Izrada video igre žanra Metroidvania u programskom alatu Unity

Rožić, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Rožić

IZRADA VIDEO IGRE ŽANRA METROIDVANIA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Rožić

JMBAG: 0016129859

Studij: Informacijski sustavi

IZRADA VIDEO IGRE ŽANRA METROIDVANIA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor/Mentorica:

Prof. dr. sc. Danijel Radošević

Varaždin, srpanj 2020.

Luka Rožić

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

Glavni predmeti proučavanja u ovom radu su video igre žanra Metroidvanie. Prvo će se ukratko reći o povijesti žanra, tada će se opisati što igru čini Metroidvaniom. Za kvalitetno opisivanje elemenata tog žanra, ukratko će se u radu analizirati postojeće Metroidvania igre. Nadalje će se navesti programski alat za razvijanje video igara Unity i Microsoft Visual Studio za izradu skripata. Prije nego li se krene sa samom izradom video igre, dat će se osnovne ideje kako će video igra izgledati, koji su ciljevi te igre te koje sve moći i sposobnosti glavni protagonist ima kojim igrač upravlja. Važno je za napomenuti da postupci izrade video igre unutar Unityja i pisanje kôda unutar Microsoft Visual Studia budu u radu prikazani i pojašnjeni. Na kraju će se donijeti zaključak o izrađenoj igri, a time i o cijelome radu.

Ključne riječi: video igre, Metroidvania, Unity, Visual Studio, C#, dizajn i razvoj

Sadržaj

1. Uvod

Industrija video igara je svakoga dana sve veća. Kako više nije slučaj da jedna obitelj ima samo jedno ili nijedno računalo nego više, veći je broj korisnika koji koriste razne internetske usluge, programe pa tako i video igre, stoga sveukupni broj igrača, uključujući i one koji posjeduju kućne i prijenosne igraće konzole, raste. Poznati veliki proizvođači video igara poput Ubisofta i Nintenda te mali studiji koji nezavisno od tih velikih razvijaju video igre su prepoznali da taj rast ujedno znači da je i potraživanje za različitim žanrovima video igara velika, što dalje znači da bi izradom igara pojedinog žanra za kojega zanimacija postoji ostvarili profit. No, upravo zbog velikog broja konkurencije se nije lako probiti u tržište kao novoosnovani studio za izradu igara, zato je potrebno imati kreativne ideje, koncepte i na kraju krajeva igru koja se ističe od drugih igara i time biti prepoznat od strane zajednice igrača u svijetu. Jedan od žanrova kod kojega kreativnost može doći do velikog izražaja jest Metroidvania.

Metroidvania video igre su poznate po svojoj kvaliteti, pažljivo osmišljenom dinamičnom svijetu, atmosferi koja igrača ne ostavlja ravnodušnim te zabavnom i izazovnom igrivošću (engl. *gameplay*), što je sve odlika vrlo kvalitetne video igre. Stoga će se u početku rada opisati kako je nastao ovaj žanr. Nakon toga će se detaljno opisati svaki element koji je važan za žanr i navest će se video igre iz tog žanra za bolje objašnjenje samih elemenata poput atmosfere. Nakon što je jasno što je zapravo Metroidvania, potrebno je definirati koji se programski alati koriste za razvoj takvih igra, u ovom slučaju će se govoriti o Unityju kao alatu kojega nezavisni razvojni tim (engl. *indie developers*) video igara najviše koriste za izradu igara takve vrste. Opisat će se ukratko i Microsoft Visual Studio koji služi za pisanje kôda, odnosno pravila po kojima će igrica pravilno raditi. Tada dolazi opis vizije video igre koja će se izraditi u Unity-u, a nakon tog slijedi glavni dio rada - izrada video igre koristeći Unity i Visual Studio. Na kraju će se iznijeti zaključak o radu.

S obzirom da ja sam volim igrati video igre raznih žanrova te su mi Metroidvanie jedne od najdražih, odlučio sam uzeti ovu temu jer mi je oduvijek bila želja izraditi vlastitu, ozbiljniju video igru, a ono što mi je također od velike važnosti jest da ću mnogo toga isprogramirati koristeći C# jezik i time svoje znanje u programiranju unaprijediti i steći iskustvo u korištenju poznatih i složenih alata poput Unityja i Microsoft Visual Studija kojega sam odabrao za ovaj rad jer sam ga upoznao i koristio u kolegiju „Programsko inženjerstvo“.

2. Metode i tehnike rada

Za izradu ovoga rada su bili korišteni izvori različitih vrsta i oblika. Proučile su se knjige, poput onih u kojima je opisana povijest video igara, žanrovi video igara, koje sve metode i tehnike je poželjno koristiti prilikom izrade video igre te što imati na umu prije i tijekom izrade same igre da bi se igračima video igra svidjela i da bi im bila zabavna za odigrati. Osim knjiga koristile su se i brojne web stranice koje opisuju Metroidvaniu kao žanr, forumi i video zapisi na YouTubeu koji na primjerima i problemima pokazuju kako se kvalitetno služiti Unityjem te kako sa C#-om napisati pojedinu mehaniku igre. Isto tako su se odigrale i brojne Metroidvania igre za bolje razumijevanje što čini igru pravom Metroidvaniom.

3. Metroidvania kao žanr video igara

U stoljetnoj povijesti filmske industrije nastali su različiti filmski žanrovi. Tako postoje dokumentarni i igrani filmovi, a igrani filmovi se dalje dijele recimo na horor, komedije, drame i trilere. Kako u filmskoj industriji postoji ta podjela filmova na žanrove, tako slično i u industriji video igara kod koje je tijekom nekoliko desetljeća nastalo različitih žanrova kao što su platformeri, avanture, pucačine iz prvoga lica, akcijske igre, akcijske avanture i tako dalje. Metroidvania je jedan od žanrova video igara, točnije podžanr akcijskih avantura, a kako je točno nastala, govorit će se o tome u sljedećem poglavlju.

3.1. Povijest Metroidvanie

Tijekom osamdesetih godina prošloga stoljeća je japanska tvrtka Nintendo popularizirao platformere kao žanr sa svojom igrom *Super Mario Bros. (1985.)* za kućnu konzolu *Nintendo Entertainment System* (NES) [1]. Godinu dana kasnije na istoj konzoli izdali su akcijsku avanturu *The Legend of Zelda* [2]. Te dvije igre su vrlo važne jer se na temelju mehanika iz tih igara razvio i pustio 1986. na tržište *Metroid* [3], igra koja je postavila temelje za ovaj žanr. Razvojni tim te igre je želio učiniti nešto novo i drugačije od onog što je prva *Super Mario Bros*. igra nudila, a to je detaljno istraživanje 2D svijeta te drugačije upravljanje protagonistom, kao što je nagli zastoj protagonista nakon što igrač više ne drži gumb za kretanje lijevo ili desno [3]. Tijekom tzv. 16-bitne ere je za kućnu konzolu *Super Nintendo* izašao *Super Metroid (1994.)* koji je žanru posvetio veću pažnju na priču igre [4, str. 4-5].

Godine 1997. izašla je igra koju se može odigrati na više različitih konzola (engl. *multiplatform video game*) pod nazivom *Castlevania: Symphony of the Night* koju je izradila japanska tvrtka Konami [5]. Igra je uvelike defininirala žanr tako da je predstavila elemente koje sadrže igre uloga (engl. *role-playing games*), kao što je podizanje razine karakteristika lika (engl*. level up*) nakon što prikupi određeni broj iskustvenih bodova (engl*. experience points*, kraće EXP ili XP). Te zadnje dvije spomenute igre su definirale žanr i tako je nastao naziv Metroidvania kao spoj naziva Metroid i Castlevania. Ostale igre iz spomenutih serijala su *Metroid: Zero Mission (2004.)*, *Metroid: Fusion (2002.)*, *Metroid: Samus Returns (2017.)*, *Castlevania: Aria of Sorrow (2003.)* [5] [6].

Žanr su dodatno popularizirale igre koje razvijaju nezavisni razvojni timovi. Neki od poznatijih naslova su *Guacamelee!, Axiom Verge* te vrlo cijenjeni *Ori and the Blind Forest* i *Hollow Knight* čiji se umjetnički stil ne temelji na pikselastoj umjetnosti (engl. *pixel art*) nego su inspirirani rukom nacrtanim animiranim filmovima [8].

4. Elementi Metroidvanie

Ako bi se govorilo o računalnoj grafici, igre Metroidvania žanra su uvijek dvodimenzionalne (2D) ili pseudo-3D (2.5D). Pojam 3D Metroidvania ne postoji, no ako bi se moralo uzeti neku igru koja bi ispunjavala gotovo sve karakteristike žanra premda je u 3D-u, tada bi to bila igra *Metroid Prime: Trilogy*. Isto tako, kada bi razvojni tim (engl. *developer*) želio razviti Metroidvania igru, mora točno znati sve karakteristike sljedećih elemenata:

- gameplay i svijet
- atmosfera, glazba i priča
- neprijatelji, glavni neprijatelji, mini glavni neprijatelji i izazovi

Na taj način se mogu odrediti i karakteristike drugih žanrova, no kada je Metroidvania u pitanju, svaki element se može vrlo detaljno objasniti jer te igre, iako 2D, nisu površne te postoje razne „filozofije“ što čini igru Metroidvaniom. Sada će se ponešto reći za svaku od tih karakteristika.

4.1. Gameplay (igrivost) i svijet

Težište svake igre ovog žanra jest istraživanje svijeta igre. Protagonist kojim igrač upravlja u početku igre nije snažan, nedostaje mu iskustvo u borbi s neprijateljima, nema posebne predmete i oružja koji bi mu pomogli s borbom protiv neprijatelja te kod savladavanja raznih prepreka, moći koje bi mu omogućile pristup lokacijama do kojih zbog raznih razloga u ranijim dijelovima igre nije u mogućnosti pristupiti. Igrač postepeno otkriva svijet, dobiva nove moći te je sve jači kada sve više napreduje kroz igru.

Ovakve igre se vremenski brzo mogu proći, u prosjeku traju oko 6 sati. S obzirom da su kratke, poznate su po tome da teže biti vrlo upečatljive igračima tako da je svako mjesto, odnosno soba (engl. room) po nečemu jedinstvena. Primjerice, jedna soba igre može sadržavati veliko stablo s neobičnim bojama pa se po tom stablu igrač uspinje, druga soba može tražiti preciznost igrača da mu protagonist ne pogine prilikom preskakanja vatre, treća soba može biti zapamćena po pikselastoj umjetnosti kojemu se igrač može samo diviti.

S obzirom da se veliki dio Metroidvanie temelji i na vraćanju u prijašnje posječene sobe (engl. *backtracking*), može nastati kod igrača osjećaj monotonosti kod igre jer žele iskusiti nešto novo. Igra *Hollow Knight* je taj problem riješila na način da je te već posječene sobe (najviše se to odnosi na sobe koje je igrač prvi put posjetio u početku igre) estetski promijenila te zamijenila stare neprijatelje jačima i/ili novima. U nastavku slijede dvije slike, gdje slika 1. prikazuje mjesto „Forgotten Crossroads“ koju igrač posjeti na početku igre, dok slika 2.

prikazuje to isto mjesto, samo što je izmijenjeno zbog određene zaraze koja se događa unutar priče igre i sadrži jače neprijatelje.

Slika 1. Hollow Knight: Forgotten Crossroads

Slika 2. Hollow Knight: Infected Crossroads

Ono što igre ovog žanra imaju, što su preuzele od *The Legend of Zelda* serijala su novi predmeti i moći. *Metroid* igre imaju primjerice svemirski skok (engl. *space jump*) koje glavnoj protagonistici *Samus Aran* omogućavaju viši skok od uobičajenog, ledena zraka (engl. *ice*

beam) koja zaleđuje neprijatelje, gravitacijsko odijelo (engl. gravity suit) pomoću kojega je moguće normalno hodati i skakati u vodi kao i izvan nje. Kod *Castlevania* serijala su to moći poput pretvorbe u šišmiša, pri čemu je tada moguće doći do svih dijelova igre.

Imajući na umu da je istraživanje kod ovakvih igara najbitniji element, važne su i karte pomoću kojih igrači vide gdje se nalaze, koje sobe su posjetili, koje stvari nisu pokupili u tim sobama te, ovisno o igri, gdje se nalaze glavni neprijatelji. Uz kartu razvojni tim najčešće odmah stavi i statistike protagonista, dakle koliko ima života, koliko je snažan, koje predmete i moći trenutno posjeduje te stavi i postavke igre kao što je mijenjanje glasnoće glazbe i slično. Slika 3. prikazuje primjer karte koju se može vidjeti u igri *Castlevania: Symphony of the Night*.

Slika 3. Casltevania: Symphony of the Night karta [9]

4.2. Atmosfera, glazba i priča

Atmosfera je uz gameplay najvažniji dio Metroidvanie te odličnu atmosferu nije lako postići. Za duboku atmosferu potrebno je više čimbenika uzeti u obzir, među kojima je i umjetnički stil (engl. *art style*) kojeg se odabire najčešće prije početka razvoja igre. Uz odabrani umjetnički stil potrebno je skladati prikladnu glazbu koja bi postavila uz priču određeni ton igre, bila to epska igra s epskom pričom i skladbom, žalosna i melankolična igra ili igra koja se

najviše temelji na izolaciji protagonista na nekom udaljenom i nesigurnom mjestu. Vrlo često se kod ovih igara ne koriste dijalozi između likova i tekstovi nego pristup da se igračima jednostavno pokaže neki događaj unutar same igre. Takav pristup je poznat pod nazivom „pokaži, nemoj reči“ (engl. *show, don't tell*). Mnoge Metroidvanie imaju minimalističke priče te stavlja se naglasak najviše na gameplay.

4.3. Neprijatelji, glavni neprijatelji, mini glavni neprijatelji i izazovi

Poželjno je da igre imaju raznovrsne neprijatelje koji predstavljaju izazov igraču. Na početku igre stavljaju se neprijatelji koji ne stvaraju veliku štetu igraču jer se igrač još upoznaje s mehanikama video igre, no kasnije dolaze neprijatelji koji imaju bolju isprogramiranu umjetnu inteligenciju (engl. *artificial inteligence*, skraćeno AI) te predstavljaju izazov igraču. Takvi neprijatelji su najčešće glavni neprijatelji koji su po izgledu i načinu napadanja jedinstveni. Neprijatelji koji su složeniji od običnih neprijatelja, ali najčešće ne toliko kao i glavni neprijatelji su mini glavni neprijatelji koji igrače drže zainteresiranima za igru te testiraju njihove vještine igrača prije nego dođu do glavnih neprijatelja.

Ostali izazovi s kojima se igrači mogu susresti su najčešće prepreke poput lave, oštrih igli, lasera, leda, otrova, mraka i tako dalje. Vrlo veliki izazov igračima kod ovih igara je i snalaženje po velikom svijetu, kuda moraju ići nakon što nešto odrade te prisjetiti se koje mjesto sadrži put za novu destinaciju do koje prije nisu mogli doći nakon što su dobili moć npr. skakanja po zidu.

5. Unity i Visual Studio

Za izradu video igre je potreban snažan programski alat, odnosno engine (hrv. motor/ pokretač). Nekada su developeri (hrv. razvojni timovi) morali unutar studija razviti svoj vlastiti engine i tek tada ići razvijati igru. Iako i danas velike organizacije i dalje razvijaju vlastite engine za igru, mnogi su počeli koristiti engine koji je netko drugi razvio. Najpoznatiji današnji enginei su Unity i njegova konkurencija Unreal Engine. Prvi se temelji na strukturiranom objektno orijentiranom jeziku C# (C Sharp) kojeg je razvila tvrtka Microsoft, dok se drugi temelji na objektno orijentiranom jeziku C++ [10]. Za izradu ovog rada se koristi Unity, ponajviše zato jer su današnje najpoznatije Metroidvanie *Ori and the Blind Forest* te *Hollow Knight* razvijene u njemu i zato što se Unity najviše koristi za 2D igre, dok Unreal Engine koji je u grafičkome dijelu moćniji od Unityja za 3D.

5.1. Unity

Unity je programski alat koji se koristi najviše za izradu video igara za kućne konzole, računala, mobilne uređaje te za uređaje koji omogućavaju virtualnu stvarnost (VR). Zbog svoje fleksibilnosti, Unity se također koristi za izradu filmova i animacija, kod arhitekture i inženjerstva za izradu modela, kod automobilske industrije koristi se za razne simulacije i slično [11] [12].

Sam program su razvili David Helgason, Nicholas Francis i Joachim Ante u Danskoj 2005. kada su razvijali igru *GooBall*. Iako igra nije bila uspješna na tržištu, primijetili su da programski alat kojeg su razvili ima mnogo potencijala, stoga su ga krenuli unapređivati jer su htjeli razviti takav engine koji bi bio pristupačniji od konkurentnih proizvoda [13]. Danas je jedan od vodećih enginea kojega održava tvrtka Unity Technologies te ga koriste mnogi nezavisni developeri za razvoj kako 2D, tako i za 3D igara za različite uređaje.

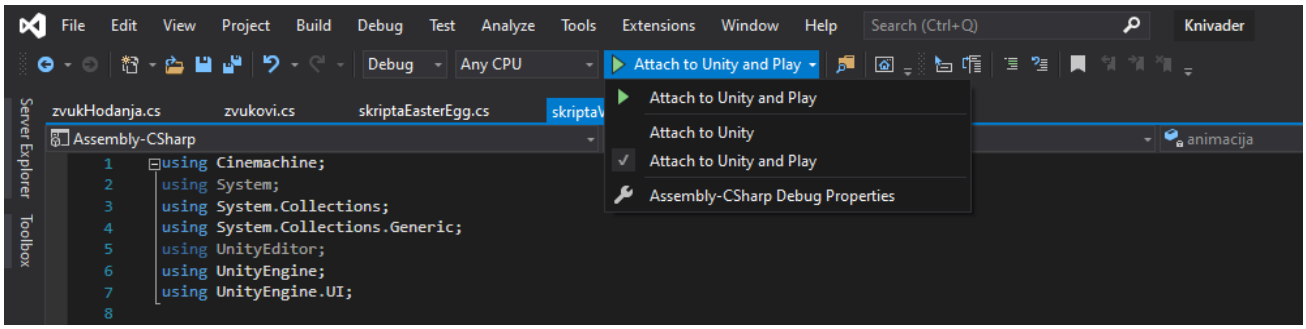
Kao i na svakom engineu, mogu se u Unityju dodavati različite slike i modeli koji mogu poslužiti kao pozadina ili kao protagonist u igri, moguće je dodati različite efekte, osvjetljenja, glazbe, grafička sučelja, animacije, kameru te složenu fiziku unutar igre. Često dolaze nove verzije Unitya koje sadrže riješene probleme prethodnih verzija, dakle može se zaključiti da se taj alat i dalje razvija i unaprjeđuje [14]. Ono što će biti novo kod Unityja u bližoj budućnosti jest da će imati besplatan *Bolt visual scripting* [15], što je zapravo vizualan način skriptiranja, dakle tu se kôd ne piše kao kod većine programa za razvoj softvera nego se metode, selekcije, varijable i objekti slažu, povezuju na vizualan način kao i kod „blueprinta“ koji je također dodatak koji omogućava vizualno skriptiranje unutar Unreal Enginea. Iako takav način

skriptiranja bi olakšalo do neke mjere izradu skripata za igru, za ovaj rad se koristio „tradicionalan“ način skriptiranja koristeći Microsoft Visual Studio.

5.2. Microsoft Visual Studio

Za igru da bi bila igriva, da bi se likovi pritiskom na pojedinu tipku mogli kretati, da bi mogli dobivati ozljede ako dotaknu neprijatelja, potrebno je napraviti skripte, odnosno programski kôd koji bi to sve omogućio. Unity podržava skripte napisane u programskom jeziku Boo, UnityScript (poznat kao i dijalekt JavaScripta za Unity) i C# [16]. Najčešće se koristi C# za skriptiranje, stoga se odabrao Microsoft Visual Studio u kojem se može pisati kôd koristeći C# programski jezik.

Visual Studio je integrirano razvojno programsko okruženje (engl. *integrated development environment*, IDE) kojega je razvio Microsoft te služi za izradu web stranica, aplikacija za mobilne uređaje i za računalne programe. Podržava različite programske jezike, među kojima su C, C#, C++, CSS, JavaScript , Python itd [17]. Za Visual Studio postoje razni dodaci koji se mogu instalirati, pa je tako i među njima dodatak koji sadrži alate za rad s Unityjem. Nakon instalacije tog dodatka, klikne se kod Unityja na *Edit->Preferences-> External Tools* i odabere Visual Studio 2019 (u ovom slučaju inačica Professional) u padajućem izborniku kod *External Script Editor* kako bi mogao komunicirati s Visual Studijem. Sada je moguće unutar Unityja izraditi skriptu, pisati kôd u Visual Studiju te nakon tog se ta skripta klikom na zelenu strijelu unutar Visual Studija poveže s Unityjem. Moguće je pokrenuti igru unutar Unityja preko Visual Studija ako se odabere opcija *Attach to Unity and Play* (Slika 4.).



Slika 4. Povezivanje kôda izrađenog u Visual Studiju s Unityjem te testiranje igre

5.3. Pojmovnik Unityja

Slijede pojmovi s kojima se svaki razvojni tim koji koristi Unity često susreće. Potrebno ih je objasniti kako bi nastavak ovoga rada bio razumljiv.

5.3.1. Gameobject

„Gameobject“ je temeljna klasa za sve entitete u Unityju. Drugim riječima, kada se kreira entitet, njemu se mogu dodijeliti razne komponente, dakle mogu mu se dodijeliti npr. „Rigidbody“ koji služi da taj objekt koji može biti protagonist, neprijatelj ili bilo što drugo, mora poštovati zakone fizike u igri.

5.3.2. Gameobject djeca (engl. child) i roditelj (engl. parent)

Svaki gameobject može imati nijednog, jednog ili više podređenih gameobjecta. Kaže se da je on njima roditelj (engl. *parent*) dok je podređeni „gameobject“ njegovo dijete (engl. child). Često se koriste takvi nazivi unutar kôda, stoga ih je bitno spomenuti.

5.3.3. Components

„Components“, odnosno komponente su u Unityju stvari koje se dodaju gameobjectima i time gameobject ima neku svoju ulogu unutar igre. Može mu se dodijeliti komponenta „Sprite Renderer“ koji služi da taj objekt ima vizualni prikaz u igri, odnosno da se vidi ta slika koja mu se dodijelila. Komponenta može biti skripta koja se doda kako bi se taj objekt u igri „ponašao po pravilima“.

5.3.4. Collider

„Collider“ je prostor, linija, granica koja može poslužiti kao tlo na kojem se može hodati. Može poslužiti kao okidač (engl. *trigger*), što znači da kada jedan gameobject koji ima u sebi npr. „BoxCollider2D“, „PolygonCollider2D“ ili „EdgeCollider2D“ uđe ili dotakne granice nekog drugog „collidera“, tada se u skripti preko metoda „OnTrigger()“ može napisati što će se tada dogoditi u igri.

5.3.5. Canvas

„Canvas“, u prijevodu „platno“ se koristi kao korisničko sučelje (engl. *user interface*, skraćeno UI) kao što su glavni izbornici, pa se tamo mogu dodati slike i tekstovi.

6. Izrada video igre

Sada slijedi najvažniji dio rada, a to je izrada same igre koristeći Unity, Visual studio i ostale programske alate. Prije tog će se ukratko pojasniti tematika i igrivost igre.

6.1. Tema i igrivost video igre – „Knivader“

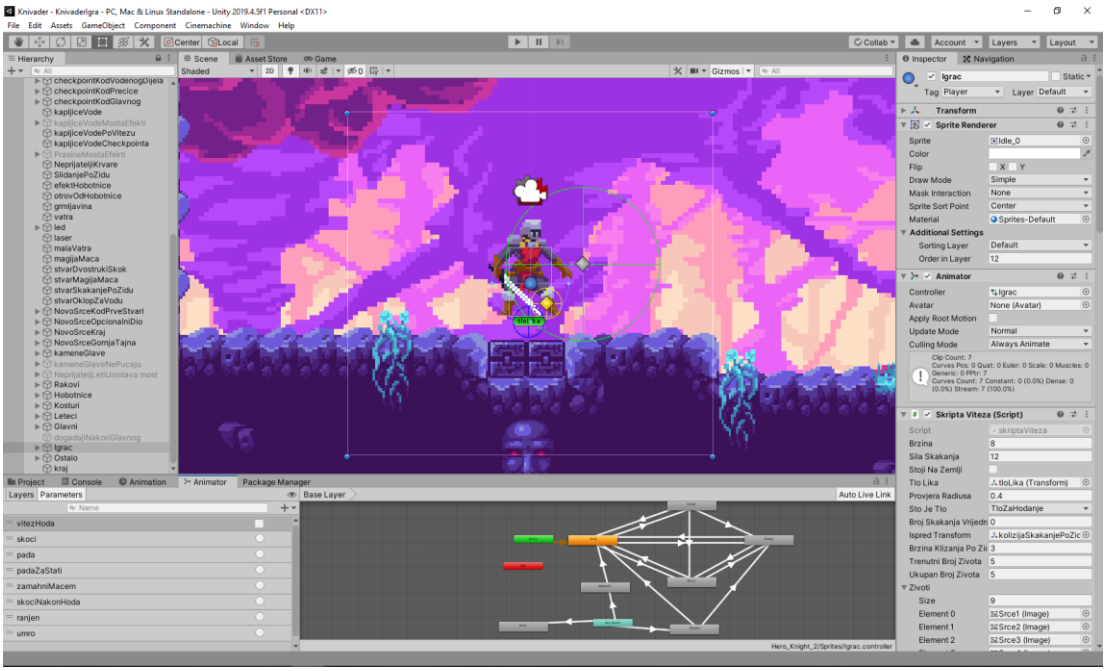
Metroidvania igra koja se izradila u Unity-u se zove *Knivader*, što je kombinacija riječi *Knight*, što u prijevodu iz engleskog jezika znači vitez i *Invader*, što znači osvajač. Glavni protagonist je vitez kojega je mađioničar zajedno s ostalim čudovištima iz fantazije teleportirao na vanzemaljski planet te je cilj viteza da nađe način da se vrati u svoj svijet.

Vitez, odnosno igrač u početku ima samo 5 života, odnosno 5 srca koja može vidjeti u gornjem lijevom dijelu svog ekrana te ima mač s kojim može napasti neprijatelje samo onda kada stoji ili se kreće na zemlji i može samo jednom skočiti dok je na zemlji. Također igrač ima mogućnost vidjeti statistiku viteza dok zaustavi igru, dakle koliko ima života, koliko ima EXP bodova te koliko mu treba do sljedećeg level up-a, kolika mu je snaga mača za stvaranje štete neprijatelju te koje predmete je putem našao. Isto tako igrač ima i kartu u kojoj vidi sve sobe koje je posjetio, u kojoj sobi se trenutno nalazi te koje sobe su međusobno povezane. Igraču je uvijek ponuđena mogućnost povratka na naslovnu scenu (engl. *title screen*) preko glavnog izbornika ili ako odustane od igre nakon što lik pogine. Predmete koje igrač pronađe tijekom istraživanja svijeta su leteće čizme pomoću kojih može još jednom u zraku skočiti, vodena medalja pomoću kojeg se može slobodno, bez uobičajenog usporavanja kretati u vodi, pandže za zid pomoću kojih se može držati i polako kliziti po zidu prema dolje te skakati po zidu, magična energija pomoću kojeg može zamahom mača stvoriti magiju koja neprijatelju prilazi kao „projektil“ te ga ozljedi, zlatno srce koje mu daje dodatni život, odnosno srce što daje igraču u pravilu više mjesta za griješenje u igri. Snaga napada viteza se poveća nakon što uništi određeni broj neprijatelja, jer na taj način sakuplja EXP bodove i level up-a se. Neprijatelje s kojima se susreće su rakovi koji hodaju lijevo desno, leteći neprijatelji koji padaju prema dolje kada im se vitez približi, hobotnice koje pucaju otrov prema vitezu kada ga ugledaju, kosturi koji služe kao mini glavni neprijatelji i glavni neprijatelj kojega je najteže ubiti. Vitez se mora čuvati i oštrih igli i lasera koje statue u obliku glave pucaju. Isto tako, kao bonus, igra sadrži i uskršnje jaje (engl. *easter egg*) kojega je teže pronaći u igri s obzirom da je dobro sakriveno. Igrač je završio s igrom kada dođe do onog dijela igre u kojoj piše „THE END“, što doslovce označava kraj igre.

6.2. Početak izrade video igre

Objasnit će se najvažnije mehanike igre „Knivader“. Kao što je već bilo rečeno, koristit će se alat Unity verzija 2019.4.5f1 Personal (u početku izrade igre je bila starija verzija, ali u međuvremenu su došla nova ažuriranja), te Microsoft Visual Studio Professional 2019 verzija 16.6.3. Za izradu i dorađivanja slika se koristio Adobe Photoshop 2020 verzija 21.0.2, dok se za pretvorbu video sadržaja s video streaming web stranica u audio koristio program 4K Video Downloader verzija 4.12.5.3670.

Prilikom pokretanja Unityja i stvaranja novog projekta, prvo se prebacilo na 2D pogled klikom na gumb „2D“. Tada se išlo preuzeti iz „Unity Asset Storea“ slike neprijatelja, viteza i okoline. Nakon toga se počelo raditi na glavnom protagonistu igre tako da se prvo dodala komponenta sprite renderer, animation na kojemu su se napravile razne animacije viteza koristeći više sličica, animator na kojemu se određuju pravila kada će se koja animacija viteza pokrenuti, collideri koji služe da bi igrač stajao na tlu ili mogu poslužiti kao okidači (engl. *trigger*) za detekciju collidera drugih gameobjecta. Na sljedećoj slici (Slika 5) se vidi sučelje programa Unity, gdje se na lijevoj strani nalazi popis gameobjecta koji su se izradili za igru, na desnome dijelu komponente objekta vitez dok se dolje vidi animator gdje se spajaju stvorene animacije za viteza. Najveću važnost u ovom radu imaju programski kôdovi za viteza, neprijatelja, predmete, glavnog izbornika i tako dalje. Sada će pristupiti programskom kôdu viteza te pojašnjenju istog.



Slika 5. Sučelje Unityja

6.3. Programski kôd igrača - vitez

Mogućnosti i mehanizmi koje vitez mora imati su već bile spomenute te će se sada po dijelovima objasniti programski kôd viteza. Bilo je nužno izložiti kôd po dijelovima kako vitez ima oko 800 linija kôda. Prvo slijedi početak skripte za viteza u kojemu su definirane sve biblioteke koje se koriste za pisanje kôda viteza:

```
using Cinemachine;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;
using UnityEngine.UI;
```

„Cinemachine“ se koristi kako bi se koristile naredbe za upravljanje kamerom u igri, „System Collections“ općenito služi kako bi se mogao pisati kôd unutar Visual Studija, dok „UnityEditor“, „UnityEngine“ i „UnityEngine.UI“ služe kako bi se moglo koristiti različitim metodama koje su važne za rad igre u Unityju. Isto tako je važno spomenuti sljedeće tri metode:

```
void Start()
void Update()
void FixedUpdate()
```

„Start()“ metoda se pokreće svaki puta kada se pokrene scena igre unutar Unityja [18]. To je prva metoda koja se u skripti nekog objekta izvršava te se tamo deklariraju pojedine varijable, tipa dodjeljivanje broja 5 varijabli „zivot“, što znači da vitez u početku ima samo 5 života. „Update()“ je metoda koja se poziva za svaki okvir (engl. *frame*) i služi najviše da se ispituje je li došlo do nekih promjena u igri [19], tipa ako je igrač došao do jednog collidera neprijatelja, izgubi jedno srce, dok se „FixedUpdate()“ koristi najviše za samu fiziku objekta kojemu je Rigidbody2D dodijeljen [20]. Sada kada su najvažnije metode objašnjene, može se započeti s fizikom igrača/lika, dakle kretanje viteza po svijetu.

6.3.1. Kretanje viteza

Nakon što se u novokreirani gameobject „Igrac“ dodaje animacija kretanja viteza, prvo što je važno za napraviti jest da se ta slika kreće lijevo kada igrač drži gumb lijeve strijele na

tipkovnici, odnosno desno ako drži gumb desne strijele. U tu svrhu se napravila skripta „skriptaVitez“ koja se dodijelila gameobjectu Igrac. U nastavku slijedi kôd te skripte napisan u Visual Studiju:

```
public class skriptaViteza : MonoBehaviour
{
        private Rigidbody2D rigidbody;
        public bool stojiNaZemlji;
        public Transform tloLika;
        public float provjeraRadiusa;
        public LayerMask stoJeTlo;

private Dictionary<KeyCode, bool> gumbiOmoguceni = new Dictionary<KeyCode,bool>();

void FixedUpdate()
  {
stojiNaZemlji = Physics2D.OverlapCircle(tloLika.position, provjeraRadiusa, stoJeTl
o);

if (Input.GetKey(KeyCode.LeftArrow) && gumbiOmoguceni[KeyCode.LeftArrow])
           {
               gumbiOmoguceni[KeyCode.RightArrow] = false;
               gumbiOmoguceni[KeyCode.LeftArrow] = true;

               moveInput = -1;
               rigidbody.velocity = new Vector2(moveInput * brzina, rigidbody.velo
city.y);

               if (stojiNaZemlji == true && sklizeSe == false)
               {
                   zvukHodanja.pokreniZvuk2("zvukTrcanja", true);
               }
               else if (stojiNaZemlji == false && sklizeSe == false)
               {
                   zvukHodanja.pokreniZvuk2("zvukTrcanja", false);
               }
               transform.localRotation = Quaternion.Euler(0, 180, 0);
               gledaDesno = true;
               animacija.SetBool("vitezHoda", true);

            }
      }
}
```

Prilikom kreiranja svake skripte se kreira „MonoBehaviour“ koji je temeljna tj. bazna klasa za svaku Unity skriptu. Nakon toga se definira Rigidbody2D koji će se koristiti za definiranje kretanja glavnoga lika. Definira se i varijabla tloLika tipa „Transform“ koji unutar FixedUpdate() ispituje dodiruje li protagonist tlo, gdje OverlapCircle provjerava dodiruje li collider gameobjecta tloLika koji je dijete glavnom gameobjectu „Igrac“ onaj sloj (engl. *layer*) koji se u ovom slučaju zove „TloZaHodanje“. U skripti nije direktno napisano da se radi o tom sloju, nego da se izbjegne „hardkodiranje“, stavila se ta varijabla kao „public“, što znači da druge skripte mogu vidjeti tu varijablu i mijenjati ju te se unutar Unityja može „drag and dropom“, tj. držanjem mišem i puštanjem toj varijabli dodijeliti određeni gameobject. Kako bi

se to ispravno provjerilo, mora se staviti trenutna pozicija gamobject-a koji provjerava tlo lika koji se nalazi u određenoj X, Y i Z-osi, krug te provjere je radijusa duljine provjeraRadiusa, te se stavi da se provjeri dodiruje li lik onaj sloj koji je definiran u varijabli stoJeTlo.

Nadalje, „Input.GetKey(KeyCode)“ služi za ispitivanje drži li igrač onaj gumb koji je definiran unutar zagrade preko „KeyCode-a“. Ako drži, vraća se vrijednost „true“. Isto tako unutar„if-a“ ispituje se je li taj gumb koji je spremljen u rječniku gumbiOmoguceni omogućen. Rječnik se ovdje koristi zbog problema koji nastaje kada igrač drži oba gumba za kretanje i vitez tada ne zna kuda bi trebao ići te dođe do problema unutar igre. Stoga kada igrač drži lijevu tipku za kretanje, u rječnik se stavlja da je lijeva tipka za kretanje omogućena, dok je desna onemogućena. Nakon toga se promijeni brzina rigidbody-a, tako da ako je smjer kretanja lijevo, tada će se igrač po X-osi kretati negativno po definiranoj brzini, dok brzina po Y-osi ostaje ista. Dalje dolaze „if-ovi“ koji ispituju stoji li igrač na zemlji te skliže li se po zidu. Ako stoji na zemlji, odnosno nije u zraku nego se kreće po zemlji, tada se aktivira zvučni efekt trčanja koji se u petlji tako dugo ponavlja dok igrač hoda po tlu. Vrlo važna linija kôda jest mijenjanje vrijednosti lokalne rotacije viteza sa „Quaternion.Euler“, gdje se mijenja rotacija gameobjecta „Igrac“ sa svom svojom djecom za 180 stupnjeva po Y-osi, odnosno za 0 ako vitez gleda na desnu stranu [21]. Ta linija je vrlo važna upravo zato jer svi collideri koji su smješteni na desnoj strani se presele na lijevu nakon što se vitez okrene na lijevu stranu i tako vitez prema pravilnoj strani napada. Na kraju se postavlja vrijednost parametra vitezHoda na true, što znači da će se u petlji ponavljati animacija koja prikazuje kako vitez hoda. Što se tiče držanja tipke za hodanje prema desnoj strani, kôd za to je vrlo sličan kao i za kretanje prema lijevoj strani, samo su vrijednosti ovoga puta suprotne, primjerice moveInput je ovoga puta jednak 1.

6.3.2. Skakanje viteza

Jedna od važnih mehanika igre jest skakanje glavnog lika te je u izradi ovog kôda pomogao video iz kanala „Blackthornprod“ [22]. U nastavku slijedi kôd unutar skripte „skriptaViteza“ koji to omogućava te se također mogu vidjeti i deklarirane varijable za klizanje po zidu:

```
private int brojSkakanja;
public int brojSkakanjaVrijednost;

bool dodirujeIspred;
public Transform ispredTransform;
bool sklizeSe;
public float brzinaKlizanjaPoZidu;

void Start()
```

```
{
brojSkakanja = brojSkakanjaVrijednost;
imaStvarSkakanjaPoZidu = false;
}
private void Update()
{
//skakanje viteza
if (Input.GetKeyDown(KeyCode.Space) && brojSkakanja > 0)
    {
        rigidbody.velocity = Vector2.up * silaSkakanja;
        brojSkakanja--;
        zvukovi.pokreniZvuk("skakanje");
    }

    else if (Input.GetKeyDown(KeyCode.Space) && brojSkakanja == 0 && stojiNaZemlji
 == true && mozeNapasti)
    {
        rigidbody.velocity = Vector2.up * silaSkakanja;
        zvukovi.pokreniZvuk("skakanje");
    }

    if (stojiNaZemlji == true)
    {
        brojSkakanja = brojSkakanjaVrijednost;
    }

}
```

U Start() metodi se definira koliko puta vitez može skočiti. Taj broj se povećava za jedan nakon što dobije stvar za dvostruki skok te se u početku definira da nema predmet za skakanje po zidu. Nadalje, unutar Update() metode se prvim „if-om“ ispituje je li pritisnut gumb „Space“ te istinitost tvrdnje da je preostao broj skakanja veći od nule. Ako je uvjet zadovoljen, mijenja se brzina (engl. *velocity*) rigidbodya tako da je nova brzina jednaka vektoru2 prema gore pomnožen brojem koji određuje koliko visoko će vitez skočiti. Vector2 se odnosi na vektor koji ima X i Y poziciju unutar igre, dok Vector3 uzima i treću, Z-os u igri, no nju se neće toliko koristiti jer se ovdje radi o 2D igri [23]. Ako prvi uvjet nije zadovoljen, tada se ispituje stoji li lik na zemlji, vrijednost varijable brojSkakanja, stoji li na zemlji te može li napasti. Ako je sve zadovoljeno, lik skače. Razlog zašto se navela varijabla „mozeNapasti“ koja je tipa „bool“ jest zato što kada igrač zamahne mačem, on se ne smije kretati za to vrijeme i ne smije odmah napasti. Kada protekne određeno vrijeme, smije napasti mačem, kretati se i skakati, pa je varijabla postavljena na „true“. Isto tako, kada lik stoji na zemlji, broj skakanja mu se vraća na 0, odnosno 1 ako ima čizme za skočiti još jednom u zraku. Vrijedi napomenuti da se svaki put pokrene zvučni efekt kada se skoči, tako da igrač dobije bolji i uvjerljiviji dojam o svijetu igre.

6.3.3. Klizanje viteza po zidu

```
if (imaStvarSkakanjaPoZidu)
        {
            if (dodirujeIspred == true && stojiNaZemlji == false && (Input.GetKey(K
eyCode.RightArrow) == true || Input.GetKey(KeyCode.LeftArrow) == true))
            {
                sklizeSe = true;
                brojSkakanja = brojSkakanjaVrijednost;
            }

            else
            {
                sklizeSe = false;
            }

            if (sklizeSe)
            {
                rigidbody.velocity = new Vector2(rigidbody.velocity.x, Mathf.Clamp(
rigidbody.velocity.y, -brzinaKlizanjaPoZidu, float.MaxValue));
                float pozicijaX = 0;
                if (gledaDesno == false)
                {
                    pozicijaX = transform.position.x + (float)0.6;
                }
                else
                {
                    pozicijaX = transform.position.x - (float)0.6;
                }
                float pozicijaY = transform.position.y - (float)0.3;
                float pozicijaZ = transform.position.z;
                Vector3 pozicija = new Vector3(pozicijaX, pozicijaY, pozicijaZ);

                if (rigidbody.velocity.y < 0)
                {
                    var slidanjeEffect = Instantiate(particleEffect, pozicija, Quat
ernion.identity);
                    Destroy(slidanjeEffect, (float)0.7);
                }

            }
        }
```

Dan je kôd za klizanje viteza po zidu također potpomognut video zapisom iz kanala „Blackthornprod“ [24], gdje se prvo provjeri ima li vitez predmet koja mu omogućuje klizanje po zidu. Ako ima, ispituje se dodiruje li on ispred sebe zid na isti način kao što se ispitivalo dodiruje li on sloj na kojem se hoda. Ako dodiruje, ako ne stoji na zemlji te ako igrač drži desnu ili lijevu strjelicu na tipkovnici (WASD tipke su onemogućene u igri), tada se vitez skliže po zidu i odmah mu se i broj koliko puta može skočiti vrati na maksimalan broj skakanja. Tada se „if“ selekcijom ponovno pita, ako se skliže, onda njegova brzina po X-osi ostaje ista, no po Y-osi se mijenja koristeći „Mathf.Clamp“ [25], gdje se prvo definira brzina klizanja po zidu koja mora biti između minimalne brzine i maksimalne, gdje se minimalnu definiralo unutar Unitya, dok je

maksimalna vrlo velik broj. Ako je brzina tog klizanja ispod minimalne, Mathf.Clamp vraća minimalnu brzinu. Ako je veća od maksimalne, vraća se maksimalna brzina. Nakon toga se definiraju X, Y i Z položaji gdje se vitez kliže kako bi se na toj poziciji instancirali, odnosno stvorili vizualni efekti prašine. Za instanciranje prašine se koristi naredba „Instantiate“ u kojoj se prvo stavlja što se želi stvoriti, gdje se želi to stvoriti u igri te pomoću „Quaternion.identity“ [26] se odredi može li se taj objekt rotirati u svijetu. Nakon toga se koristi naredba Destroy() u kojem se stavlja objekt kojeg se želi uništiti i nakon koliko sekundi se bude uništio, u ovom slučaju je ispod jedne sekunde.

6.3.4. Napad viteza mačem i magijom

Kako bi vitez mogao napasti, korištene su dvije skripte: „skriptaViteza“ i „igracNapada“. Mogu se u nastavku vidjeti kôdovi tih skripata :

```
//skriptaViteza
if (Input.GetKeyDown(KeyCode.Y) && stojiNaZemlji == true && mozeNapasti == true)
{
    mozeNapasti = false;
    animacija.SetTrigger("zamahniMacem");
    zvukovi.pokreniZvuk("napadMacem");
    Invoke("mozeNapastiBool", 0.24f);
    zvukHodanja.pokreniZvuk2("zvukTrcanja", false);
}
//igracNapada
if(stopanjeVremena <= 0)
       {
if (Input.GetKeyDown(KeyCode.Y) && GetComponent<skriptaViteza>().stoji() == true)
            {
                stopanjeVremena = zapocetoVrijemeIzmeduNapada;
                Collider2D[] krugZaNapad = Physics2D.OverlapCircleAll(pozicijaKrug
aZaNapad.position, dometKrugaZaNapad, neprijatelj);
                for (int i = 0; i < krugZaNapad.Length; i++)
                {
                    if (krugZaNapad[i].gameObject.tag == "protivnik")
                    {
         krugZaNapad[i].GetComponent<neprijateljRakAI>().smanjiZivot(snagaNapada);
                    }
                    else if (krugZaNapad[i].gameObject.tag == "protivnikLeti")
                    {
        krugZaNapad[i].GetComponent<neprijateljLetiAI>().smanjiZivot(snagaNapada);
                    }
//... još postoji nekoliko vrsta neprijatelja s drugačijim tag-ovima, no ovdje su
kao primjer uzeta samo dva
                }
            }
else
{
    stopanjeVremena -= Time.deltaTime;
}
```

Kako bi vitez napao mačem, igrač mora pritisnuti tipku „Y“, vitez mora stajati na zemlji te mozeNapasti mora biti postavljen na „true“. Ako je to slučaj, tada se pokrene okidač kojom se pokrene animacija u kojoj vitez zamahne mačem, pokrene se određeni zvuk te se „Invoke“ naredbom [27] poziva funkcija koja se zove „mozeNapastiBool“, no ta funkcija će se pozvati nakon što prođe skoro polovica sekunde. U toj funkciji se tada mozeNapasti vraća na true.

U isto vrijeme se u skripti „vitezNapada“ prvo ispituje prošlo li je dovoljno vremena da vitez može napasti. Ako je to slučaj, provjerava se je li igrač pritisnuo tipku „Y“ te preko naredbe GetComponent<skriptaViteza>().stoji() [28] dohvaća iz roditelja tog gameobjecta metoda iz skripte „skriptaViteza“ koja vraća vrijednost „true“ ako vitez stoji, odnosno „false“ ako vitez ne stoji na zemlji. Ako je cijeli uvjet ispunjen, tada se vrijeme postavlja na onoliko sekundi koliko mora proći da vitez može ponovno napasti, u polje krugZaNapad koji je tipa „Collider2D“, spremaju se svi collideri gameobjecta koji pripadaju sloju, odnosno layeru „neprijatelji“. Tada se kroz svaki collider tog polja prolazi i ispituje je li oznaka (engl. *tag*) gameobjecta tog collidera jednak određenoj oznaci, tipa da je jednak oznaci „protivnikLeti“. Ako je, onda se radi o neprijatelju koji leti i tada se dohvaća njegova skripta i poziva metoda „smanjiZivot“ u kojoj se stavlja vrijednost jednaka snazi napada viteza. Razlog zašto je bilo potrebno stavljati oznake jest što nisu svi collideri neprijateljskih objekata jednaki sloju „neprijatelji“ i tu dolazi do javljanja pogrešaka u debuggeru Visual Studija i u Unityju unutar prozora „Console“.

Što se tiče napada viteza magijom, kôd je vrlo sličan kôdu za napad s mačem, samo što se ovdje instancira novi objekt koji u sebi sadrži sliku magije, rigidbody i „Polygon Collider 2D“ kao okidač za kolizije. Svaki neprijatelj sadrži i skriptu „magijaUdaraNeprijatelje“ u kojemu se ispituje došlo li je do kolizije između tog objekta i objekta magije mača. Ukoliko je došlo, smanji život neprijatelja. Za ispitivanje je li došlo do kolizije, u ovoj igri koristili su se „OnTriggerEnter2D(Collider2D)“ [29], „OnTriggerExit2D()“ i „OnTriggerStay2D“, gdje se prvi koristi kada jedan collider uđe u drugi, drugi se izvršava kada prvi collider izađe van iz drugog, dok se treći kao petlja izvršava cijelo vrijeme dok je prvi collider unutar drugog. Za ovu situaciju je potrebno koristiti „OnTriggerEnter2D(Collider2D collision)“ te se u sljedećem jednostavnom kôdu može vidjeti primjer za to:

```
private void OnTriggerEnter2D(Collider2D collision){
        if(collision.transform.gameObject.tag == "magijaMaca")
        {
            switch (tipNeprijatelja)
            {
                case "protivnikRakMagija":
 this.transform.parent.gameObject.GetComponent<neprijateljRakAI>().smanjiZivot(2);
                    Destroy(collision.gameObject);
                    break;
        }
    }
}
```

S obzirom da se koristi ta skripta za sve neprijatelje koji imaju gameobject kao dijete koji služi za detekciju magije, potrebno je koristiti „switch...case“ izjave. Ako dođe do kolizije neprijatelja s magijom mača, tada se ispituje o kojoj vrsti neprijatelja se radi. Ako se radi o raku, onda se od roditelja tog objekta dohvaća skripta „neprijateljRakAI“ koji sadrži metodu za smanjivanje života raka. Nakon tog se uništi ta magija te se izađe iz „switch...case“ izjave.

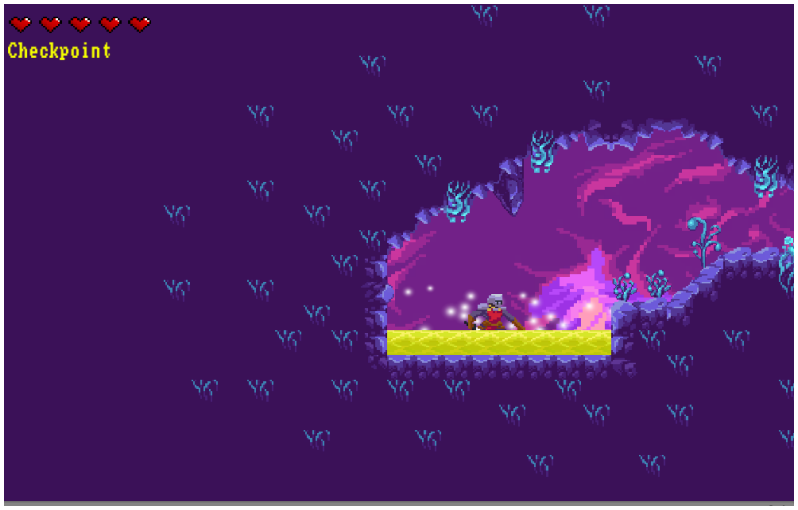
6.3.5. Smrtnost viteza i ostali mehanizmi

Zadnji važan mehanizam viteza jest njegova smrtnost. U nastavku je stavljen kôd za smanjivanje života viteza, njegovo umiranje ako mu je život nakon udarca jednak ili manji od nule te pokretanje efekta transparentnosti viteza dok je besmrtan.

```
public void smanjiZivotIgraca(int smanjenZivot)
{
if (besmrtan == false) {
      trenutniBrojZivota = trenutniBrojZivota - smanjenZivot;
      besmrtan = true;
}
if (trenutniBrojZivota <= 0)
        {
canvasIgra.SetActive(false);
canvasGameOver.SetActive(true);

Kamera.transform.gameObject.GetComponent<CinemachineBrain>().enabled = false;
Kamera.transform.position = new Vector3(-7.13f, 352.5f, -10f);
vitezUmroSprite.SetActive(true);
this.transform.position = new Vector2(zadnjiCheckpointX, zadnjiCheckpointY);
besmrtan = false;
this.transform.gameObject.SetActive(false);

}
            else
            {
                animacija.SetTrigger("ranjen");
                zvukovi.pokreniZvuk("zvukUdarenogViteza");
                StartCoroutine(transparentnost());
                Invoke("vratiSmrtnost", 1.5f);
            }
        }
}

private IEnumerator transparentnost()
{
    yield return new WaitForSeconds(0.4f);

    for (int i = 0; i < 20; i++)
    {
        if ((i % 2) == 0)
        {
            this.GetComponent<SpriteRenderer>().color = new Color(1f, 1f, 1f, 0.3f
);
        }
        else
        {
```

```
            this.GetComponent<SpriteRenderer>().color = new Color(1f, 1f, 1f, 1f);
        }
        yield return new WaitForSeconds(0.05f);
    }
}
```

Metoda „smanjiZivotIgraca()“ se poziva u drugim skriptama (npr. u skripti „neprijateljRakAI“) kada dođe do događaja koji šteti vitezu, pritom se navede koliko života se vitezu smanjuje. Ako vitez nije besmrtan, smanjuje mu se život te postaje besmrtan za jedno određeno vrijeme kako bi se igrač u to kratko vrijeme udaljio od izvora opasnosti. Ako nakon udarca ima manje od jednoga života, tada se grafičko sučelje, odnosno „canvas“ u kojem igrač može vidjeti koliko ima srca, sa SetActive(false) isključi te se aktivira sučelje za „GAME OVER“. S obzirom da se ovdje u igri koristi virtualna kamera iz „Cinemachine“ dodatka za Unity te se doda glavnoj kameri igre kako bi slijedila igrača, potrebno je isključiti tu virtualnu kameru igre te glavnu kameru s novim vektorom3 smjestiti na onu poziciju gdje se nalazi scena za „GAME OVER“. Nakon toga se aktivira posebna animacija koja prikazuje kako vitez umire. Ono što je vrlo bitno jest da se trenutna pozicija viteza smjesti na onu poziciju gdje je ušao u zlatnu vodu koja mu puni život i služi kao „checkpoint“ (Slika 6.), odnosno ako vitez negdje pogine, tada se vrati na to mjesto zlatne vode s punim životom. Svaki put kada vitez uđe u tu vodu, varijable zadnjiCheckpointX poprima vrijednost X trenutne pozicije viteza, dok zadnjiCheckpointY poprima vrijednost Y. Nakon toga se isključi gameobject, što znači da nije aktivan unutar igre te ga se ne može vidjeti. Aktivan postane samo ako igrač unutar „GAME OVER“ pozadine pritisne tipku „CONTINUE“ kako bi nastavio igrati.



Slika 6. Checkpoint igre

Ukoliko vitez nakon udarca ima i dalje više od jednog života, tada se uz zvuk i animaciju pokrene metoda preko „StartCoroutine-a“ „IEnumerator transparentnost“ [30]. Ta metoda je vrlo korisna zato što se unutar nje može koristiti naredba „yield return new WaitForSeconds()“ koja omogućava da se prvo pričeka nekoliko sekundi prije nego li se dalje krenu izvršavati

naredbe unutar te metode. Efekt transparentnosti se postiže tako da se u „for“ petlji na temelju parnosti broja varijable „i“ mijenja komponenta SpriteRenderer na način da mu se zadaje nova boja. Unutar zagrade new Color() se prva tri broja odnose na RGB boje, odnosno prvi broj se odnosi na crvenu, drugi na zelenu, dok se treći broj odnosi na plavu boju, dok se zadnji broj tipa „float“ odnosi na transparentnost i samo se nju mijenja.

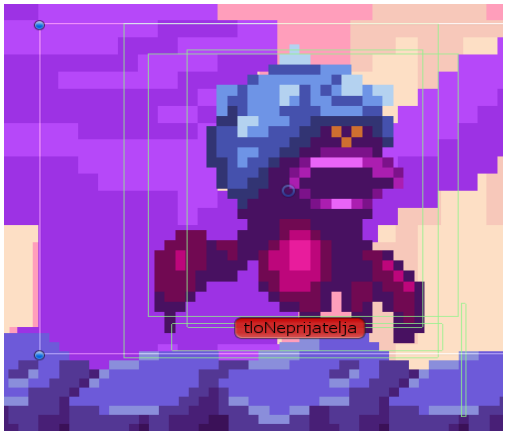
Ostali mehanizmi se odnose na „OnTrigger2D“ metode gdje se, ako se vitez nalazi u vodi, brzina rigidbody-a viteza smanjuje. Drugi primjer je ako dotakne neki predmet i time dobiva moć, tada se taj predmet obriše. Isto tako, ako vitez dotakne određene collidere koju ima svaka soba ove igre, tada se na njegovoj karti pojavi ta soba zajedno s ikonom viteza, tj. umanjenom grafičkom prezentacijom lika koji se trenutno nalazi u njoj.

6.4. Programski kôdovi neprijatelja

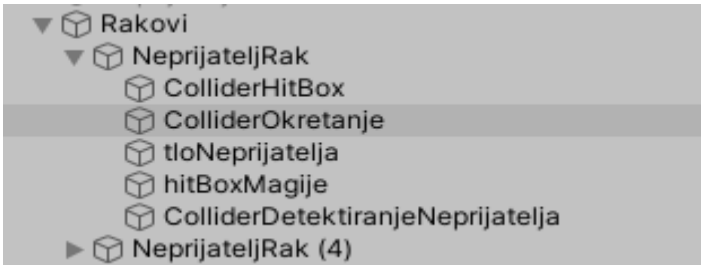
Isprogramiralo se u ovoj igri tri vrste „običnih“, slabijih neprijatelja, jedna vrsta mini glavnog neprijatelja i samo jedan glavni neprijatelj. Svaki neprijatelj ima određeni broj života, jačinu svojih napada, broj EXP bodova koje igrač dobiva nakon što ih uništi, svoju animaciju i svoje mehanike za napad viteza. Prvo će se opisati slabe neprijatelje.

6.4.1. Programski kôd neprijatelja „raka“

„Rak“ je „najobičniji“ neprijatelj koji se kreće lijevo ili desno te kada dođe do neke provalije ispred sebe, okrene se i počinje hodati prema suprotnoj strani. Inspiriran je crvenom kornjačom iz *Super Mario* serijala. Na sljedećim slikama može se vidjeti njegov izgled te koje sve collidere i djecu taj gameobject sadrži.



Slika 7. Neprijatelj „rak“



Slika 8. Gameobject NeprijateljRak sa svojom djecom

„Rak“ ima jedan collider koji služi da igrač gubi jedan život ako ga dotakne, drugi collider koji je smješten ispred raka služi za detekciju collidera sloja „TloZaHodati“ i ako ga više ne detektira, tada se gameobject rak okrene na suprotnu stranu sa svom svojom djecom. Na istom principu radi i neprijatelj „hobotnica“. Collider objekta „tloNeprijatelja“ služi za kretanje raka po collideru namijenjen za hodanje svih likova igre, „hitBoxMagije“ kojega svi slabiji neprijatelji i mini glavni neprijatelji sadrže te služi za detekciju magije viteza. Zadnje njegovo dijete sadrži collider za detektiranje drugog neprijatelja, bio to drugi „rak“ ili „hobotnica“ koja također sadrži taj collider, tako da se oba neprijatelja okrenu na suprotnu stranu kada su međusobno blizu. Važniji dijelovi kôda za „raka“ su sljedeći:

```
private void Update()
   {
        if (IsFacingRight() && zivot > 0)
        {
            rigidBodyRaka.velocity = new Vector2(moveSpeed, 0f);
        }
        else if (!IsFacingRight() && zivot > 0)
        {
            rigidBodyRaka.velocity = new Vector2(-moveSpeed, 0f);
        }
        else
        {
            rigidBodyRaka.velocity = new Vector2(0f, 0f);
        }

        if (zivot <= 0)
        {
            animacija.SetTrigger("zivotNaNuli");
            Invoke("deaktivirajNeprijatelja", 0.28f);
        }

        if (udaren == true)
        {
            udaren = false;
            Invoke("vratiBoju", (float)0.2);
        }

        foreach (Transform eachChild in transform)
        {
            if(eachChild.name == "ColliderHitBox")
            {
```

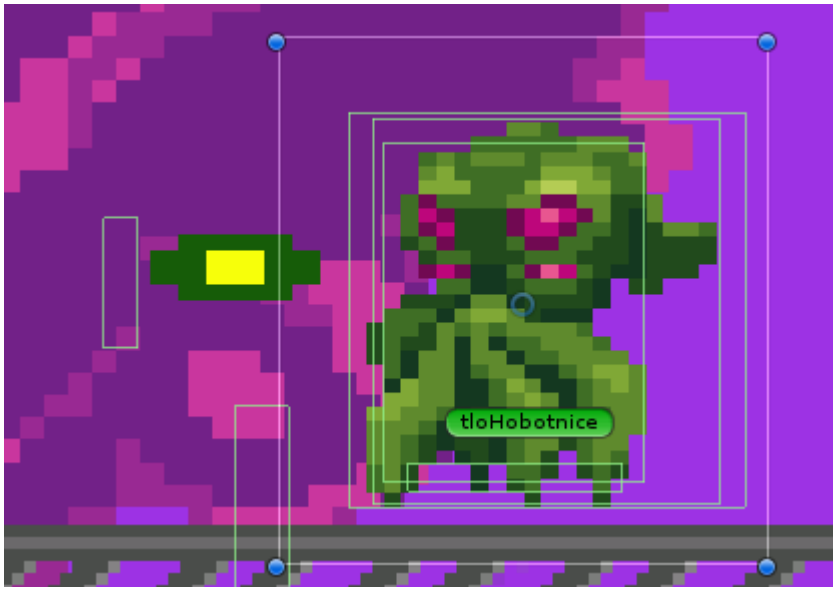
```
                if (eachChild.gameObject.GetComponent<BoxCollider2D>().IsTouching(v
itez.gameObject.GetComponent<BoxCollider2D>()))
                {
               vitez.gameObject.GetComponent<skriptaViteza>().smanjiZivotIgraca(1);
                }
            }
        }

    }

    private void vratiBoju()
    {
        this.GetComponent<SpriteRenderer>().color = originalnaBoja;
    }


    public bool IsFacingRight()
    {
        if (idiDesno == true)
        {
            return transform.localScale.x > Mathf.Epsilon;
        }
        else
        {
            return transform.localScale.x < Mathf.Epsilon;
        }
    }
```

Unutar Update() metode se kontinuirano poziva funkcija „IsFacingRight()“ kojom se ispituje rotacija raka [31]. Ako je rotacija po X-osi veća od vrlo male vrijednosti „Mathf.Epsilon“, tada se vraća „true“ te „rak“ se kreće prema desnoj strani ako mu je život veći od nule. Ako „idiDesno“ je jednak „false“, odnosno „rak“ je postavljen tako da prilikom pokretanja igre se kreće prema lijevoj strani igre, tada se vraća „true“ ako je rotacija manja od vrlo male vrijednosti, odnosno „false“ ako je veća.

Isto tako, stalno se ispituje broj života raka. Ukoliko je život raka jednak ili manji od nule, tada se pokreće animacija i ono što je važno, deaktivira se objekt. Razlog zašto se objekt samo deaktivira, a ne uništava pomoću naredbe Destroy() jest zato što kada vitez umre u igri ili dođe do checkpointa, taj rak se ponovno aktivira s punim životom na početnoj poziciji čije koordinate se spremaju u posebne varijable prilikom pokretanja igre.

Kada vitez dodirne „raka“, gubi jedan život, a s obzirom da rak ima više gameobjecta kao djecu, preko „foreach“ petlje traži se dijete koje nosi naziv „ColliderHitBox“. Kada ga nađe, ispituje se je li vitez dodirnuo raka preko naredbe „IsTouching()“ [32]. Ako dodiruje, u tom slučaju mu se smanji život.

6.4.2. Neprijatelj „hobotnica“ i leteći neprijatelj



Slika 9. Neprijatelj hobotnica

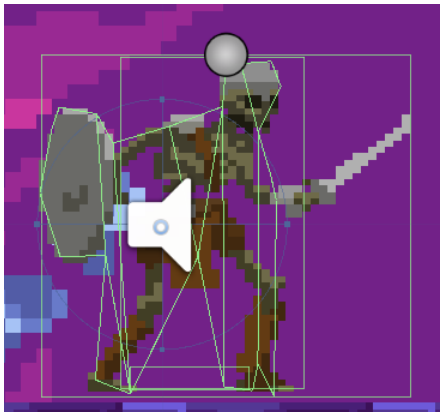
Na slici 9. se može vidjeti izgled „hobotnice“ i otrova kojeg ispaljuje. Kôd „hobotnice“ je vrlo sličan kôdu „raka“, samo što „hobotnica“ još sadrži jedan duži BoxCollider2D koji služi za registriranje igrača. Ako je vitez unutar tog dometa te ako se nalazi na istoj strani prema kojoj se „hobotnica“ kreće, tada se instancira otrov. Kôd te akcije koristi sve dosadašnje već spomenute metode, naredbe i funkcije stoga nije potrebno opisivati skriptu „neprijateljHobotnicaAI“.

Slika 10. Leteći neprijatelj

Slično kao i „hobotnica“, leteći neprijatelj (Slika 10.) također koristi BoxCollider2D za provjeravanje blizine viteza. Kada vitez dođe blizu, leteći neprijatelj počinje valovito lijevo desno padati prema dolje. Ni u ovom slučaju nije potrebno pokazati kôd za neprijatelja jer nema novih naredbi u njegovoj skripti koje bi bile važne za napomenuti.

6.4.3. Mini glavni neprijatelj kostur (engl. mini boss)

Kostur je izrađen da bude složeniji od spomenutih prethodnika iz razloga da kada mu se igrač približi, tada ga kostur počinje pratiti. Približi li se kostur igraču, počinje napadati sa svojim mačem. No ukoliko igrač zamahne svojim mačem prije njega, kostur se štiti te ga igrač nije ozlijedio. Isto tako ako igrač skoči iza kostura, kostur se i dalje štiti zbog toga što se njegov štit nalazi na suprotnoj strani od strane na kojoj drži mač (Slika 11.). Igrač može kosturu nauditi samo onda kada kostur napada i to samo na onoj strani prema kojoj kostur napada.



Slika 11. Mini glavni neprijatelj kostur

Postoji više načina kako isprogramirati da neprijatelj zna na kojoj se strani nalazi vitez. Prvi način koji se koristio za programiranje glavnog neprijatelja igre jest korištenje collidera, no alternativni i bolji način kojim se taj problem riješio nakon programiranja glavnog neprijatelja te implementirao u kôdu kostura jest računanje udaljenosti kostura od viteza naredbom „Vector2.Distance()“ u koju se stavljaju dva parametra, u ovom slučaju pozicija sivog kruga kostura koju se može vidjeti na slici 9. te pozicija viteza.

```
float udaljenostKosturaOdViteza = Vector2.Distance(transform.Find("tockaVektoraUda
ljenosti").transform.position, vitez.gameObject.transform.position);
```

Sada se pomoću „if-a“ može ispitati nalazi li se igrač blizu kostura, nije li kostur već u procesu napadanja te ako ga igrač ne napada, tada neka sam kostur napadne:

```
if (udaljenostKosturaOdViteza < 3.7 && !izvodiNapad)
     {
         rigidKostura.velocity = new Vector2(0f, 0f);
         izvodiNapad = true;
         StartCoroutine("napadni");
```

```
    }
```

Također se pomoću „if-ova“ ispituje prema kojoj strani kostur gleda. Ako gleda prema onoj strani u kojoj se nalazi igrač i ako ga igrač napada, kostur ne gubi život jer se štitom zaštiti (Slika 12.). Dio kôda u kojemu je to implementirano jest sljedeći:

```
else if (transform.position.x >= vitez.gameObject.transform.position.x && !jeliGle
daVecDesno && !izvodiNapad)
      {   animacija.SetBool("hodaj", false);
          animacija.SetTrigger("stitiSe");
          zvukovi.pokreniZvuk("zvukUdarenogStita");
          stitiSe = true;
      }
```

Slika 12. Mini glavni neprijatelj kostur se štiti

Za kostura će se prikazati i metoda koju ima svaki neprijatelj, a to je da se neprijatelji aktiviraju i vrate na svoja mjesta nakon što igrač dobije „GAME OVER“. Razlog zašto se dosad ta metoda nije spominjala sve do ovog dijela rada koji se bavi kosturom jest zato što mnoge varijable kostura moraju ponovo poprimiti početne vrijednosti koju poprime u Start() metodi:

```
public void vratiSeNaPocetnuPoziciju()
{
    unisten = false;
    zivot = puniZivot;
    foreach (Transform eachChild in transform)
    {
        if (eachChild.name == "hitBoxKostura")
        {
            eachChild.gameObject.GetComponent<PolygonCollider2D>().enabled = true;
        }
        if (eachChild.name == "macKostura")
        {
            eachChild.gameObject.GetComponent<CapsuleCollider2D>().enabled = false
;
        }
    }
    this.transform.position = new Vector2(pocetnaPozicijaX, pocetnaPozicijaY);
}
```

Prvo se stavi da kostur nije uništen te mu se život dokraja obnovi, postavi se da igrač može ponovo dobiti ozljedu ako dotakne kostura, deaktivira se mač kostura te se kostur vraća na početnu poziciju. „PolygonCollider2D“ se deaktivira odmah nakon što je život kostura manji ili jednak nuli zato što kostur više ne smije ozlijediti igrača nakon što se pokrene animacija kako kostur umire. Isto tako, ako igrač uništi kostura koji je u procesu napadanja, deaktivira se collider prilikom ponovnog stvaranja kostura, kako taj collider se aktivira samo kada je kostur u fazi napada i ima funkciju smanjiti živote igraču. U protivnom bi taj collider bio u početku uključen nakon što se kostur ponovno stvori te bi igrač dobio udarce od mača kostura iako kostur nije napao.

6.4.4. Glavni neprijatelj (engl. boss)

Glavni neprijatelj igre je čarobnjak koji koristi mač i čarolije kako bi napao viteza (Slika 13.). Sveukupno ima 4 vrste napada.

Slika 13. Glavni neprijatelj miruje

Prije nego li se krene s napadima ovog neprijatelja, važno je objasniti kako glavni neprijatelj zna gdje se nalazi igrač. Kako bi se to postiglo, koriste se dva velika BoxCollider2D te ukoliko igrač uđe u njihov prostor, prema toj strani počinje neprijatelj se kretati i napadati. Kôd kojeg lijevi i desni collideri imaju je sljedeći:

```
void Update()
{
    if (vitez.activeInHierarchy)
    {
        if (this.GetComponent<BoxCollider2D>().IsTouching(vitez.gameObject.GetComp
onent<BoxCollider2D>()))
        {
 if (vitez.gameObject.name == "Igrac" && this.gameObject.transform.parent != null)
            {
transform.parent.gameObject.GetComponent<glavniIgre>().pozicijaIgracaDesno = true;
transform.parent.gameObject.GetComponent<glavniIgre>().pozicijaIgracaDesnoSredina
= false;
```

```
            }
        }

        else if (this.gameObject.transform.parent != null)
        {
            transform.parent.gameObject.GetComponent<glavniIgre>().pozicijaIgracaD
esnoSredina = true;
        }

    }
```

Prvo se preispituje aktivnost gameobjecta „Igrac“ preko naredbe „activeInHierarchy“ [33], jer ako nije aktivan, to znači da je vitez poginuo i da se igraču prikazala „GAME OVER“ pozadina. Ako je aktivan, tada varijablu skripte svog roditelja „Glavni" postavlja da je jednak „true“ ukoliko se igrač nalazi na desnoj strani. Ukoliko roditelj tog gameobjecta koji sadrži taj collider nije aktivan, tada se neprijatelj ne kreće ako se igrač nalazi kod njega. Vrlo slično izgleda kôd i za suprotni collider i na temelju toga gdje se igrač nalazi, u skripti „glavnilgre“ se izvršava naredba za kretanje prema onoj strani gdje se nalazi igrač.

Sada slijede četiri vrste napada glavnog. Prvi napad je običnim mačem kojega glavni neprijatelj koristi dok ne koristi čarolije i samo u slučaju kada je igrač u njegovoj neposrednoj blizini. Kôd za to je vrlo jednostavan:

```
private IEnumerator napadMacem(int trenutniBroj)
   {
        yield return new WaitForSeconds(0.35f);

        this.transform.Find("macGlavnog").gameObject.GetComponent<BoxCollider2D>().
enabled = false;
        animacija.SetTrigger("stani");
        izvodiNapad = false;

        if (trenutniBroj == premaDesnojStraniGleda)
        {
            StartCoroutine("nasumicniNapadi", premaDesnojStraniGleda);
        }
   }
```

Ono što je vrlo važno ovdje za napomenuti jest varijabla trenutniBroj te zadnji „if“ koji se ispituje unutar ove metode. Metoda prima vrijednost 1 ako se igrač nalazi na desnoj strani, odnosno -1 ako se nalazi na lijevoj strani. Nakon što neprijatelj izvrši napad, ispituje se je li pozicija igrača i dalje na toj strani nakon napada tako da se usporedi taj trenutniBroj sa globalnom varijablom „premaDesnojStraniGleda“. Ako nisu jednaki, tada se ništa ne izvršava, no ukoliko jesu, odmah se poziva metoda „nasumicniNapadi“ jer se igrač nalazi na onoj strani prema kojoj je glavni neprijatelj okrenut. Kôd za spomenutu metodu koristi „UnityEngine.Random.Range(prviBroj, drugiBroj)“ [34] kako bi glavni nasumično odabrao napad ledom, vatrom ili grmljavinom, s time da čaroliju leda koristi u drugoj fazi borbe, odnosno fazi koja se aktivira nakon što mu igrač potroši pola života i tada glavni neprijatelj pocrveni te

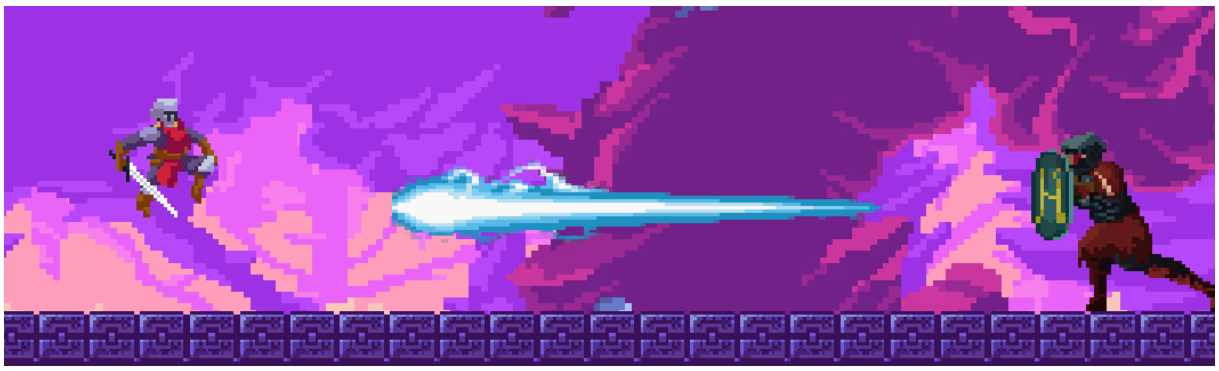
se brže kreće. Svaki napad koristi instanciranje novih gameobjecta koji imaju svoje collidere koji ozljeđuju viteza. Zanimljiv je napad ledom jer glavni neprijatelj poziva veći broj gameobjecta leda koji su poredani jedan do drugog, što se može vidjeti na sljedećem kôdu:

```
var nizLed = new GameObject[20];
for (int i = 0; i < 20; i++)
      {
       float nasumicnaVisinaLeda = UnityEngine.Random.Range(-3f, 3f);
       nizLed[i] = Instantiate(led, this.transform.position, Quaternion.identity);
       nizLed[i].transform.gameObject.transform.localPosition = new Vector2(this.t
ransform.localPosition.x - 50 + (float)5.5 * i, this.transform.localPosition.y+ 20
 + nasumicnaVisinaLeda);
          Destroy(nizLed[i], 3.5f);
      }
```

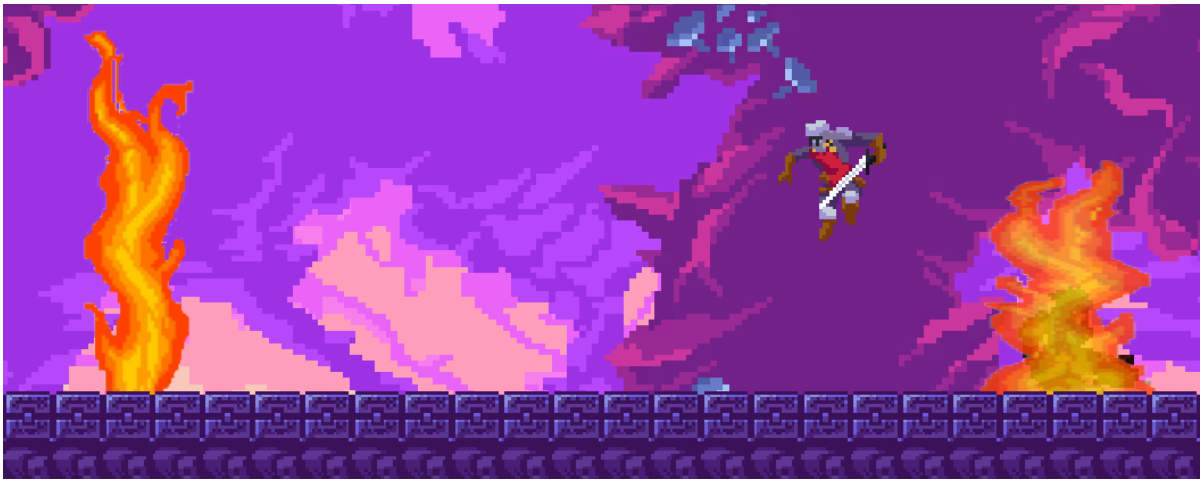
Dvadeset komada leda se preko „for“ petlje instancira, pomoću „Random.Range“ naredbe se odredi pozicija svakog leda na Y-osi dok su jedan od drugoga uvijek udaljeni za 50 po X-osi. Svaki led se sprema u polje te se za svakog odredi pozicija. Nakon nekoliko sekundi se svaki led uništi naredbom „Destroy()“.

Općenito glavni neprijatelj ima napad vatrom koja se instancira prilikom početka napada kada počinje trčati prema igraču, nakon toga se počinje kotrljati prilikom čega nastane manja vatra koja ga prati te igrač samo akcijom skakanja može izbjeći tu opasnost. To se u kôdu riješilo sa „if-ovima“ i instanciranjima. Drugi napad je grmljavina koju igrač mora preskočiti te samo u tom slučaju ne može napadati glavnog neprijatelja na onoj strani gdje drži štit. Kada glavni neprijatelj napada ledom, tada igrač mora stati između velikih padajućih komada leda. Napad mačem se događa kada collider koji se nalazi na dnu ispred glavnog neprijatelja detektira viteza, no uvjet je da tad glavni neprijatelj ne izvodi napad čarolijom. Kako izgledaju opisani napadi može se vidjeti na sljedećim slikama:

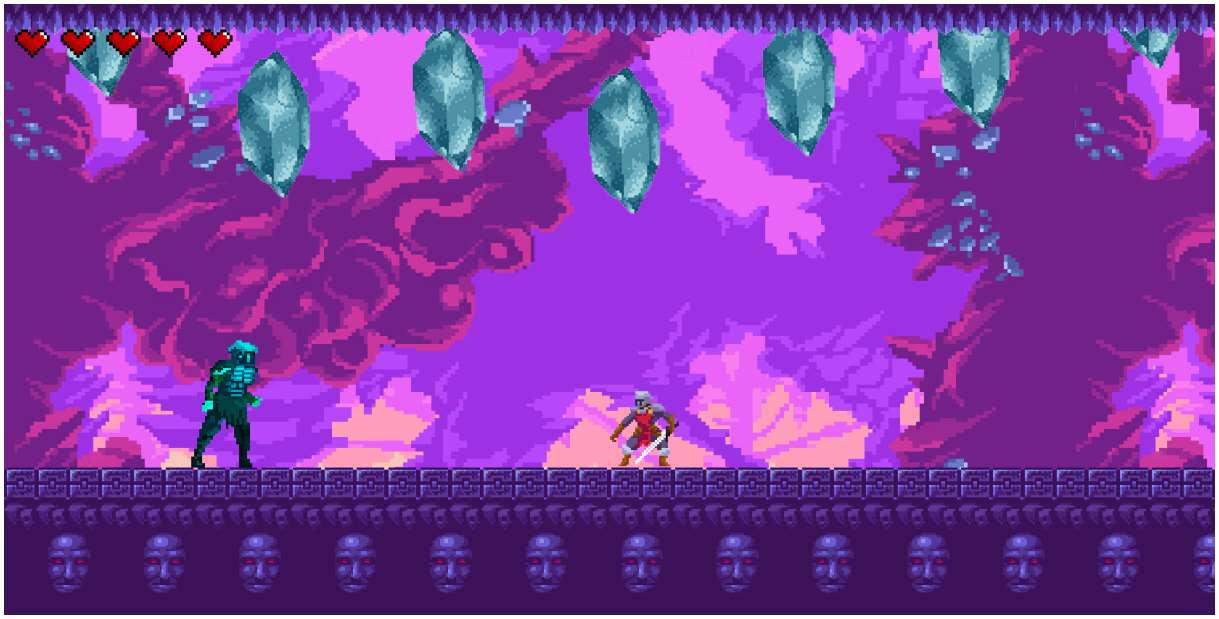
Slika 14. Glavni neprijatelj napada mačem



Slika 15. Glavni neprijatelj napada munjom



Slika 16. Glavni neprijatelj napada vatrom



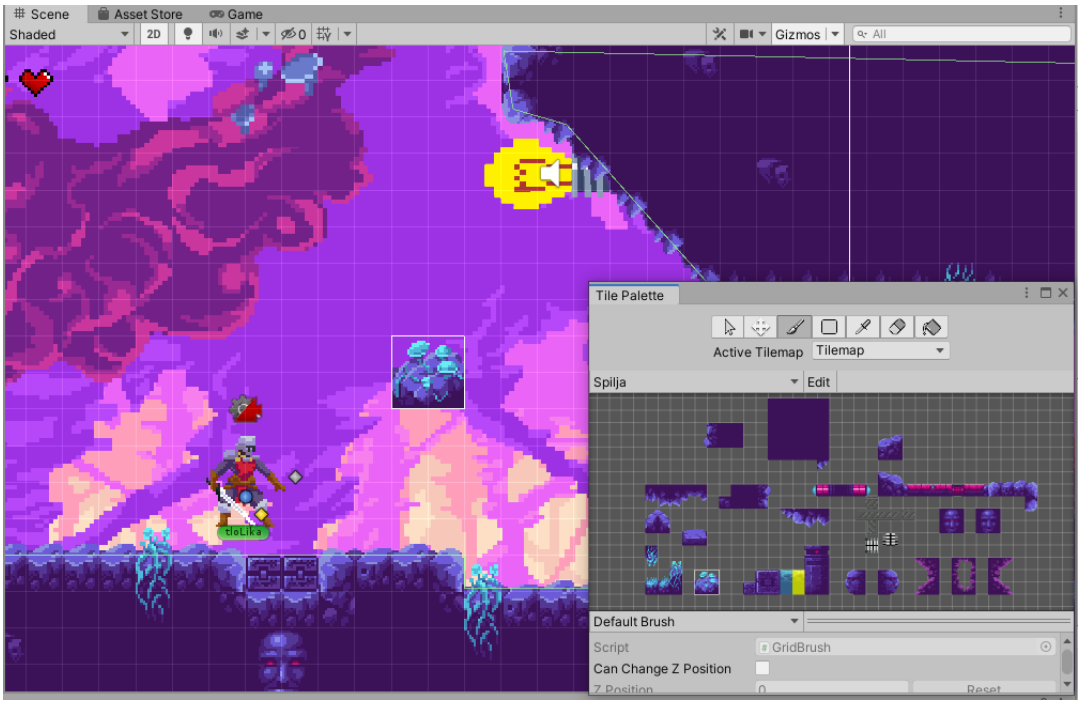
Slika 17. Glavni neprijatelj napada ledom

Nakon što igrač čarobnjaku smanji život na nulu, pokrene se animacija u kojoj glavni neprijatelj umire te nestane kamen tj. prepreka koja se stvorila prilikom ulaska igrača u sobu glavnog neprijatelja. Na taj način se onemogućilo igraču da izađe van iz sobe dok traje borba s glavnim neprijateljem. Nakon toga, igrač dobiva zadnji predmet koji mu omogućava korištenje magije.

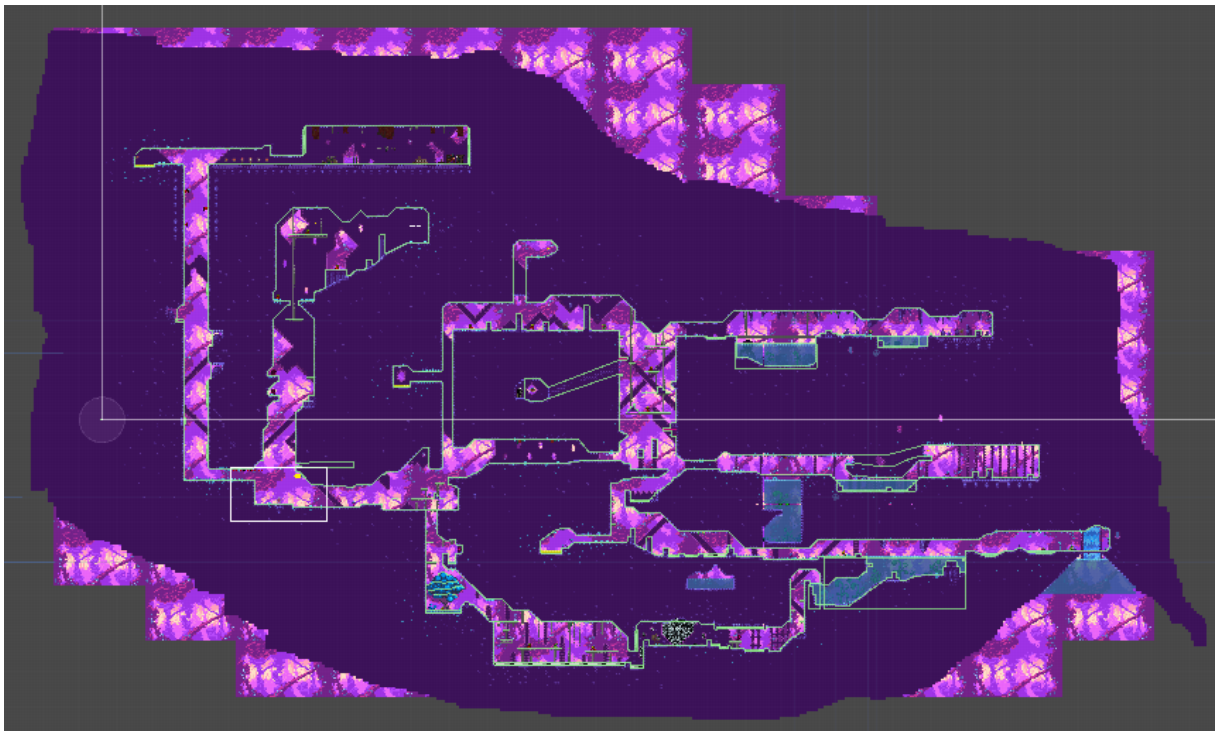
6.5. Dizajniranje svijeta

Sada kada su pojašnjeni svi likovi u igri, može se krenuti na dizajniranje svijeta. Kako bi se najlakše dizajniralo 2D igru, koristit će se „Tile Palette“, što je prozor u Unityju u kojemu se implementiraju slike, pa ih se onda koristi za „crtanje“ svijeta. U svrhu izrade igre korišten je „tileset“ špilje koji je preuzet iz Unity Asset Storea. Sam „tileset“ je zapravo skupina slika unutar jedne veće slike. Unity ima mogućnost prepoznavanja tih slika te ih može međusobno odvojiti.

Prvo se stvori novi gameobject s komponentom „Tilemap“. Nakon toga se može vidjeti mreža kockica u glavnom prozoru „Screen“. Na navedene kockice stavljaju se sličice tileseta. No prvo je potrebno odvojiti te slike te ih „drag and drop-om“ implementirati u Tile Palette [35]. Sada je sve spremno da se može po toj mreži kockica dizajnirati svijet (Slika 18.). Svijet nakon dizajna, postavljanja collidera za hodanje, postavljanje igrača i neprijatelja se može vidjeti na slici 19.



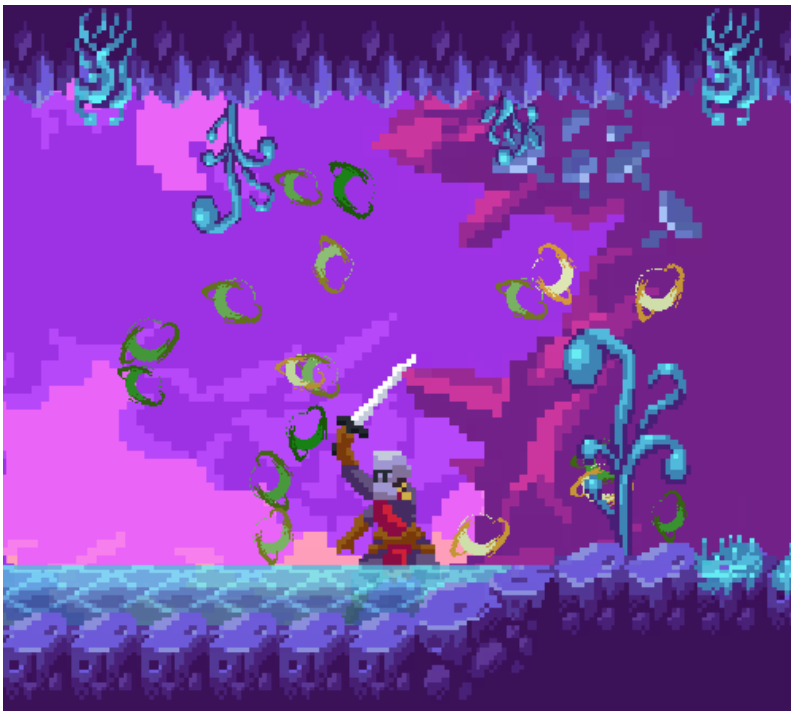
Slika 18. Glavni neprijatelj napada ledom



Slika 19. Vanzemaljski svijet igre Knivader

6.6. Izrada efekata čestica (engl. particle effect)

Efekt u Unityju se izradi tako da se na novom ili postojećem gameobjectu dodaje desnim klikom *Effect -> Particle System*. Tada se otvore mnoge mogućnosti za izradu efekata. Neki efekt može biti u 2D ili 3D-u, može se raditi o jednoj ili više slika koji se instanciraju u nekoj poziciji, također i odrediti koliko dugo se instanciraju te slike, trajanje života te slike prije nego li se uništi, utječe li gravitacija i određeni collideri na njih i tako dalje [36]. Može se zaključiti da Unity zaista nudi mnogo načina kako bi se neki efekt oblikovao i ponašao. Primjer jednog efekta se vidi na slici 20. gdje je igrač uništio hobotnicu, nakon čega se instancira veći broj malih zelenih slika koje posvuda padaju oko viteza.



Slika 20. Efekt čestica hobotnice

6.7. Glavni izbornik

Nakon što igra sadrži svijet i neprijatelje, slijedi pojašnjenje glavnog izbornika. U podpoglavlju „Tema i igrivost video igre – „Knivader“ bilo je objašnjeno što sve sadrži glavni izbornik. Kada se otvori izbornik, prvo što igrač vidi jest statistika viteza te koje sve predmete on posjeduje (Slika 21.).

Slika 21. Glavni izbornik igre

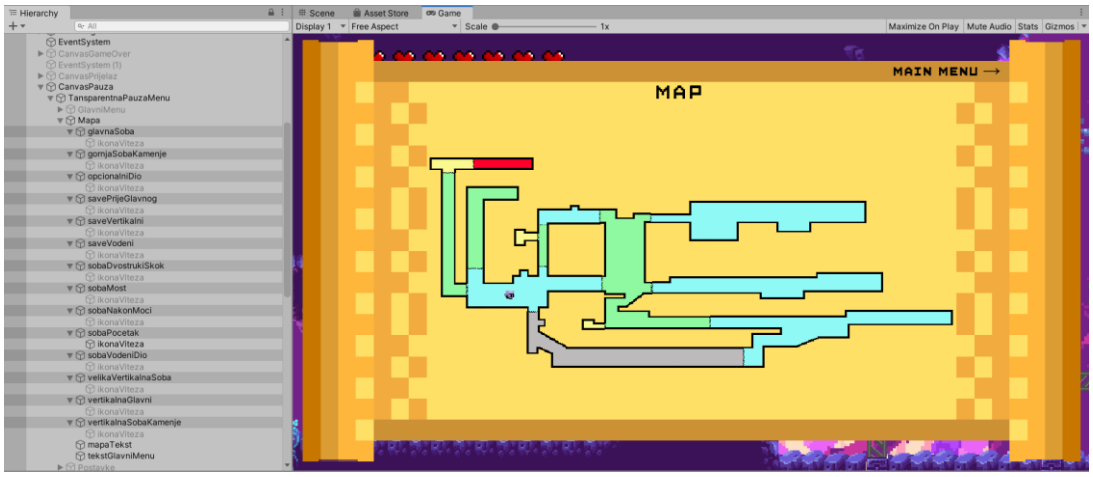
Od izuzetne je važnosti da se svi objekti u igri zaustave kada se pozove glavni izbornik igre. To se postiže tako da se stavi da je „Time.timeScale = 0f“. Kada je„timeScale“ jednak nuli, tada je igra zaustavljena, no kada je postavljena na jedan, tada vrijeme normalno teče tijekom nezaustavljene tj. pauzirane igre [37]. Ta naredba se nalazi unutar skripte „skriptaMenu“ koja je ovdje ispisana:

```
void Update()
  {
      if (!canvasGameOver.activeInHierarchy || izadiVan)
      {
          if (Input.GetKeyDown(KeyCode.Escape))
          {
              if (igraJePauzirana)
              {
                  int i = 0;
                  foreach (AudioSource aud in sviAudio)
                  {
                      if (aud != null)
                      {
                          aud.volume = sviAudioVolume[i];
                      }
                      i++;

                  }

                  zvukovi.pokreniZvuk("zvukPapira");
                  nastaviSIgrom();
else
               {
   AudioSource[] audios = FindObjectsOfType(typeof(AudioSource)) as AudioSource[];
                    sviAudio = audios;
                    int broj = int.Parse(audios.Length.ToString());
                    float[] audioVolume = new float[broj];
                    int i = 0;
                    foreach (AudioSource aud in audios)
                    {
                        audioVolume[i] = aud.volume;
                        aud.volume = 0.1f;
                        i++;

                    }
                    sviAudioVolume = audioVolume;

                    zvukovi.pokreniZvuk("zvukPapira");
                    Pauziraj();
                }

              }
```

Prvo se provjeri aktivnost pozadine „GAME OVER“ u igri, jer ako je aktivan, tada igrač ne smije pristupiti glavnom izborniku. Ako nije, tada se pritiskom na tipku „Escape“ provjerava istinitost tvrdnje da je igra pauzirana. Ako jest, tada se vrati glasnoća tog zvučnog efekta koji je bio spremljen u polju „sviAudioVolume[]“ prije nego se sasvim stišao taj zvuk tijekom zaustavljanja, odnosno pauziranja igre. Na kraju se pozove funkcija „nastaviSIgrom()“

koja postavlja timescale na jedan. U slučaju da igra nije pauzirana nakon što se pritisne „Escape“, tada se u polje spremaju svi aktivni zvučni efekti i glazbe preko naredbe „FindObjectsOfType(typeof())“ [38]. Ta naredba pretražuje sve objekte tipa „AudioSource“ te ih sprema u polje audios. Nakon toga se pomoću „foreach“ petlje sprema u posebno polje „audioVolume“ tipa „float“ glasnoća svakog zvuka. Na kraju se pozove metoda „Pauziraj()“ koja stavlja timescale na nulu.

Želi li igrač vidjeti kartu igre, pritisne lijevu strjelicu na tipkovnici dok se nalazi na glavnom izborniku nakon čega se prikaže karta cijeloga svijeta. U toj karti aktivne se slike onih soba koje je igrač posjetio (Slika 22.).



Slika 22. Karta svijeta

Karta se nacrtala koristeći program Adobe Photoshop, te se izrezala na manje dijelove koji se nazivaju u ovom žanru „sobe“. Plavo obojane sobe označavaju horizontalne sobe, zelene sobe vertikalne, žute sobe checkpoint-ove, u crvenoj sobi se nalazi glavni neprijatelj igre dok je siva soba posve opcionalna te nudi ponešto veće izazove za igrača. Soba se na karti prikaže samo onda kada igrač uđe u tu sobu koja sadrži collider kao okidač. U skripti „skriptaViteza“ se nalazi sljedeći kod:

```
if (collision.transform.gameObject.name == "glavnaSobaC")
{
    glavnaSoba.GetComponent<Image>().enabled = true;
    smjestiIkonu(glavnaSoba.transform.GetChild(0).gameObject);
}
```

Dohvaća se komponenta Image objekta s nazivom „glavnaSoba“ te se postavi da je slika omogućena, odnosno da ju se može vidjeti tijekom igre. Isto tako se dohvaća preko

naredbe „GetChild()“ prvo dijete tog objekta, koji je u ovom slučaju ikona viteza te se tako na karti može vidjeti ikona viteza u onoj sobi u kojoj se igrač trenutno nalazi.

Zadnji dio izbornika jesu „Postavke“ u kojima postoje dva odabira. Odabirom prve opcije se nastavlja s igrom, dok se drugom izađe na „title screen“ igre. Ukoliko se odabere da se izađe van iz igre, tada se poziva iz skripte „skriptaZaPrijelaz“ metoda „moraIzaciIzIgre()“ koja sadrži u sebi implementiran „UnityEngine.SceneManagement“ za korištenje naredbe koja učita određenu scenu [39], u ovom slučaju :

```
SceneManager.LoadScene("KnivaderTitleScreen“);
```

6.8. Efekt paralakse (engl. parallax efekt)

Efekt paralakse je efekt koji se koristi kako bi se stekao dojam dubine na način da kada se igrač kreće, tada se pozadina također kreće. Ona pozadina koja sadrži u sebi sliku koja prikazuje udaljenije dijelove svijeta se sporije kreće od one koja prikazuje prirodu koja je igraču vizualno bliže. Kôd za to je izrađen uglavnom koristeći YouTube video zapis [40]:

```
private float duljina;
private float startnaPozicija;

public GameObject kamera;

public float parallaxEffect;

void Start()
{
    startnaPozicija = transform.position.x;
    duljina = GetComponent<SpriteRenderer>().size.x;
}

void Update()
{
    float udaljenost = kamera.transform.position.x * parallaxEffect;
    transform.position = new UnityEngine.Vector3(startnaPozicija + udaljenost, tra
nsform.position.y, -20);
}
```

U Start() metodi se prvo odredi X pozicija pozadine te duljina slike. Nakon toga se unutar Update() metode svakoga puta instancira i deklarira varijabla udaljenost koja je tipa „float“. Toj varijabli se dodjeljuje vrijednost trenutne pozicije X glavne kamere koja je pomnožena s brojem koje se definira unutar Unitya s obzirom da je „public“. Nakon toga svaki put kada se igrač kreće, a istovremeno i kamera koja ga prati, promijeni se pozicija te slike preko novoga vektora Vector3 koja kao X vrijednost sadrži novu vrijednost X pozicije. Na taj način se stvara iluzija dubine okoline.

6.9. Zvučni efekti

Zvučni efekti se koriste za razne situacije. Kako bi se olakšalo njihovo korištenje, izrađena je posebna skripta „zvukovi“ u kojoj se u statičnu varijablu tipa „AudioClip“ smještaju datoteke audio formata (MP3, OGG, WAV). Sam kôd je izrađen na temelju YouTube video zapisa kojega je izradio Alexander Zotov [41].

```
public static AudioClip zvukSkakanja;
static AudioSource izvorSoundEffecta;

void Start()
   {
        zvukSkakanja = Resources.Load<AudioClip>("docekatiSeNaTlo");
static AudioSource izvorSoundEffecta;
izvorSoundEffecta = GetComponent<AudioSource>();
    izvorSoundEffecta.loop = false;
}


public static void pokreniZvuk(string naziv)
   {
switch (naziv)
      {

case "skakanje":
               izvorSoundEffecta.PlayOneShot(zvukSkakanja);
               break;
        }
}
```

U ovom kôdu, deklarira se prvo statična varijabla zvuka skakanja u koju se u Start() metodi pohrani određeni zvučni efekt koristeći klasu „Resources“ koja pronađe i učita tu audio datoteku koja se mora nalaziti unutar mape „Resources“. izvorSoundEffecta se koristi kako bi se dohvatila komponenta AudioSource i pokrenuo određeni zvučni efekt naredbom „PlayOneShot“. „Switch...case“ se koristi tako da bi se moglo jednom linijom kôda kod drugih skripata pokrenuti određeni zvučni efekt.

6.10. Početna scena i GAME OVER pozadina

Zadnje što je ostalo za reći su početna scena igre, odnosno „title screen“ i pozadina unutar igre kada je igra završena, odnosno „GAME OVER screen“. Igrač kada pokrene igru pred sobom vidi sljedeću sliku (Slika 23.):

Slika 23. Početna scena igrice Knivader

Početna scena koristi istu skriptu koja se koristi unutar glavnog izbornika kada se igrač želi vratiti na početnu scenu i koja se koristi prilikom odustajanja kod „GAME OVER“ pozadine (Slika 24.) Dok se početna scena igre nalazi u zasebnom „sceneu“ Unitya, „GAME OVER“ se nalazi unutar onog „scenea“ u kojoj se nalazi svijet zajedno s vitezom i neprijateljima.

Slika 24. GAME OVER pozadina

7. Zaključak

Knivader je prva cjelokupna video igra koju sam uspio od početka sam isprogramirati, a ne samo dizajnirati svijet s gotovim objektima u službenim i neslužbenim level editorima. Kroz cijeli proces izrade, susreo sam se s mnogim nepoznatim pojmovima, novim metodama i funkcijama unutar Visual Studija. Nisam znao koje sve funkcionalnosti Unity nudi, javljale su mi se mnoge pogreške u kôdu te često igra nije radila onako kako sam želio. Glavni problemi bili su propadanje neprijatelja kroz tlo te njihova detekcija tla i drugih neprijatelja, javljanje pogrešaka nakon brisanja objekata, aktiviranje pogrešne animacije protagonista u određenoj situaciji, pozicioniranje objekata tako da su vidljivi unutar kamere video igre te izrada glatkih i kontrole s pravilnim odazivom za glavnog protagonista. Što se tiče tehničkih performansi, nisam imao problema. Inspiraciju za rješenja sam ponekad tražio u Unity dokumentima, raznim forumima i YouTube video zapisima te nalazio slične probleme s kojima sam se susretao. Ono što bih još želio poboljšati u igri jesu animacije viteza kako se određene animacije pokreću u neželjenim okolnostima, precizirati collidere neprijatelja i napadanje viteza te ukupno vrijeme za prolazak igre produljiti s jednog sata na minimalno šest sati povećanjem broja soba, vrsta neprijatelja, predmeta i moći te općenito sadržaja video igre. Uzimam kao mogućnost da ću video igru izdati kao komercijalni proizvod, za što je potrebno određene besplatne/tuđe resurse koje sam koristio zamijeniti s onima vlastite izrade ili svojeg potencijalnog razvojnog tima.

Upravo rješavanjem raznih problema sam uspio produbiti svoje znanje C# programskog jezika te znam kako se može isprogramirati jednostavna umjetna inteligencija neprijatelja, razumijem što kao dizajner video igara moram u početku izrade imati na umu, koliko mi je otprilike potrebno vremena za izradu određenog mehanizma u igri te sam općenito naučio koristiti Unity, profesionalni programski alat za izradu video igara.

Metroidvania kao poseban žanr objedinjuje mnoge elemente drugih žanrova i upravo zbog toga je izabrana kao glavna tema ovoga rada. Uvjerio sam se da izrada takve vrste igre može vremenski dugo potrajati kako sadrži različite mehanizme koje je potrebno izraditi.

Nađem li se u poziciji da se profesionalno bavim razvojem video igara, svakako bih želio biti u timu koji izrađuje igru ovakvoga tipa. Žanr sadrži raznolike mehanizme čime se brojčano gledajući ne može pohvaliti svaki žanr. Sam proces razvoja takve igre je zasigurno vrlo interesantan, kao što je bila izrada Knivadera.

8. Popis slika

9. Popis korištenih resursa

9.1. Korišteni programski alati

1. Programski alat/motor: Unity,
   https://unity3d.com/get-unity/download
   Dostupno 27.7.2020.
2. Razvojno okruženje: Microsoft Visual Studio Professional 2019 (s dodatkom podrške za rad s Unity-em),
   https://visualstudio.microsoft.com/downloads/
   Dostupno 27.7.2020.
3. Grafički računalni program: Adobe Photoshop,
   https://www.adobe.com/products/photoshop/free-trial-download.html
   Dostupno 27.7.2020.
4. Program za vađenje zvuka/pretvorbu video zapisa s video stream web stranice u audio: 4K Video Downloader,
   https://www.4kdownload.com/products/product-videodownloader
   Dostupno 27.7.2020.
5. Web stranica za skraćivanje audio datoteka - audiotrimmer.org,
   https://audiotrimmer.com/
   Dostupno 27.7.2020.
6. Web stranica za dobivanje png slika od jedne slike formata gif -ezgif.org,
   https://ezgif.com/split
   Dostupno 14.8.2020.
7. Ugradnja Cinemachine u Unity-u: Unity Package Manager,
   Dostupno 27.7.2020.

9.2. Glazba i zvučni efekti

1. Glazba title screena: Metroid: Zero Mission Title screen Remix – Kingdom Hearts Insider,
   https://www.khinsider.com/midi/gba/metroid-zero-mission
   Dostupno 27.7.2020.
2. Glavna glazba unutar igre: Metroid Samus Returns Official Soundtrack - The Surface of SR388 – YouTube,
   https://www.youtube.com/watch?v=yhQWJG-_Oco
   Dostupno 27.7.2020.
3. Glavna glazba glavnog neprijatelja igre: Metroid Prime 2: Echoes - Quadraxis Battle – YouTube,
   https://www.youtube.com/watch?v=kpQ_OQAtgDQ
   Dostupno 27.7.2020.
4. Zvučni efekt: vitez dobiva udarac – freesound.org,
   https://freesound.org/people/elynch0901/sounds/464487/
   Dostupno 27.7.2020.
5. Zvučni efekt: vitez skače – freesound.org,
   https://freesound.org/people/Jummit/sounds/512677/
   Dostupno 27.7.2020.
6. Zvučni efekt: vitez hoda – freesound.org,
   https://freesound.org/people/qubodup/sounds/59992/
   Dostupno 27.7.2020.
7. Zvučni efekt: napad mačem – freesound.org,

https://freesound.org/people/bevangoldswain/sounds/54779/
Dostupno 27.7.2020.
8. Zvučni efekt: leteći neprijatelj pada– freesound.org, https://freesound.org/people/MATRIXXX_/sounds/453060/ Dostupno 27.7.2020.
9. Zvučni efekt: novo srce – freesound.org, https://freesound.org/people/pumodi/sounds/150219/ Dostupno 27.7.2020.
10. Zvučni efekt: level up – freesound.org: https://freesound.org/people/MATRIXXX_/sounds/523753/ Dostupno 27.7.2020.
11. Zvučni efekt: nova moć – freesound.org, https://freesound.org/people/MATRIXXX_/sounds/523754/ Dostupno 27.7.2020.
12. Zvučni efekt: štit– freesound.org, https://freesound.org/people/CTCollab/sounds/223629/ Dostupno 27.7.2020.
13. Zvučni efekt: kostur je udaren – freesound.org, https://freesound.org/people/cliftonmcarlson/sounds/392883/ Dostupno 27.7.2020.
14. Zvučni efekt: kostur je uništen – freesound.org, https://freesound.org/people/personwhois/sounds/458974/ Dostupno 27.7.2020.
15. Zvučni efekti: magija je spremna za korištenje, korištenje magije – YouTube, https://www.youtube.com/watch?v=nII-9lV73e4 Dostupno 27.7.2020.
16. Zvučni efekt: dobivanje nove moći za mač - freesound.org, https://freesound.org/people/Mrthenoronha/sounds/509479/ Dostupno 27.7.2020.
17. Zvučni efekt: velika voda – freesound.org, https://freesound.org/people/roboroo/sounds/436792/ Dostupno 27.7.2020.
18. Zvučni efekt: mala voda – freesound.org, https://freesound.org/people/nilbul/sounds/404829/ Dostupno 27.7.2020.
19. Zvučni efekt: laser – freesound.org, https://freesound.org/people/Julien%20Matthey/sounds/346918/ Dostupno 27.7.2020.
20. Zvučni efekt: punjenje života viteza kod checkpointa – Legend of Zelda sound effects, http://noproblo.dayjo.org/ZeldaSounds/ Dostupno 27.7.2020.
21. Zvučni efekt: pomicanje mača tijekom Game Over-a – freesound.org, https://freesound.org/people/lostchocolatelab/sounds/1460/ Dostupno 27.7.2020.
22. Zvučni efekt: listanje starog papira – freesound.org, https://freesound.org/people/gynation/sounds/82378/ Dostupno 27.7.2020.
23. Zvučni efekt: govor Drakule –YouTube, https://www.youtube.com/watch?v=OMTizJemHO8&feature=emb_title Dostupno 27.7.2020.
24. Zvučni efekt: struja – freesound.org, https://freesound.org/people/sharesynth/sounds/341666/ Dostupno 15.8.2020.

25. Zvučni efekt: lišće – freesound.org,
https://freesound.org/people/nextmaking/sounds/86015/
Dostupno 15.8.2020.
26. Zvučni efekt: vodopad – freesound.org,
https://freesound.org/people/deleted_user_7146007/sounds/383732/
Dostupno 15.8.2020.

9.3. Font, slike neprijatelja, glavnog lika, okoline:

1. Retro font: MZM Medium – Google Drive,
https://drive.google.com/file/d/1LeKSt6uQhag41uoosTPkG7kI6CjUsDpi/view
Dostupno 27.7.2020.
2. Slike neprijatelja vanzemaljaca, tileset špilje – Unity Asset Store,
https://assetstore.unity.com/packages/2d/characters/warped-caves-103250,
Dostupno 27.7.2020.
3. Slike dekoracija iz tamnice, tileset tamnice – Unity Asset Store,
https://assetstore.unity.com/packages/2d/environments/platformer-set-150023
Dostupno 14.8.2020.
4. Slike glavnog lika – Unity Asset Store,
https://assetstore.unity.com/packages/2d/characters/hero-knight-2-168019
Dostupno 27.7.2020.
5. Slike glavnog neprijatelja iz fantazije – Unity Asset Store,
https://assetstore.unity.com/packages/2d/characters/hero-nad-opponents-animation-140776
Dostupno 27.7.2020.
6. Slike 2D stvari – Unity Asset Store,
https://assetstore.unity.com/packages/2d/gui/icons/2d-pixel-item-asset-pack-99645
Dostupno 27.7.2020.
7. Slike vatre kojega glavni neprijatelj koristi – PNGKIT,
https://www.pngkit.com/view/u2q8y3r5e6u2e6a9_fireball-sprite-png-fireball-sprite-sheet/
Dostupno 27.7.2020.
8. Slike male vatre – pixelartmaker.com,
http://pixelartmaker.com/art/9302e491c4282b7
Dostupno 27.7.2020.
9. Slika leda kojega glavni neprijatelj koristi – redbubble.com,
https://www.redbubble.com/i/poster/Aquamarine-Crystal-Pixel-art-birthstone-Pixel-art-by-PixlPrints/30656476.LVTDI
Dostupno 27.7.2020.
10. Slika pahulje – pixelartmaker.com,
http://pixelartmaker.com/art/761fdd56822c92d
Dostupno 27.7.2020.
11. Slike grmljavine koju glavni koristi – Unity Asset Store,
https://www.pinterest.com/pin/411023903468719983/
Dostupno 27.7.2020.
12. Slika magije mača – NICEPNG,
https://www.nicepng.com/ourpic/u2q8u2i1w7o0q8u2_slash-super-smash-bros-logo-pixel-art/
Dostupno 27.7.2020.
13. Slika predmeta za skakanje po zidu – pixilart.com,
https://www.pixilart.com/art/shroomite-digging-claw-terraria-95a1ab0c350b53c
Dostupno 27.7.2020.

14. Slika predmeta za dvostruki skok – pixelartmaker.com,
http://pixelartmaker.com/art/292b5873a0eb6b7
Dostupno 27.7.2020.
15. Slika title screena – imgur.com,
https://imgur.com/9ZKJmUm?r
Dostupno 27.7.2020.
16. Slike Alucarda – The VG Resource Wiki,
https://www.spriters-resource.com/fullview/3228/
Dostupno 27.7.2020.
17. Slike – The VG Resource Wiki,
https://www.spriters-resource.com/snes/smetroid/sheet/1725/
Dostupno 27.7.2020.
18. Slika starog papira za glavni meni – pixelartmaker.com,
http://pixelartmaker.com/art/e5fd04dbf0c146b
Dostupno 27.7.2020.
19. Slika žarulje: deviantart.com,
https://www.deviantart.com/galacticabys/art/lamp-pixel-art-719212075
Dostupno 27.7.2020.
20. Slike vodopada: uanmuvorda.blogspot.com,
https://uanmuvorda.blogspot.com/2019/01/waterfall-animation.html
Dostupno 15.8.2020.
21. Slika drva: reddit.com,
https://www.reddit.com/r/PixelArt/comments/i5mp0q/alien_swamp_tree/
Dostupno 15.8.2020.
22. Slika lubanja: pixeljoint.com,
http://pixeljoint.com/pixelart/85504.htm
Dostupno 15.8.2020.

23. Slike metala za hodanje – vlastita izrada u programu Adobe Photoshop
24. Slika mosta za hodanje – vlastita izrada u programu Adobe Photoshop
25. Slika „THE END“ – vlastita izrada u programu Adobe Photoshop
26. Slike opasnih igli – vlastita izrada u programu Adobe Photoshop
27. Slika kapljice vode – vlastita izrada u programu Adobe Photoshop
28. Slika otrova hobotnice – vlastita izrada u programu Adobe Photoshop
29. Slika mača za Game Over – vlastita izrada u programu Adobe Photoshop
30. Slika lasera – vlastita izrada u programu Adobe Photoshop
31. Slika karte Knivadera – vlastita izrada u programu Adobe Photoshop

10. Literatura

[1] „Super Mario Bros," u *Encyclopedia Britannica.* Dostupno: https://www.britannica.com/topic/Super-Mario-Bros [pristupano 27.7.2020.].

[2] „The Legend of Zelda," u *Encyclopedia Britannica.* Dostupno: https://www.britannica.com/topic/The-Legend-of-Zelda-1688275 [pristupano 27.7.2020.].

[3] Nintendo, *NES Classic Edition: Developer Interview Metroid* [Na internetu]. Dostupno: https://www.nintendo.com/nes-classic/metroid-developer-interview/ [pristupano 27.7.2020].

[4] Super Metroid Players' Guide, Fareham: Nintendo U.K., 1994.

[5] „Castlevania: Symphony of the Night", u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/Castlevania:_Symphony_of_the_Night [pristupano 27.7.2020.].

[6] „Metroid", u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/Metroid [pristupano 27.7.2020.].

[7] „Castlevania", u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/Castlevania [pristupano 27.7.2020.].

[8] „14 Best Modern Metroidvania Games" (24.11.2018.). IGN. Dostupno: https://www.ign.com/articles/2018/11/24/14-best-modern-metroidvania-games [pristupano 27.7.2020.].

[9] Casltevania: Symphony of the Night karta [Slika] Dostupno: http://ww.finalfantasykingdom.net/sotnmaps.php [pristupano 27.7.2020.].

[10] Microsoft (20.7.2015.), *Introduction to the C# language and the .NET*. Dostupno: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework [pristupano 27.7.2020.].

[11] Unity Technologies, *Real-time solutions, endless opportunities*. Dostupno: https://unity.com/solutions [pristupano 28.7.2020].

[12] Unity Technologies, *Automotive, Transportation & Manufacturing, endless opportunities*. Dostupno: https://unity.com/solutions/automotive-transportation-manufacturing [pristupano 28.7.2020].

[13] „Unity Technologies“, u *Wikipedia, the Free Encyclopedia*. Dostupno: https://en.wikipedia.org/wiki/Unity_Technologies [pristupano 28.7.2020.].

[14] Unity Technologies, *Long Term Support*. Dostupno: https://unity3d.com/unity/qa/lts-releases [pristupano 28.7.2020.].

[15] A. Bowell, „Bolt visual scripting is now included in all Unity plans“, 22.7.2020. Dostupno: https://unity.com/solutions [pristupano 28.7.2020].

[16] aleksander (3.9.2020.), *Documentation, Unity scripting languages and you*. Dostupno: https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/ [pristupano 28.7.2020.].

[17] Microsoft, *Programming Languages*. Dostupno: https://code.visualstudio.com/docs/languages/overview [pristupano 28.7.2020.].

[18] Unity Technologies, *MonoBehaviour.Start()*. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html [pristupano 28.7.2020.].

[19] Unity Technologies, *MonoBehaviour.Update()*. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html [pristupano 28.7.2020.].

[20] Unity Technologies, *MonoBehaviour.FixedUpdate()*. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html [pristupano 28.7.2020.].

[21] Unity Technologies, *Quaternion.Euler*. Dostupno: https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html [pristupano 28.7.2020.].

[22] Blackthornprod (26.2.2018.), *2D DOUBLE / TRIPLE JUMP PLATFORMER CONTROLLER – EASY UNITY TUTORIAL*. Dostupno: https://www.youtube.com/watch?v=QGDeafTx5ug

[pristupano 29.7.2020]

[23] Unity Technologies, *Vector2*. Dostupno: https://docs.unity3d.com/ScriptReference/Vector2.html [pristupano 28.7.2020.].

[24] Blackthornprod (5.6.2020.), *How to make a 2D Wall Jump & Wall Slide using UNITY & C#!*. Dostupno: https://www.youtube.com/watch?v=KCzEnKLaaPc

[pristupano 29.7.2020]

[25] Unity Technologies, *Mathf.Clamp*. Dostupno: https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html [pristupano 28.7.2020.].

[26] Unity Technologies, *Quaternion.identity*. Dostupno: https://docs.unity3d.com/ScriptReference/Quaternion-identity.html [pristupano 28.7.2020.].

[27] Unity Technologies, *MonoBehaviour.Invoke*. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Invoke.html [pristupano 30.7.2020.].

[28] Unity Technologies, *GameObject.GetComponent*. Dostupno: https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html [pristupano 30.7.2020.].

[29] Unity Technologies, *Collider.OnTriggerEnter(Collider)*. Dostupno: https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html [pristupano 28.7.2020.].

[30] Unity Technologies, *MonoBehaviour.StartCoroutine*. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html [pristupano 30.7.2020.].

[31] Kap Koder (28.3.2020.), *Enemy Patrol/Wander AI Unity SIMPLE & EFFECTIVE CODE! Tutorial*. Dostupno: https://www.youtube.com/watch?v=MPnN9i1SD6g [pristupano 29.7.2020]

[32] Unity Technologies, *Collider2D.IsTouching*. Dostupno: https://docs.unity3d.com/ScriptReference/Collider2D.IsTouching.html [pristupano 29.7.2020.].

[33] Unity Technologies, *GameObject.activeInHierarchy*. Dostupno: https://docs.unity3d.com/ScriptReference/GameObject-activeInHierarchy.html [pristupano 29.7.2020.].

[34] Unity Technologies, *Random.Range*. Dostupno: https://docs.unity3d.com/ScriptReference/Random.Range.html [pristupano 29.7.2020.].

[35] Brackeys (31.1.2018.), *TILEMAPS in Unity*. Dostupno: https://www.youtube.com/watch?v=ryISV_nH8qw

[pristupano 29.7.2020]

[36] Blackthornprod (4.5.2018.), *HOW TO MAKE 2D PARTICLE EFFECTS – UNITY TUTORIAL* Dostupno: https://www.youtube.com/watch?v=_z68_OoC_0o&list=PLzgRnljanEDubyKs7SWHh3ixncAI8whRW&index=7&t=1s

[pristupano 29.7.2020]

[37] Unity Technologies, Time.timeScale. Dostupno: https://docs.unity3d.com/ScriptReference/Time-timeScale.html [pristupano 29.7.2020.].

[38] Unity Technologies, Object.FindObjectsOfType(). Dostupno: https://docs.unity3d.com/ScriptReference/Object.FindObjectsOfType.html [pristupano 29.7.2020.].

[39] Unity Technologies, *SceneManager.LoadScene*. Dostupno: https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html [pristupano 30.7.2020.].

[40] Dani (25.2.2019.), *Unity Parallax – How to infinite scrolling background*. Dostupno: https://www.youtube.com/watch?v=zit45k6CUMk

[pristupano 29.7.2020]

[41] Alexander Zotov (18.3.2017.), *How to add sound or audio effects SFX to Unity 2D arcade game | Very simple Unity 2D Tutorial*. Dostupno: https://www.youtube.com/watch?v=zit45k6CUMk

[pristupano 29.7.2020]