Razvoj platformske igre u okruženju Unity

Bošnjak, Mateo

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacije tehnologije

Mateo Bošnjak

Razvoj platformske igre u okruženju Unity

Završni rad

Pula, rujan, 2017. godine

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacije tehnologije

Mateo Bošnjak

Razvoj platformske igre u okruženju Unity

Završni rad

JMBAG: 0248039345, redoviti student

Studijski smjer: Informatika

Predmet: Napredne tehnike programiranja

Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan, 2017. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Mateo Bošnjak, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, rujan, 2017. godine

IZJAVA
o korištenju autorskog djela

Ja, Mateo Bošnjak dajem odobrenje Sveučilištu Jurja Dobrile
u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Razvoj platformske igre u okruženju Unity“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, rujan, 2017. godine

Potpis

_____

Sadržaj

1 Uvod

Današnja industrija razvoja igara je uvelike napredovala, u usporedbi s industrijom od prije par desetljeća. Na samim počecima industrije timovi koji su razvijali igre su znali sadržavati dva-tri člana. Nekada to čak nisu bili ni timovi nego je igru razvijao jedan programer.

Iza većine današnjih AAA naslova stoji veliki tim od 20-100 članova. AAA je akronim koji predstavlja jako puno vremena (**A** lot of time), jako puno resursa (**A** lot of resources) te jako puno novca (**A** lot of money). Tri „A“ prefiksa čine akronim AAA.

AAA naslovi zahtijevaju velike timove jer se od takvih igara očekuje da svake godine, s novim naslovima franšize, pomiču granice i donose nešto revolucionarno. Samim time ovakvi naslovi donose velike ekonomske rizike koji zahtijevaju jako velike prodaje kako bi se ostavrio profit.

Naravno tu su i nezavisni razvijači igara (indie game developers). Nezavisni razvijači igara predstavljaju razvijače koji nemaju financijsku pomoć izdavača. Nezavisni razvijači često sa svojim timovima od po 10 ljudi konkuriraju ogromnim tvrtkama u industriji igara.

U današnjoj industriji se nude razna okruženja kod razvoja igara. Osim onih okruženja koje tvrtke same razvijaju, nude se okruženja kao što su Unity, Unreal Engine te Cocos2d. Postoje tu jos i razni alati ali najbitniji od spomenutih bi bili Unity i Unreal Engine. Iako Unreal Engine možda nudi više i kvalitetnije, Unity je mnogim razvijačima privlačniji zbog svoje jednostavnosti.

U razvoju ovog projekta korišteno je okruženje/alata Unity. Unity je besplatno okruženje koje, pomoću svojega alata, razvijačima pruža olakšani način razvoja igara. Glasi kao jedan od najpopularnijih, ako ne čak i najpopularnije, okruženje/alat za razvoj igara. Svoju popularnost je stekao, osim toga što je besplatan, jednostavnošću koju pruža početnicima u razvoju igara.

U ovome projektu je bio cilj samostalno razviti igru korištenjem, već spomenutog, razvojnog okruženja Unity. Također je cilj bio vidjeti kakvi su to zadaci i prepreke koje se nalaze na putu do gotove igre, istražiti razne načine procesa razvoja, upoznati se s Unity okruženje te možda probuditi strasti za posve novo područje informatike.

U prvom poglavlju rada govorit će se o povijesti razvoja igara, kako je to izgledalo na samom početku te koje su prve igrače konzole. Također će se proći kroz žanrove igara.

U drugom poglavlju dotiče se alata koji se danas koriste. Koji su najpopularniji te najkorišteniji alati tj. okruženja za razvoj igara.

U trećem poglavlju slijedi opis igre.

U četvrtom poglavlju slijedi bolje upoznavanje s Unity alatom. Kratak opis, način na koji funkcionira te jezici s kojima se može raditi u Unity-u.

U petom poglavlju će se govoriti o problemima tijekom izrade projekta te strukturi samog projekta.

U šestom poglavlju slijedi popis svih skripti koje se nalaze u projektu te detaljniji opis najbitnijih skripti.

U sedmom poglavlju se nalazi opis pojmova kao što su „Animator“, „Collider“ te „Ray Casting“.

U osmom poglavlju dotiče se proces izrade nivoa tj. izgled terena pojedinih nivoa.

2 Razvoj igara i njihova povijest

2.1 Povijest

Prvi prihvaćeni primjer stroja za igranje je predstavljen od strane doktora Edwarda Uhlera Condoda na svjetskom sajmu u New Yorku 1940-te godine (Wired, 2010). Igra, koja je bila bazirana na drevnoj matematičkoj igri zvanoj Nim, je bila odigrana od strane 50 000 ljudi. Stroj je protiv čovjeka pobijedio u preko 90% odigranih igara (Pisano, 2015).

Međutim, prvi igrački sistem dizajniran za komercijalnu upotrebu je izumljen skoro tri desetljeća kasnije, kada su Ralph Baer i njegov tim, 1967. godine, napravili prototip „Brown Box“ (Donovan, 2010).

„Brown Box“ je bila vakuumska cijev koja se mogla spojiti na televiziju, omogućavala je igračima kontroliranje kocki kojima su se međusobno lovili po ekranu. „Brown Box“ je mogao biti programiran kako bi se igrale različite vrste igara.

„Brown Box“ je bio licenciran od strane Magnavoxa, koji je sustav objavio 1972. godine, pod nazivom „Magnavox Odyssey“ (Wolf, 2012). Sustav je objavljen par mjeseci prije Ataria, za kojeg mnogi ljudi misle da je bio prvi igrački sustav.

1972. godine, Atari je postao prva veća igračka tvrtka koja je postavila ljestvicu za veću igračku zajednicu (Stanton, 2015).

Atari je prva tvrtka koja je prodavala pravu elektroničku igru pod nazivom „Pong“ (Burnham, 2001). Arkadni strojevi su se, nakon izuma Ponga, počeli pojavljivati svugdje. Mnogi ljudi su shvatili da je igračka industrija nešto veliko; između 1972. i 1985. godine, više od 15 različitih tvrtki je započelo razvijati igre za brzo rastuće tržište.

2.3 Žanrovi

Većina igara spada u neke kategorije žanrova. Poneke igre mogu spadati i pod više žanrova dok neke igre pokušavaju izmisliti nešto novo pa ne mogu biti svrstane ni pod koji žanr. Ako igra, koja ne može biti svrstana ni pod koji žanr, postane jako popularna druge igre je pokušaju kopirati, te time stvore novi žanr. U nastavku slijedi popis nekih žanrova, opis žanra te primjer igara koje spadaju u taj žanr.

Pucačina (shooter)

Jedan od najstarijih žanorva igara je pucačina. Prva igra koja spada u tu kategoriju je igra „Spacewar!“ koju je izdao Steve Russell 1960-ih godina (Kent, 2001). Pucačina je žanr igre u kojoj igrač mora upucati protivnike ili objekte kako bi nastavio s igrom.

Slika 1. Dan Edwards i Peter Samon igraju igru „Spacewar!“

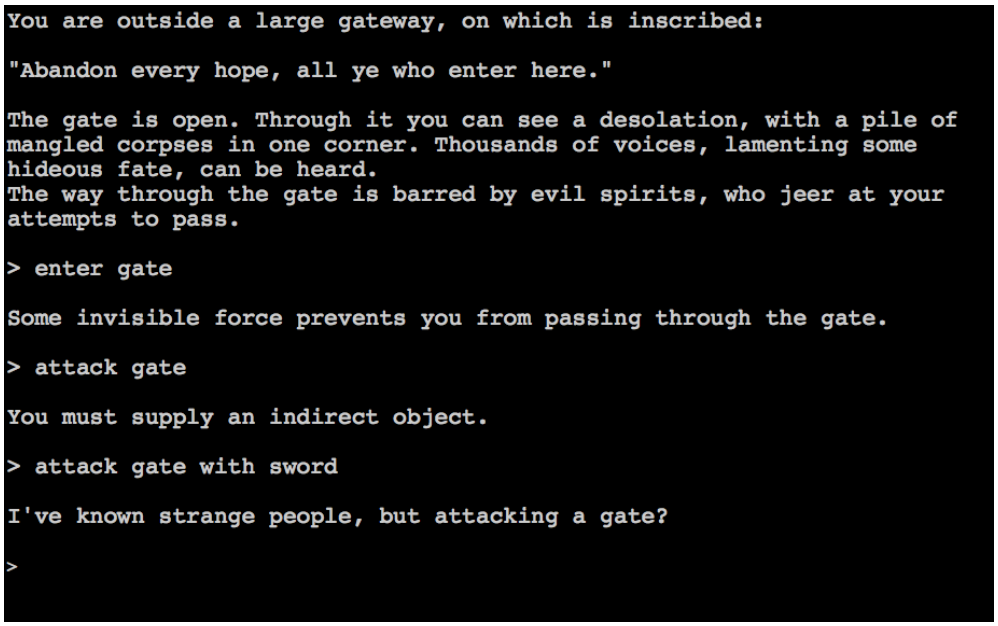
Pucačina iz prvog lica (First person shooter)

Pucačina iz prvog lica je primjer podžanra koji je postao toliko popularan da se smatra zasebnim žanrom. Pucačina iz prvog lica je žanr sličan pucačini, jedina razlika je da igrač igra iz perspektive prvog lica. Iako je „Wolfenstein 3D“ prva igra ovog žanra, tek nakon izlaska igre „Doom“ ljudi su primijetili potencijal ovakvog tipa igre. Doom je bila i prva igra koja je omogućila da više igrača igra zajedno/protiv. Neki primjeri bi bili Wolfenstein, Doom, Battlefield, itd.

Slika 2. Slika ekrana igre „Doom 1“

Avantura (Adventure)

Još jedan od prvih žanrova. Na početku su ovakve igre bile tekstualno bazirane, ali su kasnije s razvitkom tehnologije postale slikovne. „Zork“ igre su bile najpopularnije tekstualno bazirane avanture (Lowood, 2004). Neke od avantura koje su bile vizalne a ne tekstualne su „Indiana Jones“, „Monkey Island“, „Myst“ i mnoge druge.



Slika 3. Slika ekrana igre „Zork III: The Dungeon Master“

Platformerske (Platform)

Platformerski žanr je započeo 1981. godine kada su izdane igre „Donkey Kong“ i „Space Panic“. Platformerske igre su igre u kojima je bitan timing i skakanje kako bi igrač došao do svog odredišta. Jedna od najpopularnijih platformera je „Super Mario Bros“. S razvitkom tehnologije platformeri su postajali i 3D. Neke od najpopularnijih 3D platformera su „Crash Bandicoot“, „Pac-Man World“ te „Super Mario 64“.

Igre igranja uloga (Role-Playing games - RPG)

Ovakav žanr igre se razvio od igre „Dungeons and Dragons“ koja se igrala na papiru (Ewalt, 2013). Igre igranja uloga sadrže sljedeće ključne elemente:

1. Određeni cilj misije
2. Proces u kojem igrač razvija svog lika
3. Pažljivo baratanje predmetima koje lik ima sa sobom (oružja, hrana, alat,…)

Primjeri ovog žanra bi bile „Baldru's Gate“, „Diablo“, „The Witcher“, „The Elder Scrolls“, itd.

Slika 4. Slika ekrana igre „The Witcher 3“

Zagonetke (Puzzle)

Igre zagonetnog žanra se ne razlikuju previše od klasičnih zagonetki. Najveća razlika je ta što igre takvog tipa ne mogu biti pretočene u dnevni boravak. Primjer takve jedne igre je „Wetrix“, igra u kojoj igrač gradi zidove kako bi u njima sačuvao određenu količinu vode. Drugi primjeri igara ovog žanra bi bile „Tetris“, „Puzzle Boble“, „Portal“, itd.

Simulacije

Simulacije su igre koje, što preciznije, pokušavaju stvoriti nešto iz stvarnog svijeta. Takve igre mogu upravljati nečime, kao npr. „SimCity“, ili mogu biti poput igara kao što su „Microsoft Flight Simulator“ ili „Gran Turismo“.

Strateške/Taktičke

Strateške igre su slične igrama simulacije, ali su u većini slučajeva bazirane na potezima kako bi igrač dobio osjećaj što veće kontrole. Primjer ovakvih igara su „Ogre Tactics“, „Command and Conquer“, „Worms“, itd.

Slika 5. Slika ekrana igre „Worms 2: Armageddon“

Sportske

Kao što i sam naziv govori ovakav tip igara pokušava simulirati sport. Primjeri ovakvog tipa žanra su „FIFA“, „PES“, „NBA“, „Madden“, itd.

Borilačke (Fighting)

Ovakve igre omogućuju da se dva igrača sukobe u virtualnom okruženju. Većina igara nudi i kampanju ali najprimamljivije su borbe među dva igrača. Primjer ovakve igre su „Street Fighter“, „Mortal Kombat“, „Tekken“, itd.

Slika 6. Slika ekrana igre „Mortal Kombat 1“

Plesačke/Ritam

Dance Dance Revolution je najveća igra u ovom žanru. Ovakve igre zahtijevaju od igrača neke određene radnje u skladu s ritmom. Još neki primjeri bi bili „Parappa the Rapper“, „Bust a Groove“, „Gitaroo Man“, „Guitar Hero“, itd.

Horor preživljavanja (Survival Horror)

Kao što naziv sugerira, ovaj žanr je nastao od horor žanra. Glavni dio igre je preživljavanje u okruženju u kojem se nalaze zastrašujući ili uznemirujući elementi, koji mogu biti natprirodnog ili fantazijskog tipa.

3 Alati koji se danas koriste

U današnjoj industriji razvijanja igara postoje dvije mogućnosti. Jedna mogućnost je razvijanje vlastitog pokretača (engine), dok je druga mogućnost korištenje tuđeg, već razvijenog pokretača. Većina velikana ovakve industrije se odlučuje na prvu mogućnost. Primjer bi bile tvrtke poput „EA“ i „CDProjectRed“, koji imaju svoj „FrostBite“ odnosno „REDEngine“ pokretač. Druge tvrtke koje nemaju toliko novaca se odlučuju za pokretač koji im se nude na tržištu. Neki od pokretača koji se nude na tržištu su „Unity“, „Cocos2d“ te „Unreal Engine“. U nastavku slijedi opis navedenih pokretača.

3.1 Unity

Unity glasi kao jedan od najpopularnijih alata za izrađivanje video igara. U provedenoj anketi bloga „Developer economics“, čak 47% ispitanika koriste Unity u razvoju svojih igara (Wilcox, 2014). Unity nudi podršku za razvoj 2D i 3D igara. Na samom začetku Unity je bio namijenjen razvijanu 3D igara dok su 2D značajke bile tu samo radi izrade izbornika. Mnogi ljudi su koristili 2D mogućnosti, iako su bile osnovne, za izradu igara pa je stoga Unity odlučio proširiti ponuđene mogućnosti.

Jezike koje Unity podržava su C#, UnityScript (zapravo JavaScript) te Boo. Zadnji spomenuti, Boo, se jako malo koristi pa ga većina ljudi izbjegava. Netko bi pomislio da je UnityScript, zbog svoga imena, popularniji i korišteniji ali zapravo je obratno. C# je korišteniji te se koristi u velikom dijelu većih plugin-ova.

Prednosti:

- Velika ponuda resursa koje zajednica stvara
- Privlačan izgled vizualnog editora
- Podržava veliki broj platformi (mobilne, desktop, web te konzole)
- Jednostavno razvijati igru za veći broj platformi istovremeno
- Besplatna licenca nudi veliki broj mogućnosti

Mane

- Ne postoji dovoljno dobar način za dijeljenje datoteka, korisnici se moraju okretati servisima kao što su „GitHub“
- Performanse nisu dobre – do nedavno Unity je koristio samo jednu dretvu. To se promijenilo u petoj verziji Unity-a
- Izvorni kod pokretača nije dostupan. Što znači ako korisnik naiđe na grešku u pokretaču, neće je moći sam popraviti nego će morati čekati novu verziju Unity-a.

3.2 Cocos2d

Cocos2d je pokretač namijenjen za razvoj igara u 2D okruženju. Nastao je u istom razdoblju kao i iPhone SDK. Brzo se prebacio na objektivni C, te je u međuvremenu postao najpopularniji besplatni pokretač otvorenog koda za mobilne igre.

Dostupne su različite verzije pokretača koje podržavaju objektivni C, C++, C#, Java, JavaScript i Ruby. C++ verzija se najviše održava, pa stoga i podržava najveći broj platformi. U C++ verziji su i skriptni jezici Lua i JavaScript, pa korisnici i tu mogu izabrati jezik koji im više odgovara.

Prednosti

- Veliki broj podržanih platformi, većinom mobilnih
- Besplatno i tipa otvorenog koda
- Veliki broj ekstenzija i alata
- Velika zajednica koja pruža razne resurse

Mane

- Ne postoji netko to popravlja greške u pokretaču, osim zajednice
- Ne potiče se dobra struktura koda te se zbog toga često nailazi na kod koji je težak za održavanje

3.3 Unreal Engine 4

Unreal Engine je dugo godina bio prvi odabir kod razvoja 3D igara za računala i konzole. Treća generacija je podržava i mobile platforme. Četvrta verzija pokretača je nudila licence, koje uključuju izvorni kod, za bilo koga s cijenom od 19 dolara po mjesecu plus 5% udjela od prihoda.

Pokretač je napisan u C++, te je s time i jedini podržani jezik. Iako je moguće napraviti relativno dosta razvoja korištenjem „Blueprinta“. „Blueprint“ je vizualno programersko okruženje u kojima se node-ovi spajaju pomoću linija.

Prednosti

- Ostvaruje nevjerojatne performanse
- Nude se vrhunski alati za bilo koje faze razvoja igre
- Dostupnost izvornog koda omogućuje prilagođene ekstenzije te popravak grešaka u pokretaču

Mane

- Razvoj u C++, jezik koji nije namijenjen za početnike
- Krivulja učenja je puno teža od npr. krivulje za učenje kod Unity-a
- Ograničen broj podržanih starijih uređaja

4 Opis igre

Naziv igre je „Indiana Ford“. Žanr pod koji bi igra spadala bi bio platformer. Glavni zadatak igrača je da prođe kroz niz nivoa kako bi se na kraju igre sukobio s finalnim protivnikom. Na putu do zadnjeg nivoa naći će se prepreke u obliku neprijatelja i izazovno dizajniranih nivoa.

Priča igre je smještena u nepoznatom drevno gradu stare civilizacije maja. Lik igre je u potrazi za drevnim artefaktom koji svome korisniku pruža veliku moć. U potrazi za dnevnim artefaktom lik se susreće s nepoznatim bićima koji čuvaju artefakt.

Glavna mehanika igre je skakanje. Igrač pomoću skokova i skokova sa zidova pronalazi svoj put kroz nivo. U svakom nivou se nalazi ključ kojeg igrač mora pokupiti prije nego se uputi na sljedeći nivo. Na svakom nivou se nalazi određeni broj novčića koje igrač može pokupiti, svaki novčić pridonosi bodove konačnom rezultatu. Igrač susreće nekoliko različitih protivnika na putu do zadnjeg nivoa. Prepreke predstavljaju bodlje, lebdeći protivnici te toranj koji ispucava projektile. Na zadnjem nivou igrač susreće konačnog finalnog protivnika kojega mora poraziti kako bi pronašao drevni artefakt.

5 Alat Unity

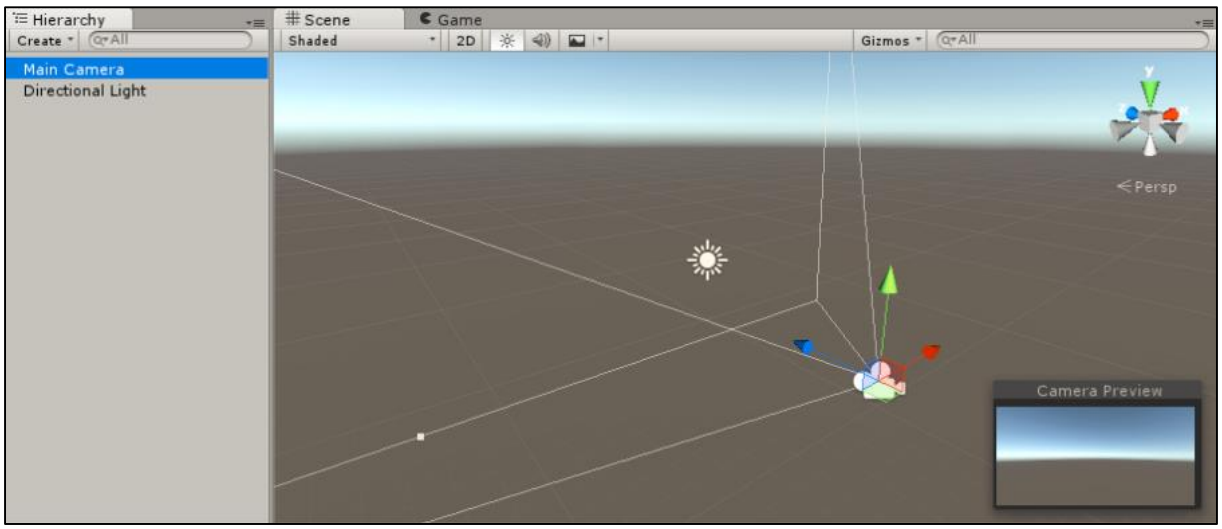
5.1 Opis alata

Kao što je već ranije spomenuto, Unity je jedan od najpopularnijih alata za izrađivanje video igara. Popularan je zbog svoje jednostavnosti te orijentiranost prema većem broju platformi. Veliki broj uputa i resursa od zajednice, te jednostavnost i privlačnost početnicima u razvoju igara su glavni faktori zbog kojih se veliki broj korisnika odlučuje na korištenje Unity-a.

Razlog odabira Unity-a je njegova jednostavnost i privlačnost korisnicima. Nije pretjerano težak alat za početnika u razvoju igara. Pruža veliki broj uputa i drugih raznih resursa koje nudi Unity zajednica.

5.2 Scene i objekti

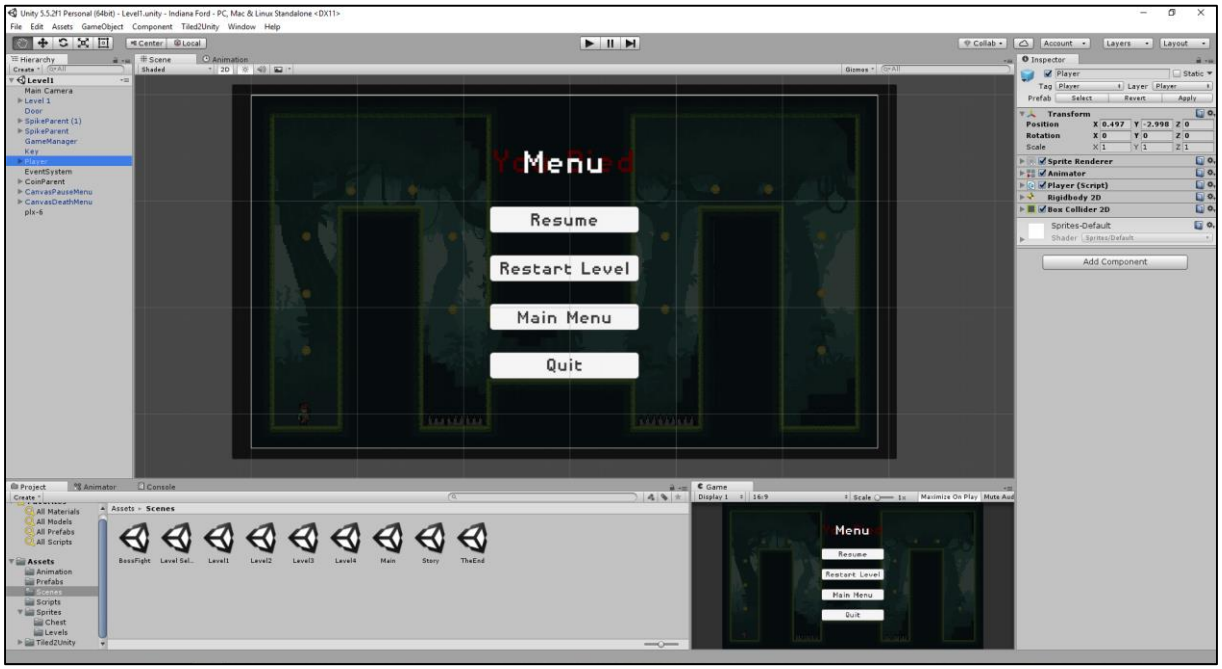
Glavna komponenta Unity je scena. Scena može predstavljati nivo igre. Kada se igra pali, automatski se pokreće zadana početna scena. Ta scena može biti neka animacija, npr. logo tvrtke koja je proizvela tu igru, ili može biti početni izbornik. U sceni se nalaze sve ostale komponente koje čine igru.



Slika 7. Početna scena

Na slici 7 se može vidjeti izgled scene kada se kreira novi projekt. Na lijevoj strani slike se nalazi hijerarhija gdje su smješteni svi objekti koji se nalaze u trenutnoj sceni. U ovom slučaju tu je objekt za kameru te za svijetlo.

Unutar scene korisnik stvara objekte koji mogu, a i ne moraju, imati međusobne interakcije. Na svaki objekt korisnik može dodati određene komponente, kao što su komponente za grafiku, fiziku, skripte, itd. Komponente objektima omogućuju međusobnu interakciju.



Slika 8. Inspektor objekta

Na desnoj strani slike 8 može se vidjeti inspektor za trenutno označeni objekt. Pod inspektorom se nalaze komponente koje su pridružene tom objektu. U ovom slučaju je označeni objekt „Player“, tj. Igrač. Neki od komponenata koje se nalaze u inspektoru objekta su „Rigidbody2D“ (komponenta zaslužena za fiziku objekta), „Boxcollider“ (komponenta koja omogućuje interakciju objekta s drugim objektima) te skripta „Player“ (koja kontrolira objekt pomoću funkcija napisane u C#).

Skripte su komponente koje pomoću programskog koda, C# ili JS, kontroliraju ponašanje objekta. Raznim naredbama unutar skripte se mogu kontrolirati objekti. Može se mijenjati brzina objekta, njegova veličina, smjer, smještaj unutar koordinatnog sustava, itd.

5.3 C# vs JavaScript

Skripte koje u Unity-u kontroliraju razne objekte mogu biti pisane u dva programska jezika. Zapravo je tu i treći programski jezik Boo, ali je on toliko malo korišten da se može i zanemariti. Tako da odluka korisnika pada na C# ili JavaScript.

Korisnici koji su se već prije upoznali s C-like jezicima (C++, Java, itd.), će se vjerojatno odlučiti na C#. Dok će se korisnici kojima je dinamički jezik poput PHP-a draži, odlučiti na JavaScript.

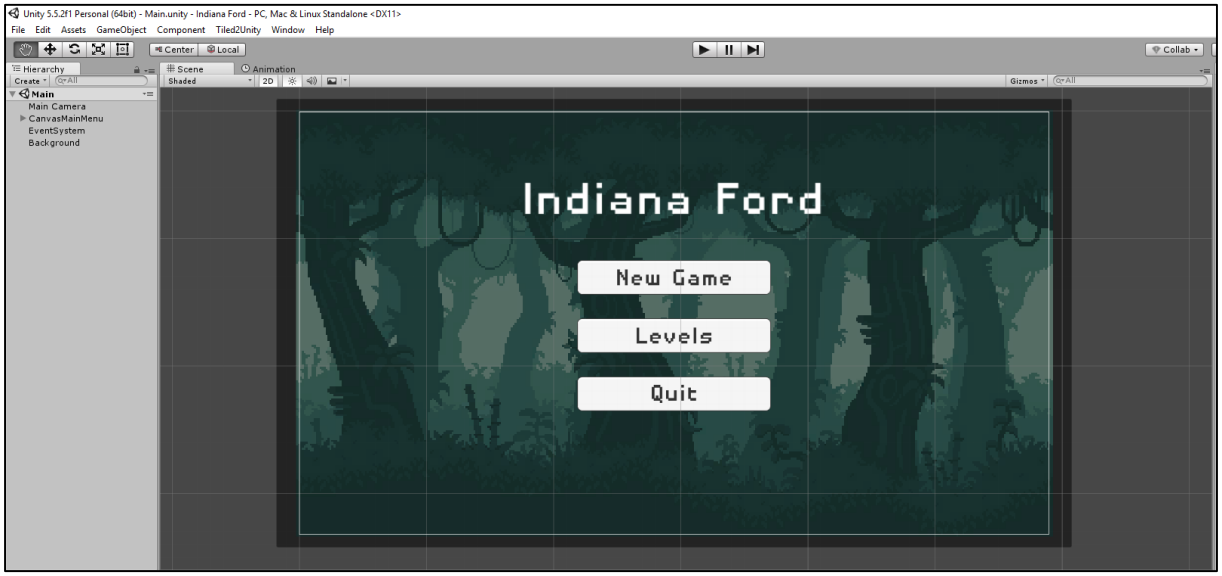
Velika prednost C# jezika je njegova organiziranost, kao i kod većine objektno orijentiranih jezika. Još jedna prednost C# je ta da je veliki broj resursa (tutoriali, alati, ekstenzije) napisan prvenstveno za C#. Mana leži u tome da je, u usporedbi s JavaScriptom, C# teži jezik za početnike. Prednost JavaScripta je njegova jednostavnost, koja često zna biti privlačna početnicima u programiranju. Nedostatak JavaScripta je zastupljenost u zajednici te činjenica da može doći do loše organiziranosti koda zbog dinamičnosti jezika.

6 Opis problema

U nastavku slijedi opis projekta, njegova struktura i sve njegove komponente, te opis svih problema tijekom razvoja projekta. Prvo slijedi objašnjenje scena koje se nalaze u projektu, nakon scena će biti objašnjenje sve komponente koje sadrže pojedine scene te skripte koje upravljaju svim komponentama.

6.1 Scene

Postoje tri glavne scene u projektu. To su izbornik (na početku ulaska u igru), nivo (ukupno se nalazi pet različitih nivoa, ali će samo jedan od njih biti pobliže objašnjen) te scena koja sadrži priču u kojoj je smještena igra.



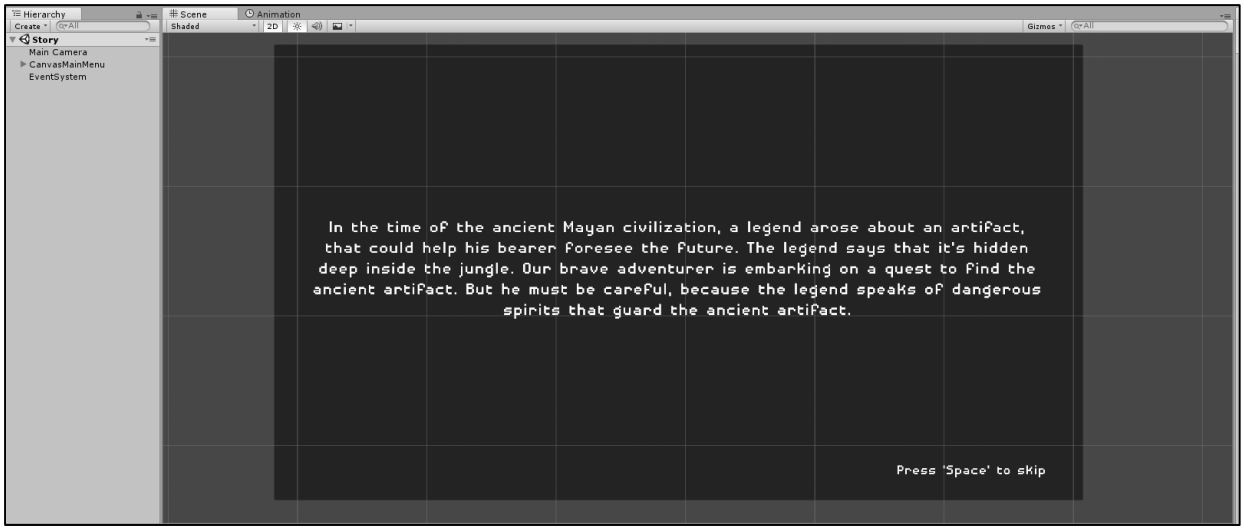
Slika 9. Izgled scene za izbornik

Na slici 9 je prikazan izgled scene za izbornik. Na lijevoj strani je prikazana hijerarhija scene, tj. svi objekti koji se nalaze unutar scene. Objekti koji se nalaze u sceni su „MainCamera“, „CanvasMainMenu“, „EventSystem“ te „Background“. Pod objektom „CanvasMainMenu“ se nazale gumbovi. Objekt „MainCamera“ služi kao kamera scene, tj. ona predstavlja ono sto igrač vidi kada igra igru. „CanvasMainMenu“ je objekt u kojem se nalaze gumbovi, on sačinjava izbornik na ovoj sceni. „EventSystem“ je objekt kojeg koristi „CanvasMainMenu“, on je kreiran uz kreiranje canvasa. „Background“ je objekt koji predstavlja pozadinu ove scene.

Gumbovi u objektu „CanvasMainMenu“ omogućuju korisniku navigaciju kroz igru. Izbornik pruža opcije za započinjanje nove igre, odabir levela ili izlazak iz igre.

Slika 10. Izgled scene za odabir nivoa

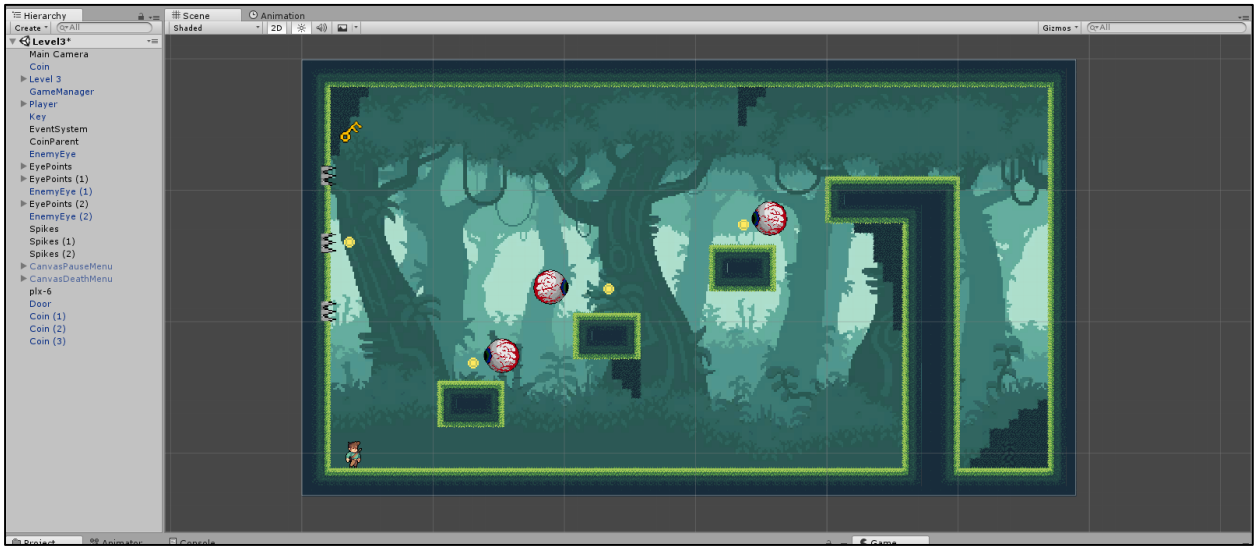
Gumb „Levels“ vodi igrača na novi izbornik, prikazan na slici 10, u kojem može birati nivo za igranje.



Slika 11. Izgled scene za priču

Na slici 11 je prikazan izgled scene za priču. Na lijevoj strani slike se nalazi hijerarhija scene. Unutar scene se nalaze objekti „MainCamera“, „CanvasMainMenu“ te „EventSystem“. Kao i u prethodnoj sceni „MainCamera“ predstavlja ono što igrač vidi u toj sceni. „CanvasMainMenu“ u ovom slučaju sadrži objekt u kojem se nalazi tekst priče, te

pruža korisniku mogućnost da preskoči ispis priče pritiskom gumba „space“. „EventSystem“ je objekt kojeg koristi „CanvasMainMenu“.



Slika 12. Izgled scene za nivo tri

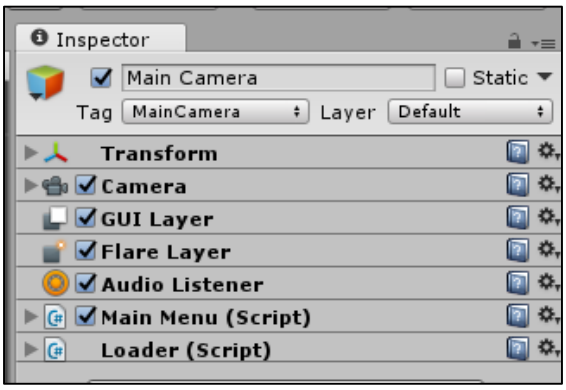
Na slici 12 je prikazan izgled scene za jedan od nivoa, u ovom slučaju je prikazan nivo tri. Na lijevoj strani slike je prikazana hijerarhija scene. Objekti koji se nalaze u sceni, a nisu već prije spomenuti, su „GameManager“, „Player“, „Coin“, „Key“, „EnemyEye“, „Spikes“ te „Door“. „GameManger“ je objekt koji kroz cijeli tijek igre ima samo jednu instancu, u odjeljku za objekte će biti pobliže objašnjen. „Player“ je objekt koji predstavlja lika u sceni. Sve njegove radnje su opisane unutra skripte koja ga kontrolira. „Coin“ je objekt koje igrač može pokupiti kako bi povisio svoj konačni rezultat. „Key“ je objekt koji otključava vrata nivoa. Korisnik ga mora pokupiti ako želi nastaviti na sljedeći nivo. „EnemyEye“ je objekt koji predstavlja protivnika. Njegovo ponašanje je opisano u skripti. „Spikes“ je objekt koji predstavlja protivnika, ali u ovom slučaju statičnog. Zadnji objekt „Door“, predstavlja vrata kroz koja igrač ide na sljedeći nivo. Vrata se otvaraju onda kada korisnik pokupi ključ na tom nivou.

U nastavku slijedi detaljno objašnjenje svih objekata koji se nalaze unutar scena, tj. unutar projekta.

6.2 Objekti

Sve stvari u scenama, od lika do predmeta koje lik može pokupiti, predstavljaju objekti. U nastavku slijedi detaljno objašnjenje za sve objekte koje se nalaze unutar igre.

Main Camera

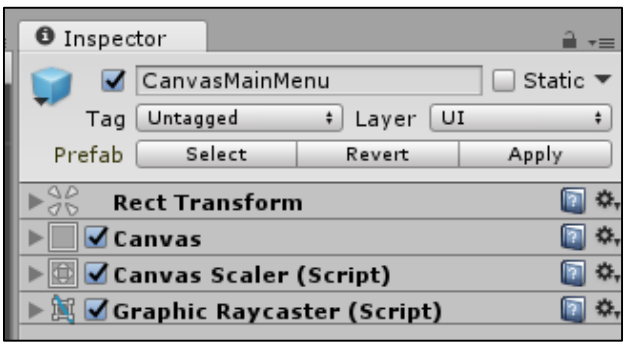


Slika 13. Inspektor za objekt MainCamera

Objekt „MainCamera“ predstavlja sliku koji igrač vidi kada igra igru. Na slici broj 13 je prikazan inspektor za objekt „MainCamera“. U inspektoru se nalaze sve komponente koje označeni objekt sadrži. Sve komponente, osim dvije komponente za skripte, su stvorene kada je objekt kreiran. Skripta „MainMenu“ služi za navigaciju kroz izbornik, dok skripta „Loader“ služi za inicijaliziranje objekta „GameManager“.

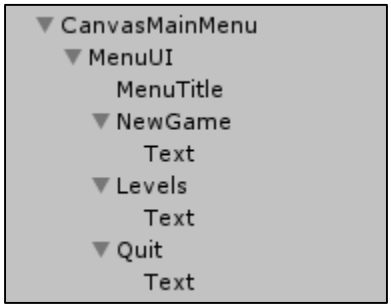
CanvasMainMenu

Objekt „CanvasMainMenu“ sadrži gumbove za navigaciju kroz igru. Nudi igraču gumbove za novu igru, odabir nivoa te izlazak iz igre. U igri se pojavljuju varijacije objekta pod nazivom „CanvasPauseMenu“ i „CanvasDeathMenu“. Nude slične mogućnosti kao i „CanvasMainMenu“.



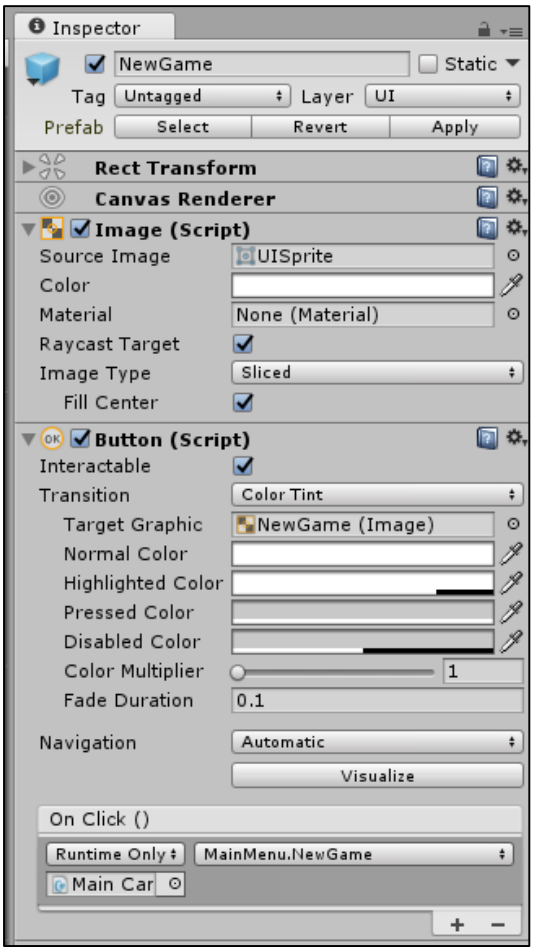
Slika 14. Inspektor za objekt CanvasMainMenu

Na slici broj 14 je prikazan inspektor za objekt „CanvasMainMenu“. Sve komponente su kreirane kada je objekt stvoren.



Slika 15. Hijerarhijski izgled objekta CavnasMainMenu

Na slici 15 je prikazan hijerarhijski izgled objekta „CanvasMainMenu“. Sadrži objekte koji predstavljaju gumbe.

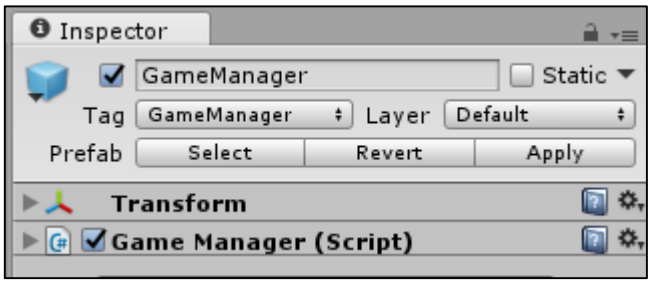


Slika 16. Inspektor objekta

Na slici 16 je prikazan inspektor za objekt koji predstavlja gumb. Pod komponentom button se nalazi „On Click“ metoda, koja se okida kada igrač klikne gumb.

Game Manager

Objekt „Game Manager“ je objekt koji se inicijalizira na početku igre te se uništava na kraju. Tijekom igre postoji samo jedna kopija tog objekta. Pomoću njega se kontrolira igra, odlučuje se koji je sljedeći nivo, te se prati rezultat igrača tijekom igre.

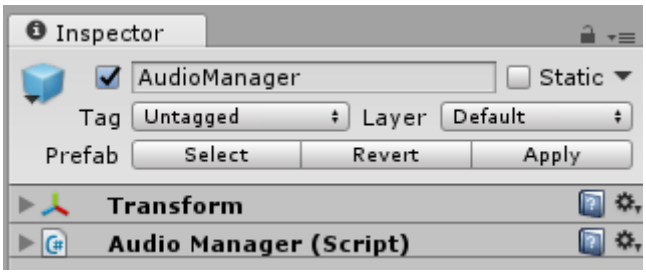


Slika 17. Inspektor za objekt GameManager

Na slici 17 je prikazan inspektor za „Game Manager". Dodana je jedna komponenta, skripta. Sve skripte na objektima će biti pojašnjene u nastavku.

Audio Manager

Objekt „Audio Manager“ je objekt sličan objektu „Game Manger“. Jedina razlika je u tome što je on zaslužan za zvuk u igri.

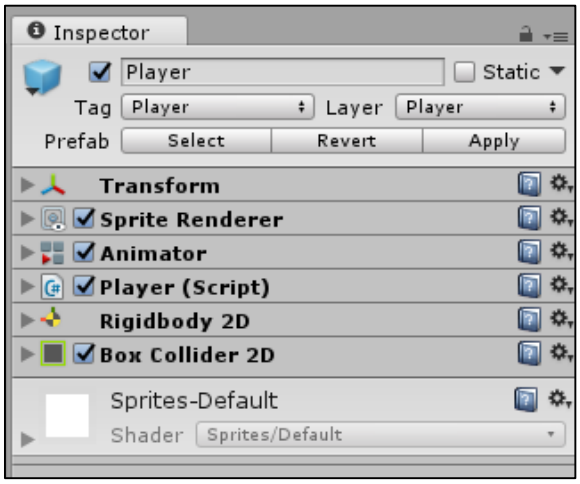


Slika 18. Inskektor za objekt AudioManager

Na slici 18 je prikazan inspektor za objekt „Audio Manager“.

Player

Objekt „Player“ je objekt koji predstavlja lika koje igrač kontrolira.

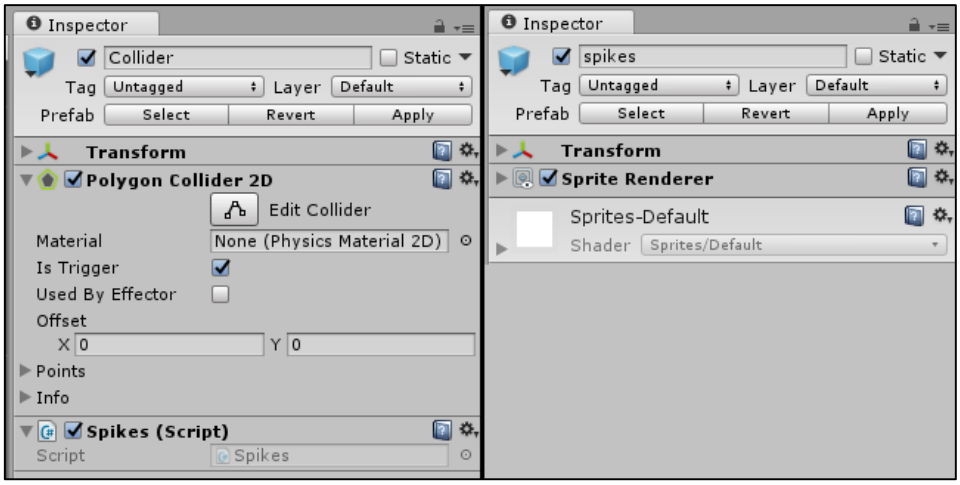


Slika 19. Inspektor za objekt Player

Na slici 19 je prikazan inspektor za objekt „Player“. Komponenta „Sprite Renderer“ služi za grafiku lika. Animator je komponenta zaslužna za animaciju. Skripta „Player“ je skripta pomoću koje igrač kontrolira objekt. „Rigidbody 2D“ je komponenta koja objektu daje fizička svojsta (masu, gravitaciju, itd.). „Box Collider 2D“ je komponenta koja objektu omogućuje interakciju s drugim objektima. Da nema te komponente, objekt bi samo „propao“ kroz teren.

Spike

Objekt Spike predstavlja bodlje koje se nalaze na pojedinim nivoima.

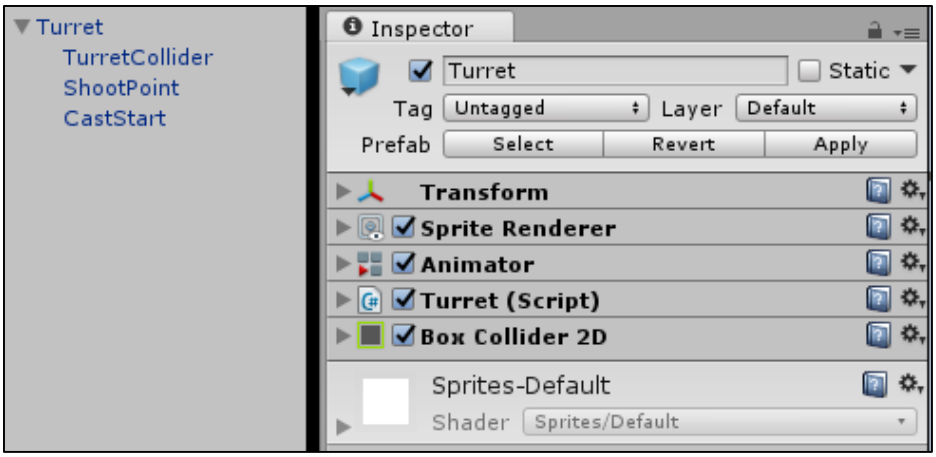


Slika 20. Inspektor za hijerahijsku grupu Spike

Na slici 20 je prikazan inspektor hijerarhije objekta. Objekt sadrži dva objekta. Jedan objekt koji predstavlja collider, te tri duplikata objekta koji predstavlja bodlje. Polygon Collider 2D je okidač. Collider okidač služi kako bi se provjerilo da li je neki drugi objekt ušao u područje collider-a.

Turret

Objekt predstavlja neku vrstu tornja koji ispucava projektile na igrača.

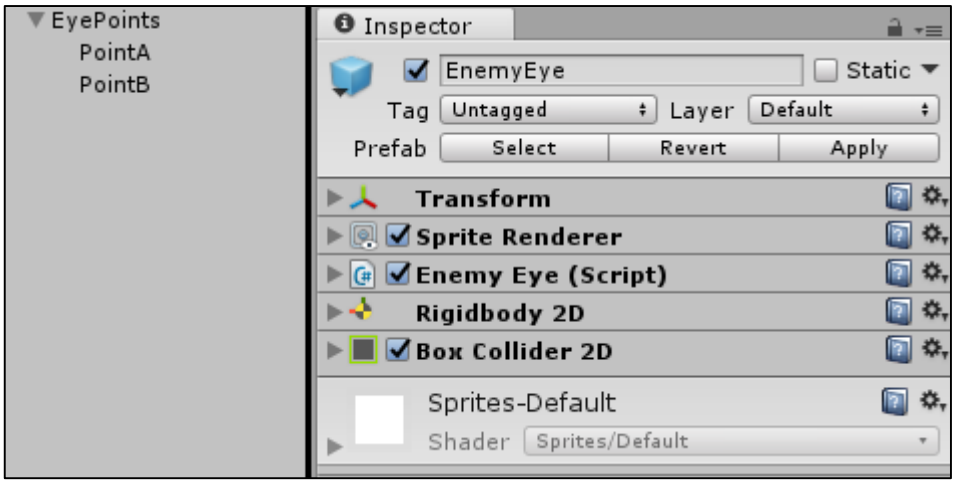


Slika 21. Inspektor za objekt Turret

Na slici 21 na desnoj strani je prikazan inspektor za objekt. Objekt sadrži komponentu za skriptu, komponente „Sprite Renderer“, „Animator“ te „Box Collider 2D“. Komponenta „Box Collider 2D“ je okidač. Na lijevoj strani slike se nalazi hijerarhija objekta. Objekt sadrži tri podređena objekta, objekt „ShootPoint“, „CastStart“ te „TurretColider“. Objekt „ShootPoint“ svojim položajem određuje na kojem se mjestu stvaraju projektili, dok objekt „CastStart“ određuje poziciju gdje započinje „Ray Cast“. „Ray Cast“ će biti kasnije pobliže pojašnjen.

EnemyEye

Objekt „EnemyEye“ predstavlja protivnika. Protivnik se kreće horizontalno od jedne do druge točke.

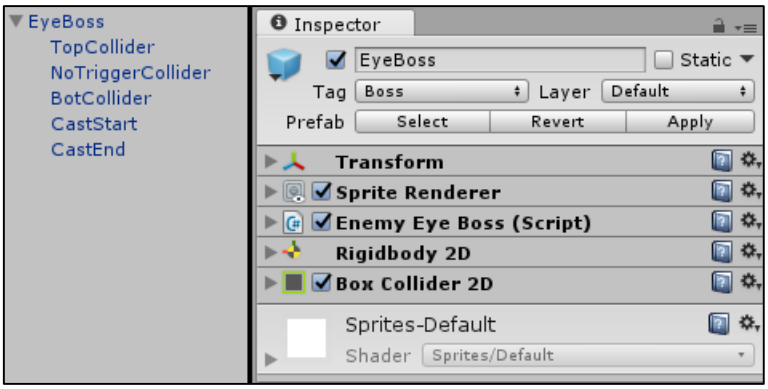


Slika 22. Inspektor za objekt EnemyEye

Na slici 22 na desnoj strani prikazan je inspektor objekta. Objekt sadrži „Sprite Renderer“, skritpu, „Rigidbody 2D“ te „Box Collider 2D“. Na lijevoj strani je prikazana hijerarhija za objekt „EyePoints“. Objekt „EyePoints“ sadrži dva podređena objekta koji određuju područje po kojem se objekt „EnemyEye“ može micati.

EyeBoss

Objekt „EyeBoss“ predstavlja finalnog protivnika u igri.



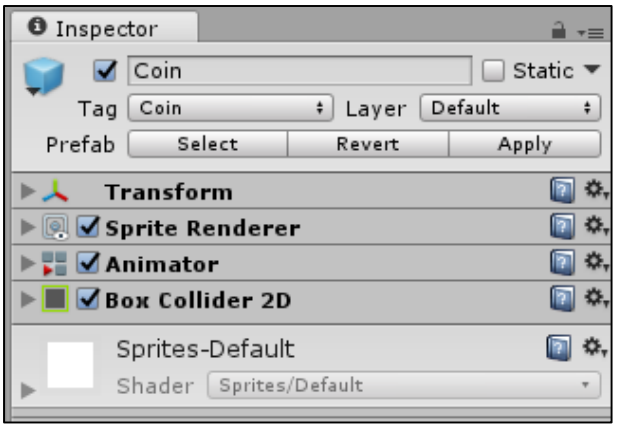
Slika 23. Inspektor za objekt EyeBoss

Na slici 23 na desnoj strani, prikazan je inspektor za objekt. Komponente koje objekt sadrži su „Sprite Renderer“, skripta, „Rigidbody 2D“ te „Box Collider 2D“. Na lijevoj strani

je prikazana hijerarhija objekta. Objekt sadrži objekte „TopCollider“, „BotCollider“, „NoTriggerCollider“, „CastStart“ te „CastEnd“. „TopCollider“ i „BotCollider“ su okidači. „TopCollider“ služi za registraciju udaraca od strane igrača, dok „BotCollider“ služi za registraciju udarca od strane protivnika. „NoTriggerCollider“ je collider koji služi kako protivnik ne bi „propao“ kroz teren.

Coin

„Coin“ objekt predstavlja novčić kojeg igrač može skupljati prilikom prelaska levela.



Slika 24. Inspektor za objekt Coin

Na slici 24 je prikazan inspektor objekta. Objekt sadrži „Sprite Renderer“, „Animator“ te „Box Collider 2D“. „Collider 2D“ predstavlja trigger.

Key

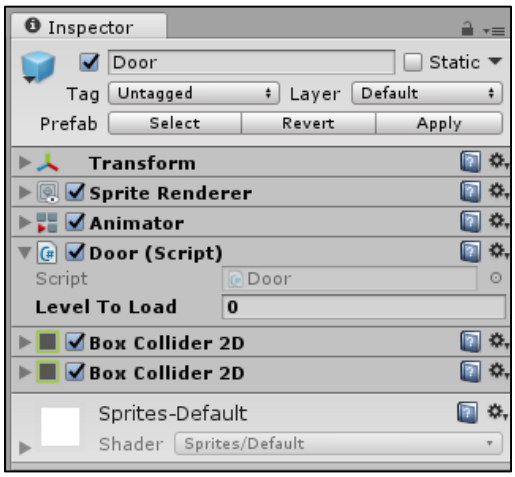
Objekt „Key“ predstavlja ključ kojeg igrač mora pokupiti kako bi prešao na sljedeći nivo.

Slika 25. Inspektor za objekt Key

Na slici 25 je prikazan inspektor. Objekt sadrži „Sprite Renderer“, „Box Collider 2D“, „Animator“ te skriptu.

Door

Objekt „Door“ predstavlja vrata kroz koja igrač mora proći kako bi nastavio na sljedeći nivo.



Slika 26. Inspektor za objekt Door

Na slici 26 je prikazan inspektor. Komponente koje objekt sadrži su „SpriteRenderer“, „Animator“, skripta te „Box Collider 2D“. „Box Collider 2D“ je u ovom slučaju trigger.

7 Skripte

Skripte omogućuju korisniku da programira ponašanje objekata koji se nalaze u igri. Kao što je već ranije spomenuto, skripte mogu biti napisane na tri različita programska jezika. Sve skripte u projektu su napisane u C#. U nastavku slijedi detaljan opis skripta koje se nalaze u projektu.

Slijedi popis svih skripti koje se nalaze u projektu te kratko objašnjenje za što pojedina skripta služi. Nakon popisa skripti slijedi objašnjenje glavnih mehanika igre.

- Chest – skripta koja mijenja vidljivost kovčega koji se pojavljuje na kraju igre
- Door – skripta koja provjerava da li je igrač pokupio ključ te javlja game manager-u kada treba pokrenuti novi nivo
- DoorKey – javlja game manager-u kada je igrač pokupio ključ
- EnemyEye – skripta koja kontrolira kretanje protivnika
- EnemyEyeBoss – skripta koja kontrolira kretanje finalnog protivnika, njegov napad te ponašanje nakon napada
- EnemyEyeBossBot – skripta koja postavlja zdravlje igrača na nulu, kada collider igrača uđe u collider protivnika
- EnemyEyeBossTop – skripta smanjuje zdravlje protivnika za jedan, svaki put kada collider igrača uđe u collider protivnika
- GameManager – skripta koja pamti rezultate, da li je nivo otključan, te se brine za pokretanje odgovarajućeg nivoa
- Audio Manager – skripta koja je zaslužna za zvuk u igri
- Sound – skripta koja sadrži podatke za zvukove
- Loader – skripta koja stvara instancu game manager-a
- MainMenu – skripta zaslužna za izbornik na početku igre
- Menu – skripta za izbornik unutar nivoa i za izbornik koji se pojavljuje nakon smrti
- MenuLevelSelector – skripta za izbornik odabira nivoa
- Player – skripta zaslužna sve operacije vezane za igrača
- Spikes – skripta koja postavlja zdravlje igrača na nulu kada collider igrača uđe u collider bodlji
- Story – skripta služi za preskakanje priče
- StoryText – skripta zaslužna za prikazivanje teksta priče na sceni za priču

- Turret – skripta za pucanje tornja, ujedno provjerava da li je igrač unutar collider-a koji predstavlja maksimalnu distancu pucanja te provjerava da li toranj vidi igrača
- TurretBullet – skripta koja određuje smjer ispaljenog projektila te provjerava da li je projektil pogodio igrača

U nastavku će biti objašnjenje funkcije u skriptama koje ostvaruju glavne mehanike igre. Slijede funkcije iz skripte Player.

```
protected void HandleMovement()
{
        float horizontal = Input.GetAxisRaw("Horizontal");
        rb2d.velocity = new Vector2(horizontal * Speed, rb2d.velocity.y);
}
```

Slika 27: Funkcija HandleMovement

Na slici 27 je prikazana funkcija „HandleMovement“ koja je zaslužna za kretanje igrača. Poziva se u funkciji „Update“ (funkcija se poziva svaki frame). U varijablu „horizontal“, tipa float, se sprema vrijednost „Input.GetAxisRaw(„Horizontal“)“, tj. sprema se vrijednost od -1 do 1, ovisno o tome koja je tipka na tipkovnici pritisnuta. Nakon toga varijabli „rb2d“ (varijabla tipa „RigidBody 2d“) se postavlja nova vrijednost za „velocity“, u obliku vektora. Novi vektor mijenja vrijednost x osi na dohvaćenu vrijednost puta „Speed“ (zadana vrijednost koja predstavlja brzinu igrača) te kopira vrijednost y osi.

```
protected void HandleJumping()
{
    if (Input.GetButtonDown("Jump") && !IsSliding)
    {
        if (IsGrounded)
        {
            rb2d.velocity = new Vector2(rb2d.velocity.x, JumpPower);
            canDoubleJump = true;
        }
        else
        {
            if (canDoubleJump)
            {
                canDoubleJump = false;
                rb2d.velocity = new Vector2(rb2d.velocity.x, 0);
                rb2d.velocity = new Vector2(rb2d.velocity.x, JumpPower * SecondJumpPower);
            }
        }
    }
}
```

Slika 28. Funkcija HandleJumping

Na slici 28 je prikazana funkcija HandleJumping koja je zaslužna za skakanje igrača. Na početku funkcija se provjerava da li je korisnik pritisnuo tipku koja je zaslužna za skakanje te da li je varijabla „IsSliding“ lažna (varijabla govori da li igrač klizi po zidu). Ako je uvjet zadovoljen, slijedi nova provjera u kojoj se ispituje da li je varijabla „IsGrounded“ istinita (varijabla govorio da li se igrač nalazi na tlu). Ako je varijabla istinita slijedi kod koji postavlja vrijednost za „velocity“ varijable „rb2d“ na novi vektor. X os ostaje ista dok y os postaje „JumpPower“ (prethodno zadana varijabla koja određuje jačinu skoka). Također se varijabla „canDoubleJump“ postavlja na „true“. Ako „IsGrounded“ nije istinita, slijedi novi dio koda koji se izvršava samo ako je varijabla „canDoubleJump“ istinita. Varijabla govori da li igrač može napraviti još jedan skok u zraku. Ako je provjera zadovoljena slijedi kod koji postavlja varijablu „canDoubleJump“ na „false“. Nakon toga postavlja „velocity“ y ois, od varijable „rb2d“ na 0 te nakon toga postavlja na „JumpPower“ * „SecondJumpPower“.

```
void HandleWallSliding()
{
    rb2d.velocity = new Vector2(rb2d.velocity.x, -0.2f);

    IsSliding = true;

    if (Input.GetButtonDown("Jump") && Input.GetAxisRaw("Horizontal") == 0)
    {
        rb2d.velocity = new Vector2(-wallDir * wallJump.x, wallJump.y);
        transform.localScale = new Vector3(-wallDir, 1, 1);
        FacingRight = !FacingRight;
    }
}
```

Slika 29. Funkcija HandleWallSliding

Na slici 29 je prikazana funkcija HandleWallSliding koja je zaslužna za klizanje po zidu te skakanje sa zida. Funkcija se poziva samo kada je varijabla „isWall“ istinita (varijabla govori da li igrač dodiruje zid) te varijabla „IsGrounded“ lažna. Na početku se „velocity“ y osi za varijablu „rb2d“ postavlja na -0.2. Varijabla „IsSliding“ se postavlja na „true“. Nakon toga se provjerava da li je korisnik pritisnuo tipku za skok te da li je „Input.GetAxisRaw(„Horizontal“)“ jednaka nuli. Ako je provjera zadovoljena slijedi kod koji postavlja novi vektor za „velocity“. X os poprima vrijednost –wallDir * wallJump.x. Varijabla „wallDir“ predstavlja orijentaciju lika (da li lik gleda za desno ili za lijev), a varijabla „wallJump.x“ predstavlja jačinu skoka za x os. Y os novog vektora poprima

vrijednost „wallJump.y“, varijabla predstavlja jačinu skoka po y osi. Varijabla „transform.localScale“ poprima novi vektor koji mijenja orijentaciju lika. Na kraju varijabla „FacingRIght“ dobiva vrijednost „!FacingRight“, tj. varijabla postaje lažna ako je bila istinita i obrnuto.

```
void OnTriggerEnter2D(Collider2D col)
{
    if (col.CompareTag("Player"))
    {
        player.GetComponent<Player>().HandleDeath();
    }
}
```

Slika 30. Funkcija OnTriggerEnter2D za Spike

Na slici 30 je prikazan primjer funkcije za collider-e. U ovom slučaju se funkcija okida kada collider uđe u drugi collider. Postoje varijacije gdje se provjerava ostanak te izlazak. Funkcija sa slike uspoređuje tag collider-a koji je ušao, u ovom slučaju provjerava da li je to collider s tag-om „Player“, te poziva odgovarajuću funkciju. Funkcija pripada objektu „Spike“. Isti princip se koristi kod objekata „EnemyEye“ te „EnemyEyeBoss“. Princip collider-a te njihov izgled će biti detaljnije pojašnjen u nastavku.

```
void OnTriggerEnter2D(Collider2D col)
{
    if (col.CompareTag("Player"))
    {
        gm.isUnlocked = true;
        Destroy(gameObject);
    }
}
```

Slika 31. Funkcija OnTriggerEnter2D za DoorKey

Na slici 31 je prikazana slična funkcija onoj prethodnoj. Funkcija pripada objektu „DoorKey“. Ako je provjera tag-a valjana, funkcija javlja game manager-u da je ključ pokupljen, te uništava objekt ključ.

```
void RangeCheck()
{
    Debug.DrawLine(CastStart.position, target.position, Color.blue);

    bool test = Physics2D.Linecast(CastStart.position, target.position, 1 << LayerMask.NameToLayer("Wall"));

    if (!test && isAwake)
        Attack();
}
```

Slika 32. Funkcija RangeCheck

Na slici 32 je prikazana funkcija RangeCheck koja provjerava da li je igrač vidljiv. Funkcija pripada objektu „Turret“. Funkcija koristi ray casting kako bi provjerila vidljivost igrača. Ray cast sadrži filter, u ovom slučaju je to sloj s imenom „Wall“, ako linija ,koja se stvori putem ray casting-a, ispuni uvjet filtera tada vraća vrijednost „true“. Ray casting će biti detaljnije objašnjen u nastavku. Slijedi provjera u kojoj se ispituje vrijednost varijable „test“ te varijable „isAwake“ (varijabla koja provjerava da li je igrač u dometu objekta turret, provjera se vrši putem collidera). Ako je uvjet zadovoljen poziva se funkcija „Attack“.

```
public void Attack()
{
    bulletTimer += Time.deltaTime;

    if (bulletTimer >= shootInterval)
    {
        Vector2 direction = target.transform.position - transform.position;
        direction.Normalize();

        GameObject bulletClone;
        bulletClone = Instantiate(bullet, shootPoint.transform.position, shootPoint.transform.rotation) as GameObject;
        bulletClone.GetComponent<Rigidbody2D>().velocity = direction * bulletSpeed;

        bulletTimer = 0;
    }
}
```

Slika 33. Funkcija Attack

Na slici 33 je prikazana funkcija „Attack“ koja zaslužna za ispaljivanje projektila iz objekta „turret“. Na početku funkcije se povećava varijablu „bulletTimer“ (predstavlja vrijeme od zadnjeg ispaljenog projektila). Slijedi provjera koja provjerava da li je „bulletTimer“ veći od „shootInterval“ (interval za ispaljivanje projektila). Ako je uvjet zadovoljen u varijablu „directon“ se sprema smjer do igrača od objekta „turret“. Nakon toga se deklarira varijable „bulletClone“, te se u istu varijablu instancira novi objekt „bullet“. Objekt „bullet“ predstavlja projektil. Prilikom instanciranja se zadaje rotacija objekta. Nakon toga se mijenja njegova brzina te se varijabla „bulletTimer“ postavlja na nulu.

Slijedi funkcija za zvuk. Na slici 34 je prikazana funkcija za zvuk u igri.

```
public void Play(string sound)
{
    Sound s = Array.Find(sounds, item => item.name == sound);
    if (s == null)
    {
        Debug.LogWarning("Sound: " + name + " not found!");
        return;
    }

    s.source.volume = s.volume * (1f + UnityEngine.Random.Range(-s.volumeVariance / 2f, s.volumeVariance / 2f));
    s.source.pitch = s.pitch * (1f + UnityEngine.Random.Range(-s.pitchVariance / 2f, s.pitchVariance / 2f));

    s.source.Play();
}
```

Slika 34. Funkcija Play

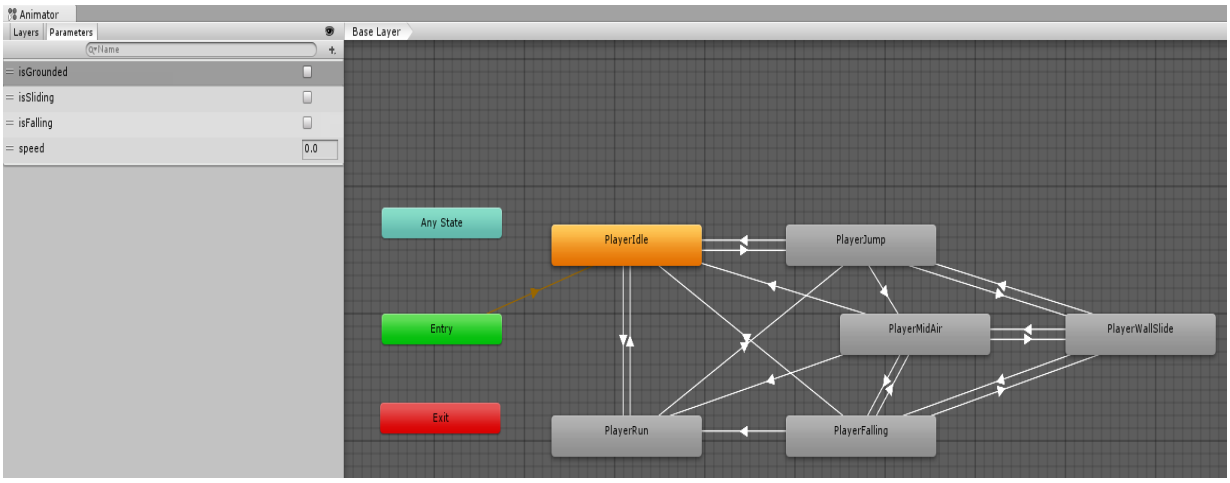
Funkcija pripada objektu „Audio Manager“. Funkcija kao argument dobiva ime zvuka kojeg treba pokrenuti. Nakon toga ga traži u polju koje je tipa „Sound“, te ga pokreče nakon što ga pronađe.

8 Dodatne značajke

U ovom poglavlju će biti objašnjene neke dodatne značajke, one su: „animator“, „collider“ te „ray casting“.

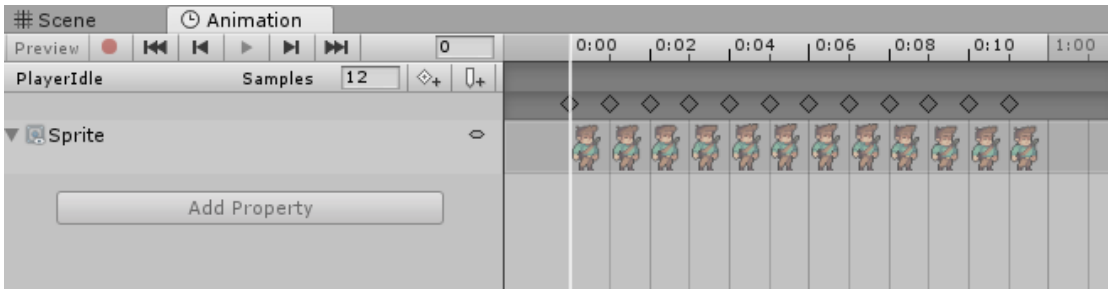
8.1 Animator

Animator je dio Unity-a koji je zaslužan za animacije objekta.



Slika 35. Izgled animatora

Na slici 35 je prikazan izgled animatora. U ovom slučaju je to animator za igrača. Animator se sastoji od stanja (state). Stanja su prikazana kao oblačići. Narančasto stanje predstavlja početno stanje. Stanja mogu biti povezana sa strelicama, naziv za strelice je prijelaz (transition). Animator sadrži i parametre. Pod oznakom „a“ su vidljivi parametri animatora. Parametri mogu biti numerički, bool ili okidači (trigger). Prijelazi koriste parametre kako bi znali kada trebaju prijeći u drugo stanje. Svako stanje ima pridruženu animaciju.



Slika 36. Izgled animacije

8.2 Collider

„Collider“ je komponenta koja se pridružuje objektu. Omogućuje objektu interakciju s drugim objektima.

Slika 37. Izgled collider-a

Na slici 37 je prikazan izgled „collider-a“. Objekt igrača sadrži „collider“, te objekt koji predstavlja teren sadrži „collider“. Kada se scena pokrene te fizika počne djelovati na igrača, igrač će padati dok se njegov „collider“ i „collider“ terena ne dodirnu. Da igrač ili teren nemaju „collider“, igrač bi samo propao kroz teren.

„Collider“ može biti i okidač. Na slici 35 ni jedan „collider“ nije okidač. Ako je „collider“ okidač, onda se mogu vršiti provjere kao kod npr. objekta „Spike“. Provjere koje se mogu vršiti su: da li je neki drugi collider ušao u collider objekta, da li je izašao te da li je trenutno u njemu.

Slika 38. Interakcija između collider-a

Na slici 38 je prikazana interakcija između dva „collider-a“. Jedan „collider“, od igrača, nije okidač dok je drugi „collider“, od objekta „Spike“, okidač. Vrši se provjera na ulasku. Ako igračev „collider“ uđe unutar „collider-a“ objekta „Spike“, tada se zdravlje igrača postavlja na nulu.

8.3 Ray casting

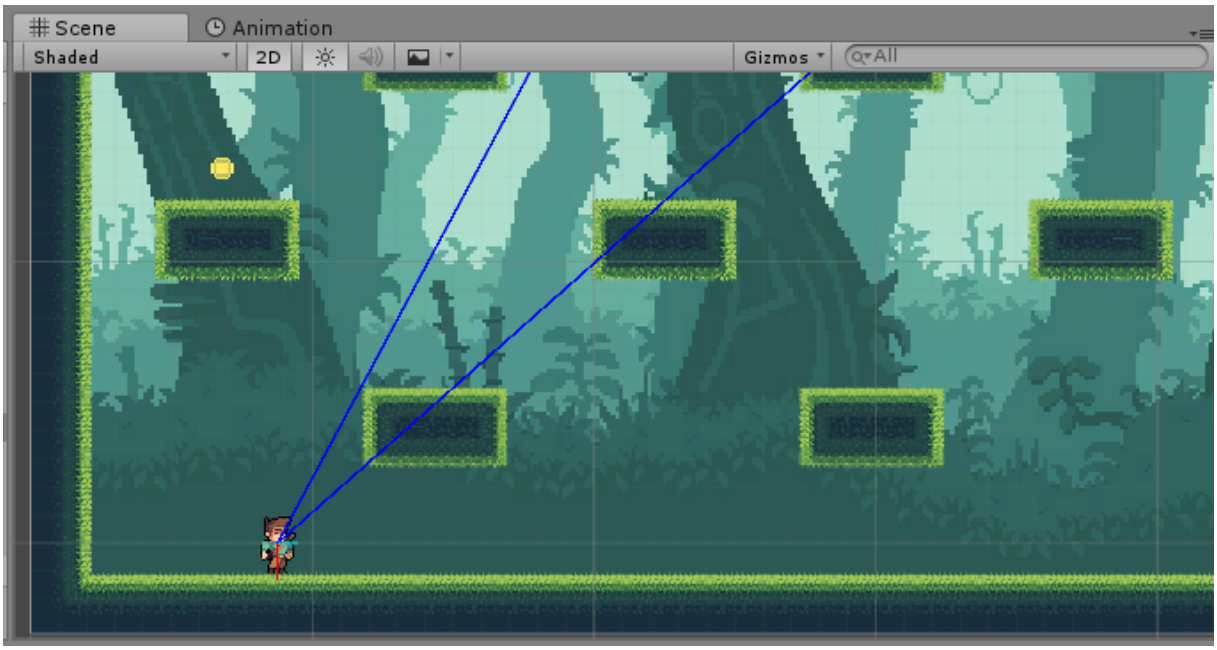
„Ray casting“ je tehnika u kojoj se povlači imaginarna linija od jedno do druge točke. Prilikom stvaranja linije moguće je zadati određene filtere. Npr. filter koji će govoriti da li je linija prošla kroz zid.

```
Debug.DrawLine(CastStart.position, target.position, Color.blue);

bool test = Physics2D.Linecast(CastStart.position, target.position, 1 << LayerMask.NameToLayer("Wall"));
```

Slika 39. Kod za ray casting

Na slici 39 je prikazan kod za ray casting. Prva linija kod unutar editora pokazuje liniju koja nastaje „ray casting-om“, a druga linija predstavlja „ray casting“ s uključenim filtrom za zidove.

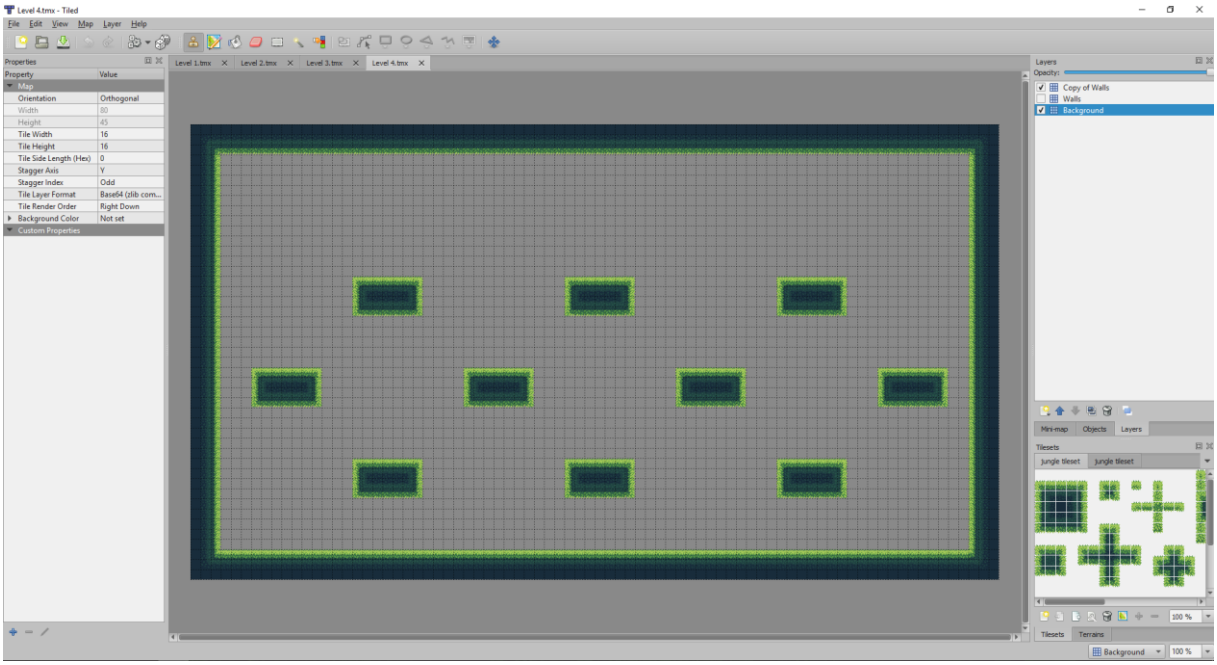


Slika 40. Ray casting u editoru

Na slici 40 je prikazan „ray casting“. Pomoću „debug-era“ je dodana boja liniji koja nastaje „ray casting-om“.

9 Alata za izradu nivoa

Alat koji je korišten za izradu terena na nivoima je „Tiled“. Tiled je besplatni alat, otvorenog koda, koji služi za izradu terena nivoa za Unity. Plugin „Tiled2Unity“ se koristi kako bi datoteka kreirana pomoću Tiled, pretvorila u kompatibilnu datoteku za Unity.



Slika 41. Izgled Tiled alata

Autor grafike za teren, igrača, protivnika te ostalih stvari je Jesse M.

10 Zaključak

Igre predstavljaju jedan od mnogih načina putem kojih se čovjek umjetnički izražava.

U današnjoj industriji razvijana igara postoji velika konkurencija te je jako teško napraviti kvalitetan i popularan proizvod. Tržište je preplavljeno igrama i pokušajima igara što vodi do toga da neke kvalitetne igre nikad ne isplivaju na površinu.

Uz sve navedene probleme, te ogromnu konkurenciju koju predstavlja veliki broj AAA tvrtki, nezavisni razvijači imaju težak put pred sobom ako žele uspjeti na tržištu igara.

Odabir okruženja tj. pokretača igre također predstavlja problem. Dvije su mogućnosti, prva mogućnost je izrada vlastitog pokretača dok je druga mogućnost korištenje već gotovih pokretača na tržištu.

Neka od besplatnih okruženja koja spadaju pod drugu mogućnost su Unity, Cocos2d te Unreal Engine. Svako, od navedenih okruženja, ima svoje određene prednosti i mane pomoću kojih razvijač odlučuje koje okruženje najbolje odgovara potrebama njegove igre.

Unity okruženje je korišteno u izradi ovog projekta. Unity je besplatno okruženje koje korisnicima nudi pregršt mogućnosti za razvoj svoje igre. Za razliku od ostalih navedenih besplatnih okruženja, Unity pruža jednostavnost koja je privlačna velikom broju početnika koji žele započeti svoj put u industriji razvoja igara.

Ako razvijač želi da njegova igra konkurira današnjim velikanima te dostigne neki industrijski standard, razvijač mora uložiti veliku količinu vremena i truda kako bi to ostvario. Alternativu uloženim resursima predstavlja veći razvojni tim, ali i to dolazi s drugim problemima. Također su tu i prepreke izdavanja igara. Dok iza AAA razvijača stoje ogromni izdavači koji troše milijune na marketing, nezavisni razvijači nemaju takav luksuz na raspolaganju.

Iz svega navedenog se da zaključiti da je proces razvoja igara kompleksan i dug proces koji zahtjeva od razvijača da uloži veliku količinu truda i vremena kako bih mogao konkurirati velikanima u današnjoj industriji razvoja igara.

Literatura

Chris Baker (2010) www.wired.com/2010/06/replay/, datum pristupa 2.9.2017.

David M. Ewalt (2013) Of Dice and Men: The Story of Dungeons & Dragons and The People Who Play It

Henry E. Lowood (2004) www.britannica.com/topic/Zork-1688286#ref796387, datum pristupa 2.9.2017.

Jesse M (2017) jesse-m.itch.io/jungle-pack, datum pristupa 2.9.2017.

Mark J. P. Wolf (2012) Before the Crash: Early Video Game History

Mark Wilcox (2014) https://www.developereconomics.com/top-game-development-tools-pros-cons, datum pristupa 2.9.2017.

Raffaele Pisaon (2015) A Bridge between Conceptual Frameworks: Sciences, Society and Technology Studies

Rich Stanton (2015) A Brief History Of Video Games: From Atari to Xbox One

Steven L. Kent (2001) The Ultimate History of Video Games

Tristan Donovan (2010) Replay: The History of Video Games

Van Burnham (2001) Supercade: A Visual History of the Videogame Age 1971-1984

Popis slika

Sažetak

Cilj ovog rada je bio prikaz procesa izrade igre od samih početaka do kraja razvoja. Opis svih komponenata od kojih se igra sastoji te programskog koda koji stoji iza svega toga.

Spomenuta je povijest razvoja igara te žanrovi koji danas postoje.

Opisani su najkorišteniji alati tj. okruženja koja se koriste u današnjoj industriji, te su navedene prednosti i mane pojedinih alata. Detaljnije je opisan odabrani, Unity, alat.

Također je opisana struktura rada te svi objekti i komponente koje se nalaze u samom radu.

Kroz rad se pokazalo da je proces razvoja igre kompleksan i dug te zahtjeva veliku količinu resursa od strane razvojnog programera odnosno razvojnog tima.

Ključne riječi: okruženje, alat, Unity, pokretač, scene, skripte, komponente, objekti

Abstract

The purpose of this thesis was to show the process of making a game, from the beginning to the end of the development. To portray the game components and the code that lies behind them.

The history of game development and game genres has been touched upon.

The thesis mentions tools that are being used in today's game development industry, and it touches upon the pros and cons of said tools. Unity has been covered in more detail.

The structure of the project and all containing components and objects has been mentiond as well.

The thesis has shown that the process of developing a game is complex and demands a lot of resources from the developer and the developer team, respectively.

Key words: framework, tool, Unity, engine, scene, scripts, components, objects