladu s time je li igrač mrtav ili ne. Metoda Restart ponovo učitava trenutnu scenu, vraća igru u aktivno stanje (Time.timeScale na 1), te resetira stanje pauze i smrti. Metoda MainMenu vraća igrača na glavni izbornik i ponovo pokreće igru (Time.timeScale na 1). Metoda QuitGame zatvara igru.

Slika 25 Pause Menu



Slika 26 Canvas prikaz životnih bodova i coinsa

```
public class PauseMenuSkripta : MonoBehaviour
{

    public GameObject gameOverUI;
    public Button resumeButton;
    private bool jePauziran;
    public bool jeMrtav;
    // Start is called before the first frame update
    void Start()
```

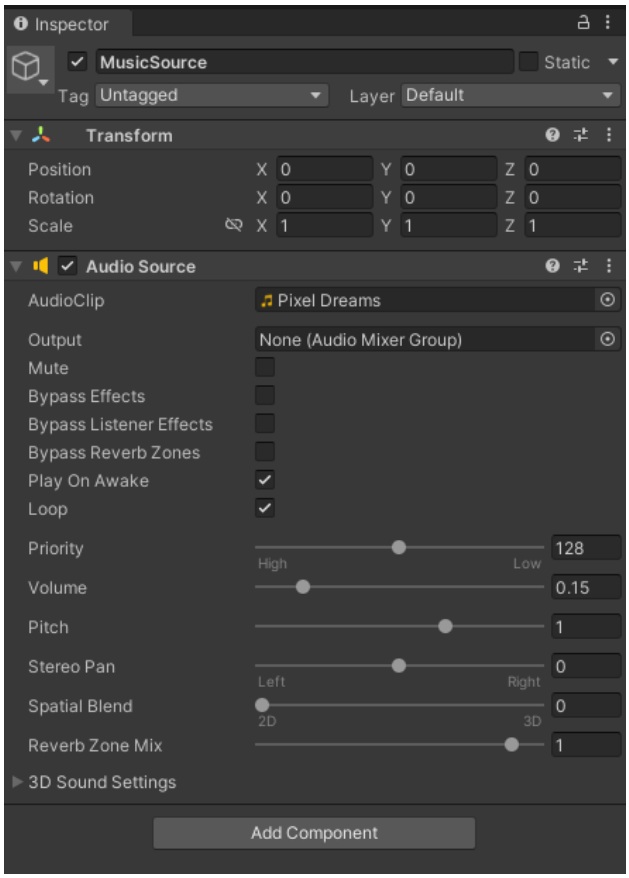
```
{
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
    gameOverUI.SetActive(false);
    jeMrtav = false;
    jePauziran = false;
}

// Update is called once per frame
void Update()
{

    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (jePauziran)
        {
            ResumeGame();
        }
        else
        {
            PauseGame();
        }
    }


    if (gameOverUI.activeInHierarchy)
    {
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
    }
    else
    {
        Cursor.visible = false;
        Cursor.lockState = CursorLockMode.Locked;
    }


}

public void PauseGame()
```

```
{
    gameOverUI.SetActive(true);
    Time.timeScale = 0f;
    jePauziran = true;

    if (jeMrtav)
    {
        resumeButton.interactable = false;
    }
    else
    {
        resumeButton.interactable = true;
    }

}

public void ResumeGame()
{
    gameOverUI.SetActive(false);
    Time.timeScale = 1f;
    jePauziran = false;
}


public void gameOver()
{
    gameOverUI.SetActive(true);
    Time.timeScale = 0f;
    if (jeMrtav)
    {
        resumeButton.interactable = false;
    }
    else
    {
        resumeButton.interactable = true;
    }
}

public void Restart()
{
```

```
        Time.timeScale = 1f;
        jePauziran = false;
        jeMrtav = false;
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void MainMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("MainMenu");
    }

    public void QuitGame()
    {
        Application.Quit();
    }
}
```

8.5. Sound Manager

SoundManager skripta upravlja reprodukcijom background muzike u igri. Također koristi Singleton obrazac kako bi se osiguralo da postoji samo jedna instanca SoundManager objekta. Backgroudnd pjesma je postavljena na prvoj sceni tj. na sceni MainMenu, te je opcija Loop uključena kako bi muzika ostala svirati sve dok je igra upaljena. Singleton obrazac omogućava mi da promjenim scenu bez da izgubim muziku. Muzika je kreirana kroz Microsoft Copilot aplikaciju, korištenjem Suno plugina kako bi se kreirala muzika kroz prompt.



Slika 27 Sound Manager background muzika

```
public class SoundManager : MonoBehaviour
{
    public static SoundManager instance {  get; private set; }
    private AudioSource audioSource;

    // Start is called before the first frame update
    private void Awake()
    {
        audioSource = GetComponent<AudioSource>();

        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else if (instance != null && instance != this)
        {
            Destroy(gameObject);
        }
```

```
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void PlayAudio(AudioClip _zvuk)
    {
        audioSource.PlayOneShot(_zvuk);
    }

}
```

9. Zaključak

Razvoj ovog završnog rada omogućio mi je detaljan uvid u procese i tehnike izrade video igara, s naglaskom na kreiranje 2D vertikalnog platformera. Kroz rad na ovom projektu, stekao sam značajno iskustvo u primjeni Unity alata i tehnika, te u razumijevanju kompleksnosti koje dolaze s razvojem igara. Izrada vlastite 2D vertikalne platforme bila je izazovan, ali vrlo poučan proces. Unity je pružio snažan skup alata koji su olakšali ovaj posao, omogućujući mi da se fokusiram na kreativne aspekte dizajna igre, uključujući razvoj mehanike kretanja, implementaciju neprijatelja, dizajn nivoa, i integraciju zvučnih efekata.

Kroz ovaj rad, naučio sam mnogo o upravljanju igračem, uključujući njegovo kretanje, skakanje, i interakciju s okolinom. Također, istražio sam kako dizajnirati razine koje su izazovne i zanimljive za igrače, uključujući postavljanje prepreka i neprijatelja na način koji doprinosi dinamičnom igranju.

Animacija je također bila ključni jako zanimljivi dio ovog projekta. Kroz rad s Animator komponentom u Unity-ju, stvorene su fluidne i realistične animacije za glavnog lika, neprijatelje i razne objekte u igri.

Kroz razvoj 2D vertikalnog platformera u Unity-ju, stekao sam dublje razumijevanje razvoja video igara i procesa stvaranja uzbudljivih i interaktivnih iskustava za igrače. Ovaj rad mi je pružio dragocjeno iskustvo i inspiraciju za daljnje istraživanje i razvoj u području video igara te pokazao kako Unity može biti moćan alat za ostvarivanje kreativnih ideja.

Popis literature

- Programski alat Unity (2024.) Preuzeto 15. siječnja 2024. sa https://unity.com/
- Microsoft Visual Studio 2022 (2024.) Preuzeto 15. siječna 2024 sa https://visualstudio.microsoft.com/vs/
- Unity Asset Store (2024.) Preuzeto 21.7.2024. sa https://assetstore.unity.com/
- Photopea (2024.) Preuzeto 30.8.2024. sa https://www.photopea.com/
- NewsRoom (2023.) A brief history of the platformer. Preuzeto 25.7.2024. s https://newsroom.activisionblizzard.com/p/a-brief-history-of-the-platformer
- PcMag (2024.) The Best Platformers for 2024. Preuzeto 2.9.2024. s https://www.pcmag.com/picks/the-best-platformers
- RedBull (2017.) The evolution of platform games in 9 steps. Preuzeto 27.7.2024. s https://www.redbull.com/in-en/evolution-of-platformers
- ChatGpt (2024.) Background slika za Main Menu. Preuzeto 3.9.2024 s https://chatgpt.com/
- Microsoft Copilot (2024.) Background muzika. Preuzeto 3.9.2024. s https://www.microsoft.com/en-us/copilot-app/?form=MY02E2&OCID=MY02E2&ch=1
- Unity Asset Store (2024.) 2D Hand Painted Platformer Environment. Preuzeto 23.1.2024. s https://assetstore.unity.com/packages/2d/environments/2d-hand-painted-platformer-environment-227159
- Unity Asset Store (2024.) Dragon Warrior (Free). Preuzeto 29.8.2024. s https://assetstore.unity.com/packages/2d/characters/dragon-warrior-free-93896
- Unity Asset Store (2024.) Pixel Starter Pack by Gamertose. Preuzeto 4.9.2024. s https://assetstore.unity.com/packages/2d/environments/pixel-starter-pack-by-gamertose-168809
- Unity Asset Store (2024.) RPG Essentials Sound Effects - FREE!. Preuzeto 4.9.2024 s https://assetstore.unity.com/packages/audio/sound-fx/rpg-essentials-sound-effects-free-227708

Popis slika