Izrada 2.5D računalne igre u Unity razvojnom okruženju

Miler, Romana

Sveučilište
Sjever

Završni rad br. 568/MM/2018

Izrada 2.5D računalne igre u Unity razvojnom okruženju

Romana Miler, 0799/336

Varaždin, rujan 2018. godine

Sažetak

U ovom radu prikazan je proces izrade jednostavne 2.5D side-scrolling računalne igre te su objašnjeni korišteni programi i njihovi alati. Programe koje sam koristila tijekom izrade igre su Autodesk Maya i Unity.

Izrada igre ne mora nužno biti kompliciran proces, ovisno o tome koliko veliku i zahtjevnu igru želimo napraviti. Igre najčešće razvijaju manji timovi ako je riječ o nekim manje zahtjevnim igrama (npr. mobilne aplikacije) ili veliki timovi koji rade na većim projektima za velike izdavače igara. Takvi timovi imaju dizajnere koji stvaraju vizualni dio igre, ideju, priču, pravila, itd., te programere koji su zaduženi za pisanje kodova i skripti i koji će dovesti u funkciju sve što su dizajneri zamislili. Stoga, ne možemo očekivati da ćemo samostalno moći napraviti nešto na razini nečega na čemu radi tim ljudi.

Kada je riječ o 2.5D side-scrolling igri, zapravo se radi o modernoj verziji 2D igara koje se još zovu i platforme. U slučaju 2.5D side-scrolling igara uglavnom se radi o igrama čiji su likovi i prostor trodimenzionalni, ali je projekcija kamere dvodimenzionalna, odnosno ima fiksni kut. Ovaj žanr igre je najveću popularnost doživio osamdesetih godina dvadesetog stoljeća, nakon čega su devedesetih došle konzole koje su podržavale trodimenzionalni sustav igranja i to je bacilo platforme u zaborav.

Zadnjih godina su se pojavile 2.5D side-scrolling igre koje su doživjele poprilično veliku popularnost, a i nostalgija za starim platformama sve je veća, tako da se ponovno javljaju stare platforme koje su sada kompatibilne s današnjim konzolama i računalima.

Ključne riječi: 2.5D side-scrolling, Autodesk Maya, Igra, Skripte, UI, Unity

Abstract

This thesis presents the process of making a simple 2.5D side-scrolling computer game and explains the used programs and their tools. The programs that I used during the game development are Autodesk Maya and Unity.

Creating a game does not necessarily have to be a complicated process, it depends on how big and demanding game we want to make. Some less demanding games (for example, mobile apps) are usually developed by smaller teams, and on the other hand, larger teams work on major projects for major game publishers. Such teams have designers who create the visual part of the game, idea, story, rules, etc., and programmers who are in charge of writing codes and scripts and they should make everything work the way designers have imagined. Therefore, we cannot expect ourselves to be able to do something on the level of something that is developed by the whole team.

When it comes to the 2.5D side-scrolling game, it's actually a modern version of the 2D game called the platform. In the case of 2.5D side-scrolling games, these are mostly games with three-dimensional characters and scene, but the projection of the camera is two-dimensional, i.e. has a fixed angle. This genre of gameplay was the most popular in the 1980s, and in the nineties came consoles that supported the three-dimensional gaming system and dropped the platform into oblivion.

In the past few years, 2.5D side-scrolling games gained quite a big popularity, and nostalgia for older platforms is getting bigger, so old platforms are now coming back and they are compatible with today's consoles and computers.

Keywords: 2.5D side-scrolling, Autodesk Maya, Game, Scripts, UI, Unity

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za multimediju, oblikovanje i primjenu

PRISTUPNIK: Romana Miler | MATIČNI BROJ: 0799/336

DATUM: 2.3.2018. | KOLEGIJ: 3D modeliranje

NASLOV RADA: Izrada 2.5D računalne igre u Unity razvojnom okruženju

NASLOV RADA NA ENGL. JEZIKU: Creating a 2.5D computer game in Unity engine

MENTOR: dr.sc. Andrija Bernik | ZVANJE: Predavač

ČLANOVI POVJERENSTVA:
1. mr.sc. Dragan Matković, v. pred. - predsjednik
2. doc.art. Robert Geček - član
3. dr.sc. Andrija Bernik, pred. - mentor
4. Nikolina Bolčević Horvatić, asistent - zamjenski član
5.

Zadatak završnog rada

BROJ: 568/MM/2018

OPIS

Cilj ovog završnog rada je izraditi PC igru korištenjem programa Autodesk Maya i Unity. U radu se upoznaje s poviješću 2.5D računalnih igara i osnovnim karakteristikama i mogućnostima navedenih programa.

U praktičnom dijelu će se izraditi PC igra uz pomoć spomenutih programa. U Autodesk Mayi će se izraditi, texturirati i animirati svi 3D modeli. U Unity se ubacuju modeli napravljeni u Mayi te će tamo postati dinamični, u smislu kretanja kroz prostor, te će se na taj način stvoriti igra. Igra će se sastojati od jednog levela u kojem će glavni lik morati prelaziti razne prepreke, sukobljavati se s neprijateljima i doći do odre enog mjesta kako bi nivo završio.

U radu je potrebno:
- opisati program Autodesk Maya
- opisati program Unity
- opisati korištene alate oba programa
- opisati izradu tekstura i animacije 3D modela
- opisati postupak izrade igre u Unity-u

REPUBLIKA HRVATSKA 5 SVEUČILIŠTE SJEVER

ZADATAK URUČEN: 04.04.2018. | POTPIS MENTORA: Bernik

Sveučilište
Sjever

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Romana Miler *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom Izrada 2.5D računalne igre u Unity razvojnom okruženju *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Romana Miler
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, Romana Miler *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom Izrada 2.5D računalne igre u Unity razvojnom okruženju *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Romana Miler
(vlastoručni potpis)

Sveučilište
Sjever

Odjel za Multimediju, oblikovanje i primjenu

Završni rad br. 568/MM/2018

Izrada 2.5D računalne igre u Unity razvojnom okruženju

Student

Romana Miler, 0799/336

Mentor

dr.sc. Andrija Bernik, pred.

Varaždin, rujan 2018. godine

Predgovor

Temu za ovaj završni rad izabrala sam zbog velikog zanimanja za svijet 3D-a i igara općenito. Cilj mi je bio steći nova znanja, odnosno proširiti znanje u programu Autodesk Maya koje sam do sad stekla na fakultetu te steći jedno potpuno novo radom u programu Unity. U tome mi je uvelike pomoglo i čitanje knjiga i stručnih radova iz područja 3D modeliranja i vezanih uz oba programa.

Zahvaljujem mentoru dr.sc. Andriji Berniku, pred., koji me je usmjeravao tijekom pisanja ovog završnog rada i svojim savjetima mi pomagao u rješavanju problema. Također, zahvaljujem roditeljima i prijateljima na pruženoj podršci tijekom studiranja.

Popis korištenih kratica

2D	Two-dimensional Dvodimenzionalno
3D	Three-dimensional Trodimenzionalno
2.5D	Two-and-a-half-dimensional Dvaipoldimenzionalno/Dvije i pol dimenzije
AOV	Arbitrary Output Variables Proizvoljno izlazne varijable
UI	User interface Korisničko sučelje
GUI	Graphical user interface Grafičko korisničko sučelje
NURBS	Non-Uniform Rational B-Splines
MEL	Maya Emmbeded Language
IK	Inverse kinematics Inverzna kinematika
FPS	Frame per second Okvir po sekundi
SF	Science fiction Znanstvena fantastika

Sadržaj

1. Uvod

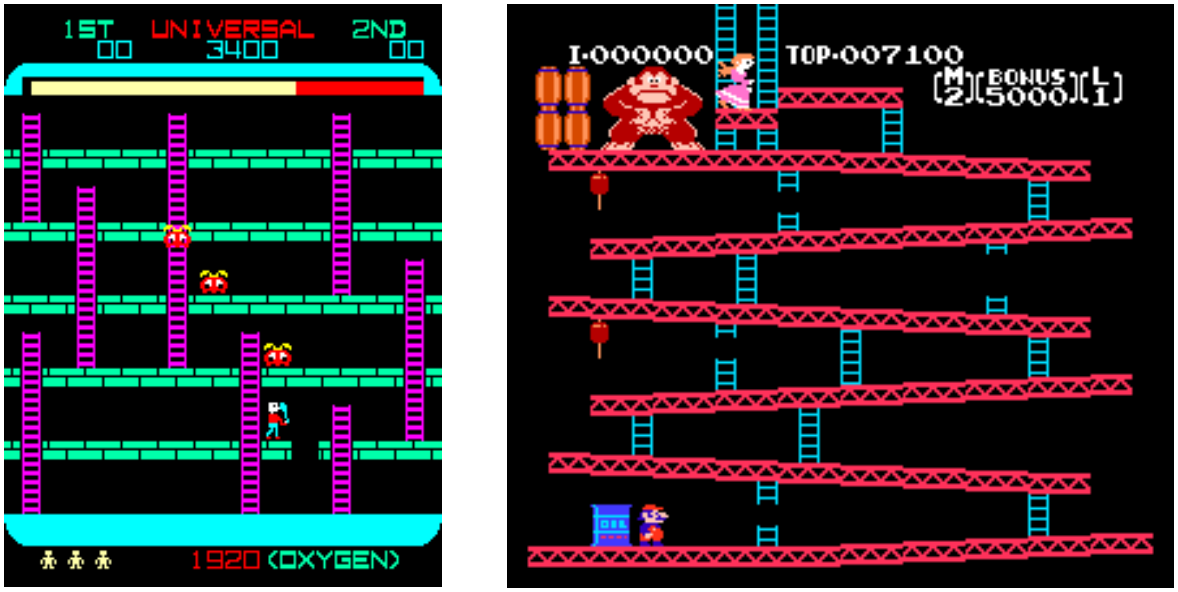
Cilj ovog rada je napraviti 2.5D side-scrolling igru koristeći programe Autodesk Maya i Unity. Kroz rad na igri objašnjavaju se alati i mogućnosti oba programa te principi modeliranja, teksturiranja, izrade UV mapa, animiranja, programiranja/skriptiranja, itd. U teorijskom dijelu rada upoznaje se s nastajanjem 2.5D side-scrolling igara te se opisuju osnovne značajke, mogućnosti, alati i korisnička sučelja programa Maya i Unity. Navedeni programi su jedni od najjačih softvera za razvoj 3D okruženja i igara u svijetu. Maya omogućuje poprilično jednostavnu izradu modela te modifikaciju istih, kao i teksturiranje, animiranje i još mnogo toga. Ima vrlo široku upotrebu, pogotovo u filmskoj i gaming industriji. Možda smo gledali neke poznate filmove ili animirane filmove ili igrali neke video igre velikih izdavača, a nismo bili svjesni koji softver stoji iza svih tih efekata, animacija, modela, itd. Tako se Maya može pohvaliti sudjelovanjem u izradi SF klasika Matrix, kao i u izradi igara franšize Crash Bandicoot te Grand Theft Auto III. S druge strane, Unity se može pohvaliti svojim jednostavnim korisničkim sučeljem koje olakšava snalaženje u programu novim korisnicima, kao i s mogućnosti izrade 2D, 3D ili u ovom slučaju, 2.5D igara čak i ako igru izrađujemo samostalno. U njemu su se izradili jedni od najvećih naslova gaming svijeta. Tako Unity iza sebe ima naslove poput Deus Ex: The Fall, Assassin's Creed: Identity, Rust, Ori and the Blind Forest, Inside, itd.

U praktičnom dijelu rada izrađuje se igra pomoću navedenih programa. U Mayi se modelira, teksturira i animira većina 3D modela i na taj način se dobije željeni izgled scene u igri. Rad se nastavlja dalje u programu Unity u kojem svi ti modeli dolaze u život i dobivaju smisao kao jedna cjelina. Da bi igra kao funkcionalna aplikacija uopće mogla postojati, objekti ubačeni u Unity zahtijevaju skripte vezane uz njih u kojima je isprogramiran način na koji se kreću kroz prostor ili dobiju neku svoju ulogu u igri. Prilikom programiranja koristi se programski jezik C# koji je danas najpopularniji kod programera. U igru je ubačen pozadinski zvuk te popratni zvukovi kao efekti na određene radnje ili objekte kako bi igra izgledala što dinamičnije i zanimljivije. Igra je sastavljena od glavnog izbornika (engl. main menu) i jednog nivoa (engl. levela) u koji se ulazi kada u glavnom izborniku stisnemo na gumb Play. Preostala dva gumba u glavnom izborniku su Hints (naputci za igranje) i Quit (izlazak iz igre). Cilj igre je pobijediti neprijatelje i prijeći prepreke kako bi skupili sva zmajeva jaja i pobijedili. Ako se u igri pobijedi ili izgubi, igra se ponovno resetira na početak.

2. Povijest i razvoj 2.5D side-scrolling igara

Platformske video igre su uglavnom 2D igre u kojima se sve većinom odvija oko glavnog lika s kojim igrač upravlja, odnosno trči i skače po platformama, podlogama, podovima, stepenicama ili nekim drugim objektima, ovisno je li riječ o horizontalnom ili vertikalnom kretanju lika. No, kada je riječ o 2.5D platformskim igrama, razlika je jedino u grafici, tj. perspektiva je ili 2D grafička projekcija koja koristi razne tehnike koje simuliraju da slika ili prizor izgledaju trodimenzionalno ili je igra zapravo trodimenzionalna i ograničena je na dvodimenzionalnu ravninu ili ima virtualnu kameru s fiksnim kutom. [1]

Prve platformske video igre razvijene su početkom osamdesetih godina što platformsku igru čini jednim od prvih žanrova video igara. Pojam platforma ili platformer krenuo se upotrebljavati tek nekoliko godina kasnije za opisivanje te vrste igre. Platformske igre su se jako brzo popularizirale te postale dostupne širokoj javnosti. Mnogi smatraju da je Space Panic (izdana 1980.) prva prava platformska video igra, dok neki smatraju da je prva platformska igra zapravo Nintendov Donkey Kong (izdana 1981.). U svakom slučaju, upravo su ti rani klasici poput Donkey Konga, Space Panica i Mario Brosa imali velik utjecaj u oblikovanju žanra platformske igre. [1]



Slike 2.1. i 2.2. Space Panic i Donkey Kong [1][2]

Stvaranjem trodimenzionalnih sustava za igranje kao što su Nintendo-64 i Sony Playstation došlo je do propadanja dvodimenzionalnog razdoblja. S novim 3D svijetom bilo je teško očekivati razvoj side-scrolling igre, tako da su platformske igre krenule padati u zaborav. Nekoliko naslova pokušalo je kombinirati 2D i 3D svjetove, stvarajući svijet 2.5 dimenzije. Takve igre su prvenstveno bile platformske, ali neprijatelji i neki određeni dijelovi igre igrali su se u 3D okruženju. No te igre nisu bile iznimno uspješne i nisu privlačile preveliku pozornost. Jedne od prvih 2.5D igara bile su Pandemonium!, Einhänder i Klonoa: Door to Phantomile. [2]

Slike 2.3. i 2.4. Klonoa: Door to Phantomile i Pandemonium! [3][4]

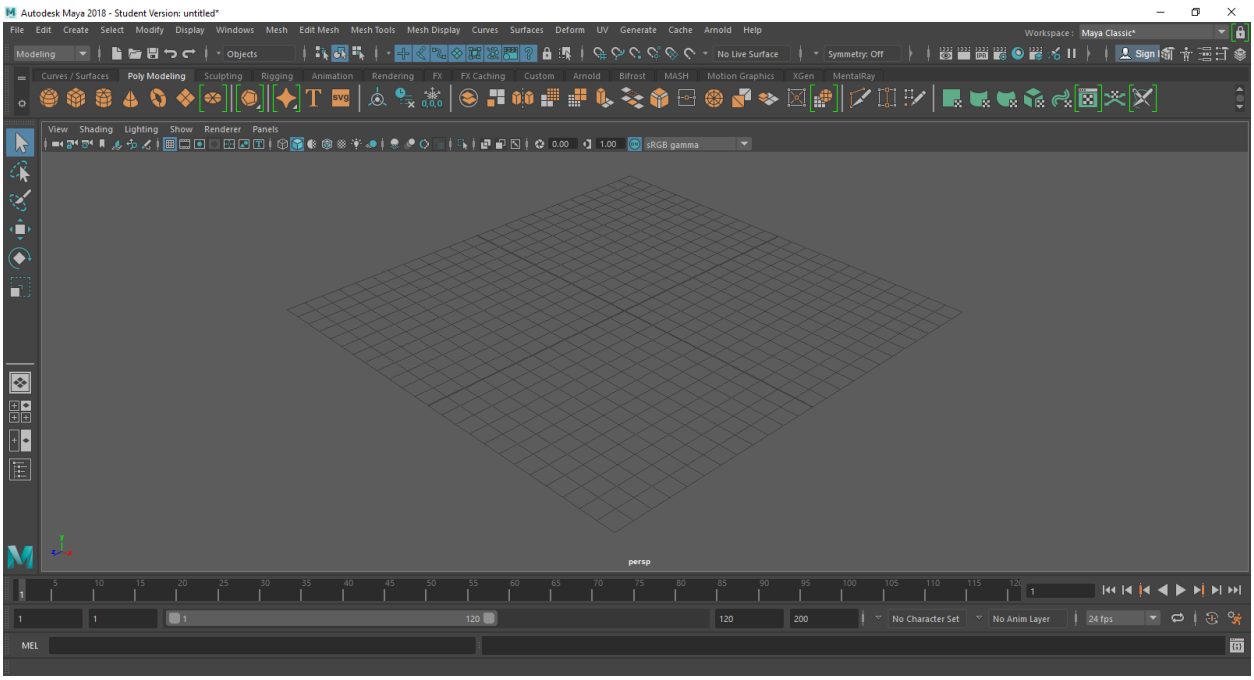
Danas se ponovno pojavljuju stari 2D side-scrolleri. Sustavi kao što su Nintendo Wii omogućuju svojim korisnicima da ponovno igraju stare igre tako da ih preuzmu sa svojih starih Nintendo konzola. Također, sve je više 2.5D igara koje su iznimno popularne, poput Unravel, Ori and the Blind Forest, Little Nightmares, itd. [2]


Slike 2.5. i 2.6. Ori and the Blind Forest i Unravel [5][6]

3. Autodesk Maya

Autodesk Maya (skraćeno Maya) je softver koji nudi skup kreativnih značajki za 3D računalnu animaciju, modeliranje, simulaciju, renderiranje, teksturiranje. Izvorno je razvijen od strane Alias Systems Corporationa (skraćeno Alias) 1998., a trenutno je u vlasništvu Autodeska (od 2005.). Maya se najviše koristi za stvaranje interaktivnih 3D aplikacija, uključujući video igre, animirani film, TV serije i vizualne efekte. Njenu široku primjenu možemo vidjeti prema činjenici da je korištena prilikom izrade vrlo poznatih filmova i animiranih filmova kao što su The Matrix, Shrek, Finding Nemo, Harry Potter and The Deathly Hollows: Part 2, animiranoj seriji South Park u kojoj se koristi i dan danas te u igrama Crash Bandicoot (trilogija i Crash Team Racing), Grand Theft Auto III, itd.

Iako je korisničko sučelje Maye tijekom godina nadograđivano, ono se nije previše mijenjalo. Aliasov tim je odmah na početku kreirao sučelje tako da bude pregledno i lako razumljivo novim korisnicima. Osim obične Maye, postoji i Maya LT. Maya LT je alat za 3D modeliranje i animaciju izrađen posebno za kreatore igara koji nudi snažan set kreativnih alata i workflowa za stvaranje objekata za igru za mali dio Mayine pune cijene. Najnovije izdanje, Maya 2018, uključuje velika poboljšanja workflowa i nove značajke čiji je cilj stvaranje sadržaja lakše nego ikad prije. Neke od značajki su novi UV Editor koji nudi snažne alate za stvaranje UV mapa s lakoćom. Također, ažuriran je Viewport 2.0 koji je kompatibilan s OSD 3.1, složenim mrežama za sjenčanje, svjetlosnim sjenama i okluzijom okoline što omogućava bolji prikaz izgleda renderiranih slika. Arnold 5 je uključen u instalaciju Maye i sadrži neke nove shadere te ima podršku za svjetlosnu grupu AOV i poboljšanu podršku za upravljanje bojama. Novi MASH Dynamics omogućuje primjenu dinamičkih svojstava na mreže (engl. mesh), koje se spajaju s postojećim čvorovima (engl. nodes) da bi se stvorila vizualno zapanjujuća pokretna grafika sa samo nekoliko klikova. No to su samo neke od odličnih promjena koje je dobila Maya 2018. [3][10][11]

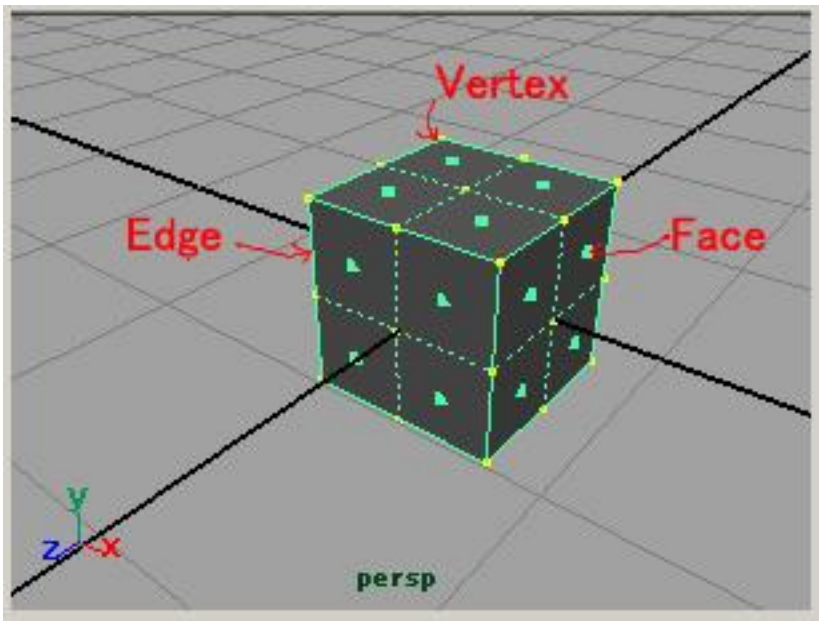


Slika 3.1. Korisničko sučelje Maye

3.1. Modeliranje

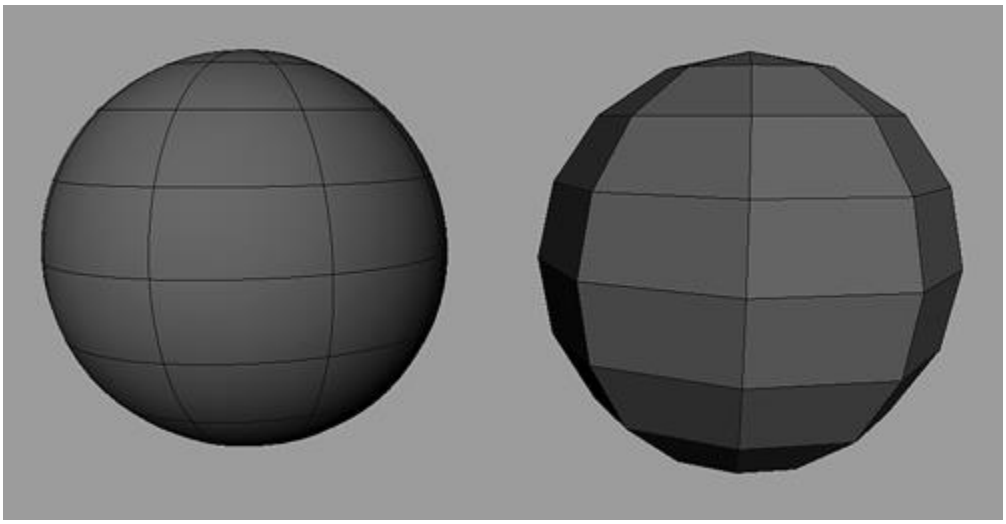
Modeliranje omogućava stvaranje trodimenzionalnih objekata na koje kasnije želimo primijeniti teksturu, animaciju, simulaciju, itd. U Mayi je moguće modeliranje na tri načina [4][5][7]:

1. Poligonalno modeliranje (engl. Polygonal modeling) – Poligon je 3D matematička konstrukcija napravljena od tri ili više točaka koje imaju X, Y i Z koordinate u 3D prostoru. Kod ovakvog modeliranja najvažnija je točka (engl. vertex), odnosno vrh u dvodimenzionalnom prostoru. Dvije točke spojene ravnom linijom tvore rub (engl. edge). Tri točke spojene s tri ruba tvore trokut koji je ujedno i najjednostavniji poligon. Više trokuta ili drugih oblika čine elemente, tj. složenije poligone, a svaki od poligona koji čine element zovu se lice (engl. face).



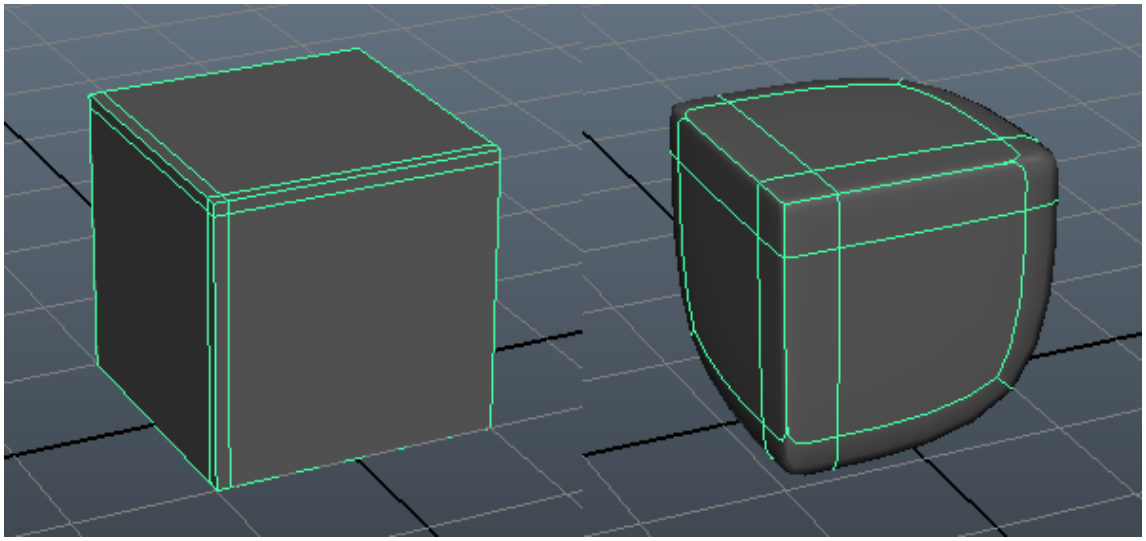
Slika 3.2. Komponente poligona u Mayi [7]

2. NURBS modeliranje (engl. NURBS – Non-Uniform Rational B-Splines) – NURBS objekt je matematički prikaz jedne krivulje ili površine. Krivulje se sastoje od početka, zakrivljenosti i kraja krivulje. Bilo koja točka na NURBS krivulji ili površini ima točan prikaz koordinata koji se uspoređuje s poligonalnim površinama.



Slika 3.3. Lijevo: NURBS, desno: poligon, imaju jednak broj faceva i subdivizija [8]

3. Subdivizijsko modeliranje (engl. Subdivision surface modeling) – modeli se baziraju na kombinaciji poligona i krivulja, tj. baziraju se na poligonalnom modeliranju, a kao rezultat se dobije glatka površina. Subdivizijske površine omogućuju podjelu određenih područja mreže objekta, što daje mogućnost za fino podešavanje ili zaglađivanje određenog dijela površine bez mijenjanja cijele mreže. Subdivizijske površine uglavnom nisu podržane izvan Maye, pa je potrebno prvo pretvoriti takve površine u poligone ili NURBSe prije eksportiranja.



Slika 3.4. Princip subdivizijskog modeliranja [9]

3.2. Teksturiranje

Svaki 3D model koji izradimo možemo i teksturirati, odnosno možemo mu dodijeliti boju, materijal, svojstvo. Na samom početku teksturiranja odabire se tip materijala koji želimo primijeniti na našem 3D modelu. Postoji nekoliko osnovnih materijala koji određuju hoće li model biti mat, sjajan, transparentan ili reflektirajući. Ti materijali su Lambert, Blinn, Phong, Phong E, Anisotropic i Ramp. [6]

Početak procesa teksturiranja je dodavanje materijala 3D modelu. Kada biramo materijal za 3D model, moramo uzeti u obzir kako on reagira na svjetlo, a osnovne postavke koje se mogu prilagoditi svakom materijalu su boja, sjajnost, refleksija, transparentnost i ostali detalji elemenata na sceni. Dobrim odabirom materijala i korištenjem najvažnijih izbornika za teksturiranje

(Attribute Editor, Hypershade, UV Texture Editor) možemo postići izradu što realističnije slike kod renderiranja. [6]

Osnovni materijali [6]:

Lambert – mat materijal bez ikakvog sjaja ili naglašavanja. Po osnovnim postavkama se nalazi na svakom modelu. Upotrebljava se za površine kao što su keramika, kreda, mat boje, drvo, itd.

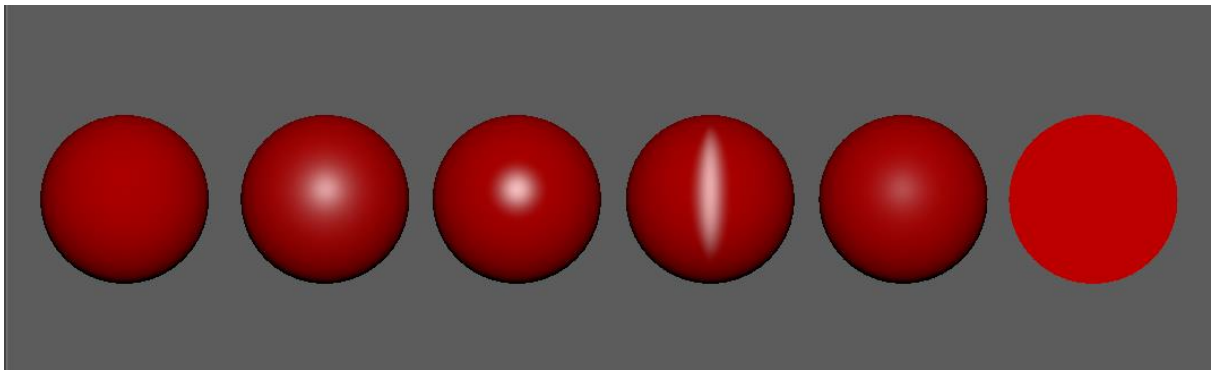
Blinn – reflektirajući, sjajni materijal. Sjaj na materijalu se može prilagođavati prema potrebi, a doživljava se kao metal ili ogledalo. Materijal koji najbolje prikazuje metalne površine.

Phong i Phong E – stakleni, sjajni materijal s oštrim osvjetljenjem. Na Phongu se može podešavati zrcalnost i refleksija boje, dok Phong E ima mekše prijelaze nego običan Phong i omogućuje brže renderiranje.

Anisotropic – postepen prijelaz boja koji se koristi na sjajnim predmetima kao što su CDi, sjajne posude, svila, saten, pero. Mogu se podesiti oštrina prijelaza i refleksija, a polukružni prijelazi se mogu rasporediti i orijentirati na različite načine.

Ramp – materijal kojim se kontroliraju prijelazi između više boja i tekstura, način promjene boje sa svjetlom, kut gledanja između boja. Teksture se također mogu dodati na materijal i mogu se miješati s bojama.

Surface Shader – materijal koji ne sadrži sjenčanja i ne uzima u obzir sjene i svjetla. Najbolje ga je koristiti za pozadine kao što je nebo, za jako osvjetljene znakove i za crtane materijale.



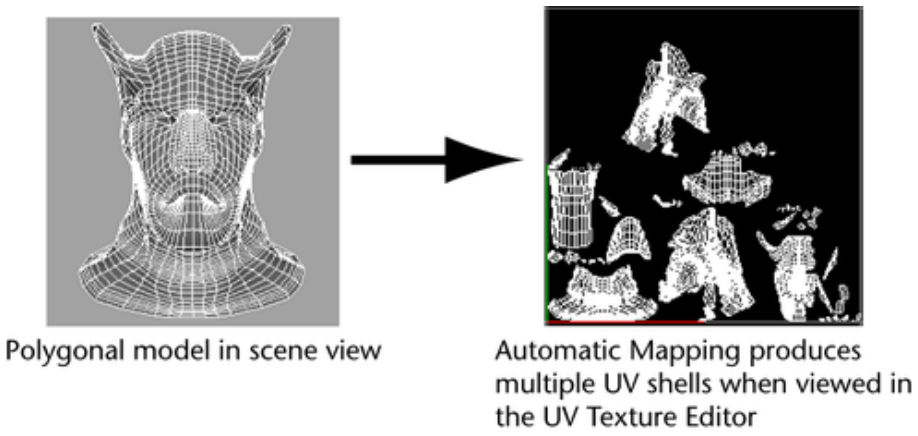
Slika 3.5. Osnovni materijali u Mayi

3.3. UV mapiranje

Prilikom teksturiranja nije važno samo primijeniti materijal i sliku, nego je važno tu sliku i prilagoditi izgledu modela. Taj proces prilagodbe naziva se UV mapiranje. Odnosno, UV mapiranje je proces gdje se izrađuju i uređuju UV točke koje se pojavljuju kao ravne, dvodimenzionalne mreže ploha na dvodimenzionalnoj slici koju smo primjenili na model (tekstura) i pojavljuje se u UV Texture Editoru. UV Texture Editor se koristi za pogled i uređivanje UV točaka, a teksturna slika se postavlja u pozadini UV mreže za lakše uređivanje UV točaka. Sadrži mnogo korisnih alata s kojima se može manipulirati UV točkama. Na svaki tip modela može biti primijenjen različita tehnika UV mapiranja, ovisno što želimo dobiti. Postoji više tehnika UV mapiranja, a izabire se ona koja odgovara obliku i vrsti modela. Takve tehnike nisu nužno točne, stoga je potrebno i dalje urediti UV točke u UV Texture Editoru. [4][6][8]

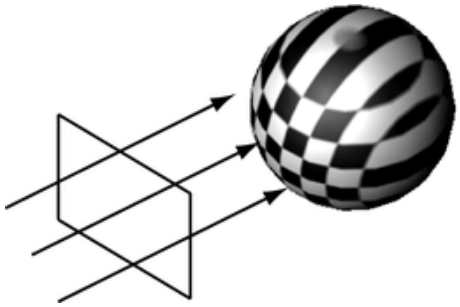
Tehnike UV mapiranja [4][6]:

Automatic UV Mapping – izrađuje poligonalnu UV mrežu pokušavajući naći najbolji UV položaj projiciran iz više pogleda. Ova metoda UV mapiranja je korisna na složenijim oblicima gdje planar, cylindrical ili spherical projekcija ne stvara UV točke koje su korisne.



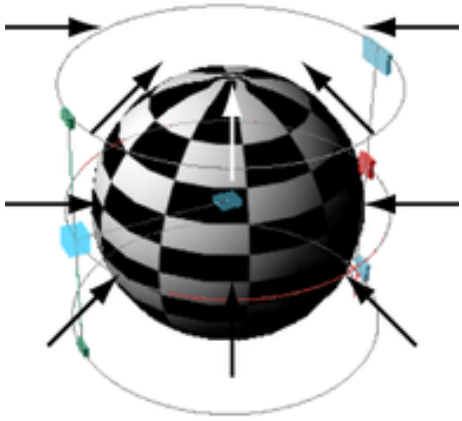
Slika 3.6. Automatic UV Mapping [10]

Planar UV Mapping – projicira UV na mrežu kroz ravnu plohu. Ovaj način projekcije je najbolji za objekte koji su relativno ravni ili su barem u potpunosti vidljivi iz jednog kuta kamere. Planar Mapping obično daje preklapajuće UV ljuske (engl. shell) koje mogu biti savršeno postavljene i izgledati kao jedna UV ljuska.



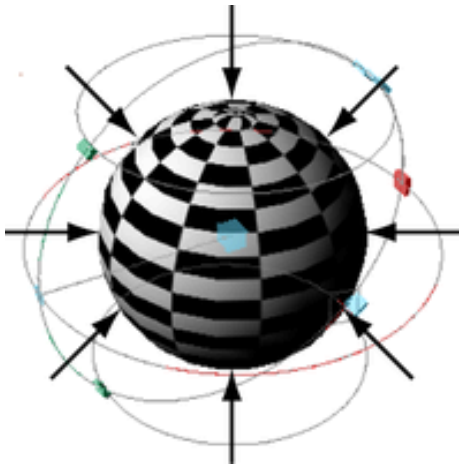
Slika 3.7. Planar UV Mapping [11]

Cylindrical UV Mapping – izrađuje UV točke za objekte temeljenje na cilindričnoj projekciji. Ova projekcija je najbolja za objekte koji su okruglog, cilindričnog oblika i koji su zatvoreni.



Slika 3.8. Cylindrical UV Mapping [12]

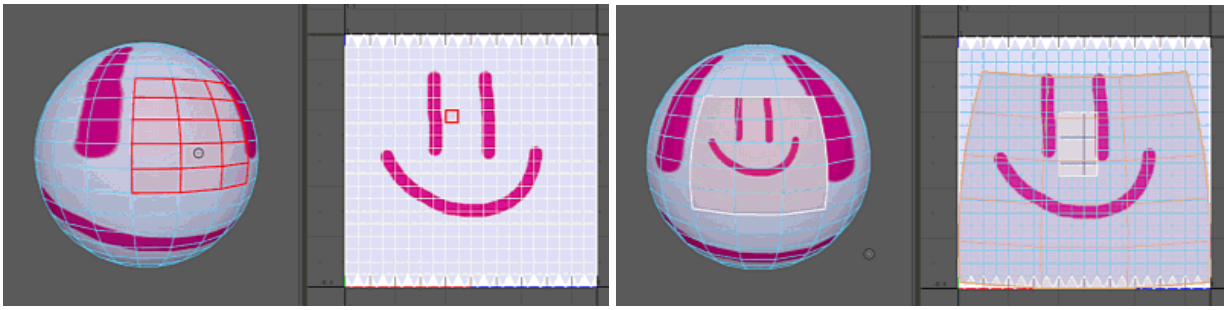
Spherical UV Mapping – izrađuje UV mrežu korištenjem projekcije koja se temelji na sferičnom (kuglastom) omotu oko modela. Ova projekcija je najbolja za oblike koji su vidljivi u sklopu okruglog objekta i potpuno su zatvoreni.



Slika 3.9. Spherical UV Mapping [13]

Camera based UV Mapping – izrađuje UV teksturne koordinate prema trenutnom pogledu kamere. Projekcija funkcionira na principu planarnog mapiranja, odnosno pogled kamere postaje ploha projekcije na objekt.

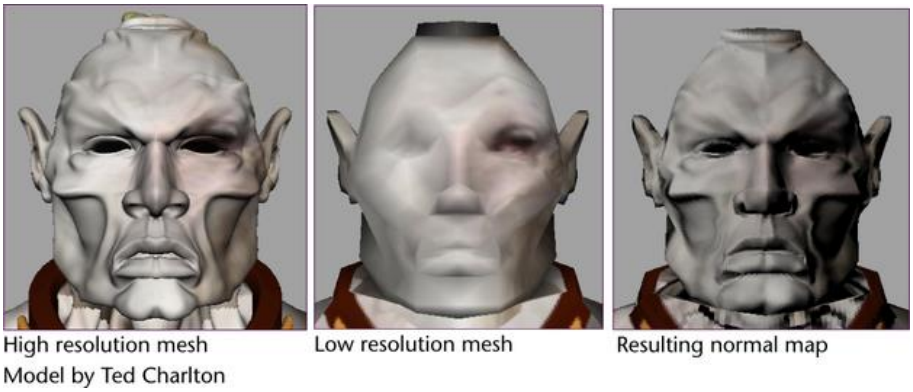
Best plane UV mapping – izrađuje UV točke za poligonalnu mrežu temeljenu na licima (engl. facevima)/točkama (engl. vertexima) koje odredimo projiciranjem najboljih moguće povezanih komponenti.



Slike 3.10. i 3.11. Best plane UV mapping [14]

Contour Stretch UV Mapping – projicira teksturu na selekciju poligona nekog objekta umjesto projiciranja prema specifičnoj formi nekog objekta. Analizira četiri kuta selekcije kako bi se utvrdilo kako je najbolje rastegnuti UV poligone prema priloženoj teksturi. Ovaj način mapiranja se prvi put pojavio u Mayi 2016.

Normal mapping – koristi visoko rezolucijsku mrežu da bi se izradila mreža za nisko rezolucijsku mrežu. Ovaj način mapiranja koristi se u situacijama kada želimo zadržati detalje visoko rezolucijske mreže, ali se mora paziti na zahtjevnost modela, pogotovo ako se nalazi na velikoj sceni s ostalim visoko poligonalnim objektima ili ako imamo limitiranu snagu procesora.



Slika 3.12. Normal mapping [15]

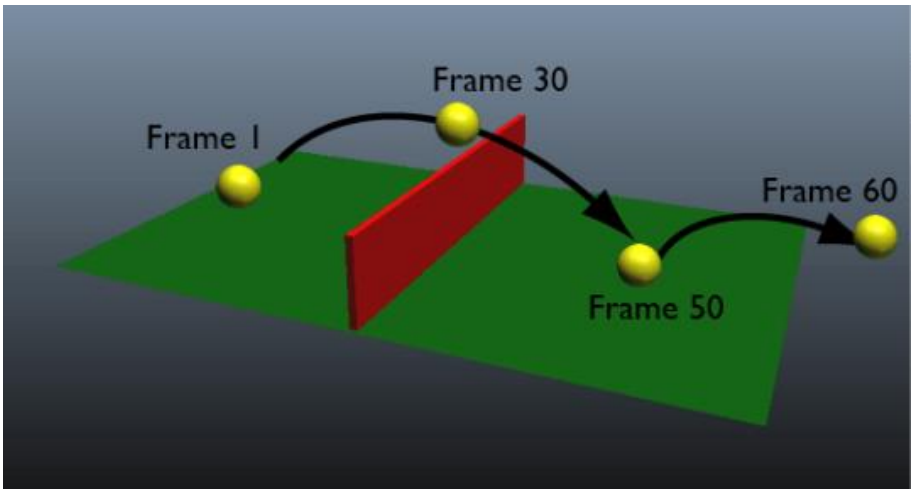
3.4. Animacija

Kad je u pitanju animacija, Maya nam pruža da ostvarimo gotovo sve što smo zamislili. Animacija danas je sveprisutna, stoga je važno imati mogućnost što jednostavnije napraviti željeni proizvod, bilo to za reklamiranje, neku vrstu zabave ili bilo što drugo. Upravo Maya sadrži veliku paletu raznih alata i tehnika koji omogućuju izgradnju i manipulaciju virtualnih trodimenzionalnih objekata.

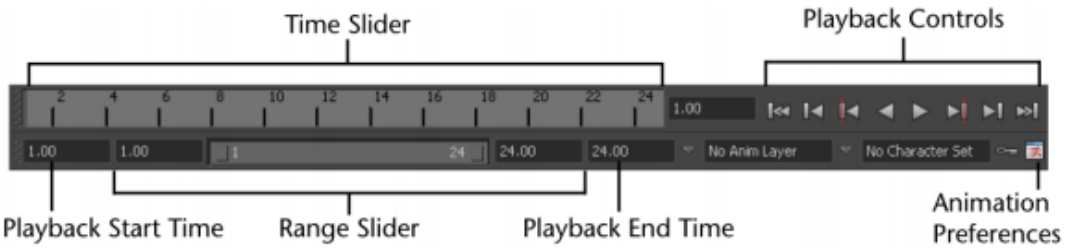
Maya u sklopu svog korisničkog sučelja sadrži traku za animaciju u kojoj upravljamo s postavkama naše animacije. Princip animiranja je taj da postavljamo ključne okvire (engl. keyframeove), odnosno dodjeljujemo vrijednost atributu nekog objekta (npr. mijenjamo translaciju, rotaciju, skaliranje, boju) u neko određeno vrijeme. Većina animacijskih sustava koristi okvire (engl. frameove) kao osnovnu mjernu jedinicu jer se svaki okvir reproducira dovoljno brzo da ostvari iluziju gibanja. [4]

Frame rate (Frames per second, FPS) ili okvir po sekundi, koji se koristi za reprodukciju animacije temelji se na mediju na kojem će se animacija reproducirati (npr. TV, video igra, itd.). Kada postavimo nekoliko ključnih okvira u različito vrijeme s različitim vrijednostima, Maya

generira te vrijednosti atributa između svakog označenog vremena koji se reproducira na sceni. Rezultat toga je pokret ili promjena tih objekata i njihovih atributa tijekom određenog vremena. [9]



Slika 3.13. Animacija po frameovima (okvirima) [16]



Slika 3.14. Animacijska traka u Mayi [16]

Elementi animacijske trake u Mayi [9]:

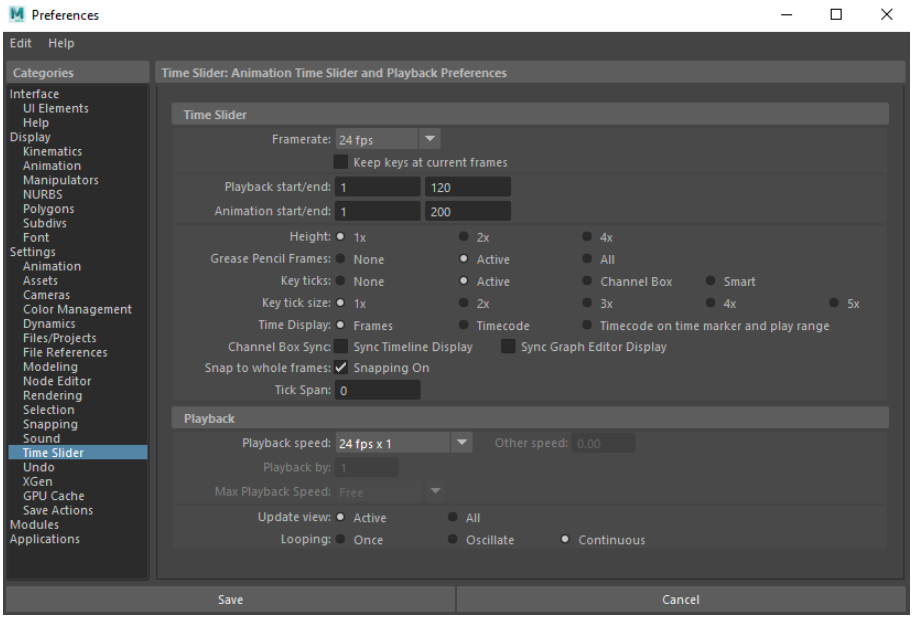
Time Slider – prikazuje raspon reprodukcije i ključnih okvira koji su postavljeni za odabrani objekt. Ključni okviri se prikazuju kao crvene okomite crte. Kvadratić s desne strane Time Slidera omogućuje postavljanje trenutnog vremena (okvira) animacije.

Playback Controls – kontrole reprodukcije animacije. Kontrole su: Go to Start of Playback Range (idite na početak raspona reprodukcije), Step Back One Frame (korak unazad za jedan okvir), Step Back One Key (korak unazad za jedan ključni okvir), Play Backwards (reprodukcija unazad), Play

Forwards (reprodukcija naprijed), Step Forward One Key (korak naprijed za jedan ključni okvir), Step Forward One Frame (korak naprijed za jedan okvir), Go to End of Playback Range (idite na kraj raspona reprodukcije).

Range Slider – kontrolira raspon okvira koji se reproduciraju kada se klikne na gumb za reprodukciju. Playback Start Time kućica označava početak reprodukcije, tj. prvi okvir koji će se reproducirati, a Playback End Time kućica označava kraj reprodukcije, tj. zadnji okvir koji će se reproducirati.

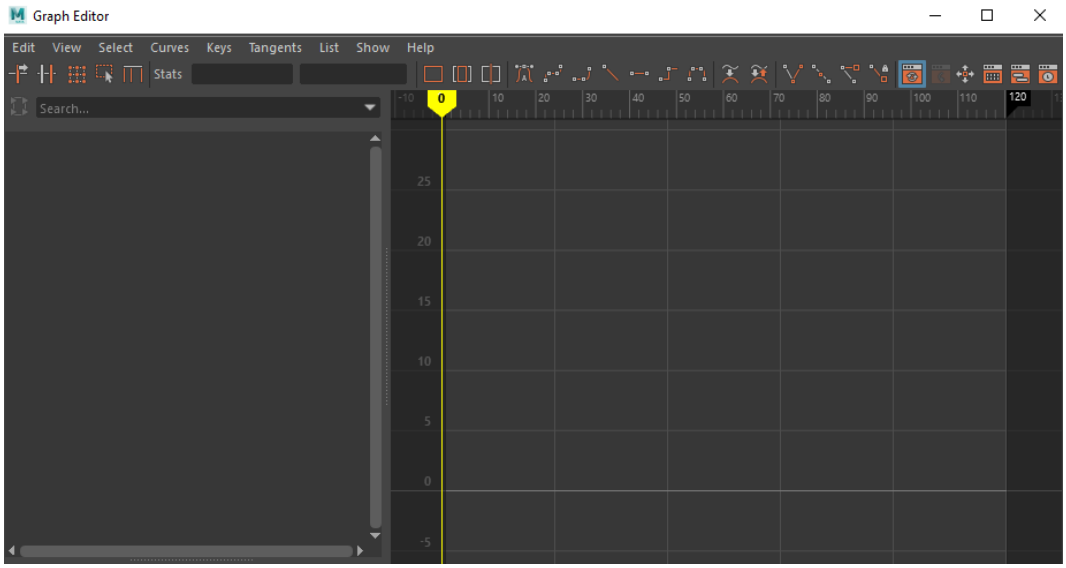
Animation Preferences – prikazuje prozor postavki animacije.



Slika 3.15. Time Slider (postavke animacije)

3.4.1. Graph Editor

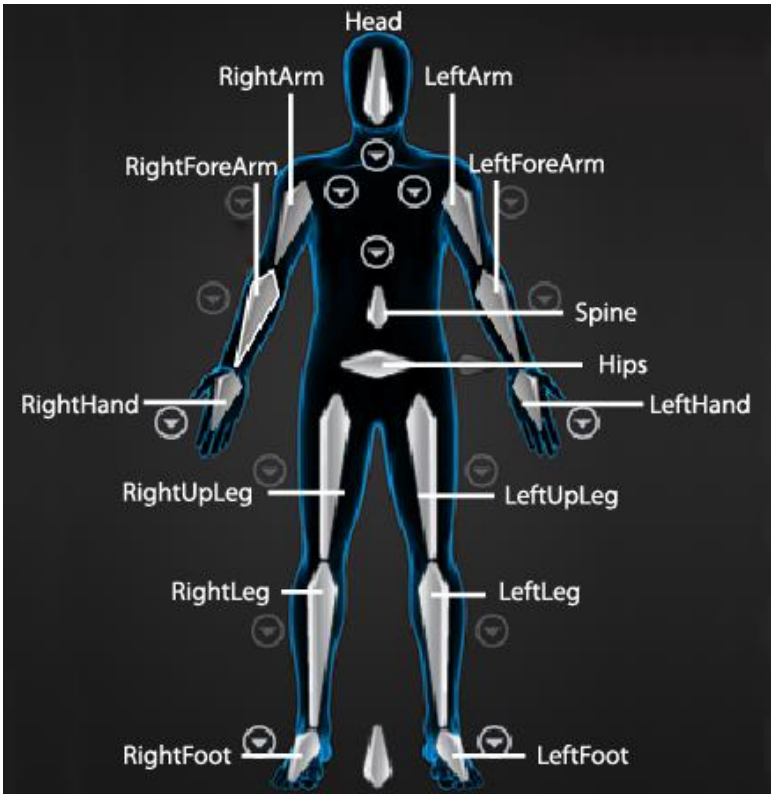
Graph Editor – dvodimenzionalni prikaz animacije u sceni. Editor ima osi x i y. X (horizontalna os) obično prikazuje vrijeme u okvirima. Y (vertikalna os) prikazuje vrijednost ključnih okvira u jedinicama za translaciju, skaliranje i rotaciju. Kada se animacija rafinira, kontrole u Graph Editoru nude veću fleksibilnost i vizualne povratne informacije o interpolaciji između ključnih okvira. [8]



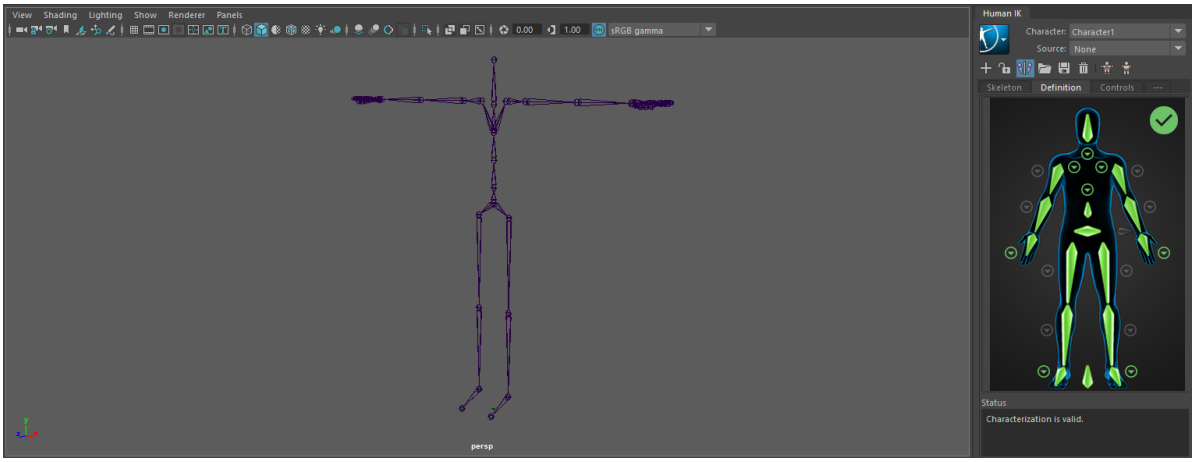
Slika 3.16. Graph Editor

3.4.2. HumanIK

HumanIK – animacijski middleware koji omogućuje inverznu kinematku (engl. IK – inverse kinematics) cijelog tijela (kostura). HumanIK alati u Mayi omogućuju postavljanje ključnih okvira (engl. keyframing) za cijelo tijelo (kostur) lika ili samo određene dijelove tijela (kostura). Svaki lik kojeg želimo animirati pomoću HumanIKa, mora biti postavljen kao HumanIK lik, odnosno da bi funkcionirao kao HumanIK lik, mora imati važeću definiciju kostura (skeletalnu strukturu). HumanIK solver očekuje da čvorovi (engl. nodes) lika budu povezani hijerarhijski. Npr. desno rame je roditeljski čvor (engl. parent node) desnog lakta, koji je roditeljski čvor desnog ručnog zgloba. HumanIK bio-mehanički model je opremljen "znanjem" na koje načine se ti čvorovi (engl. nodes) mogu kretati. Kada solver treba premjestiti jedan ili više čvorova da zadovolji zahtjeve IK efekta ili izvorne animacije, koristi to ugrađeno "znanje" za stvaranje novih poza. [4]

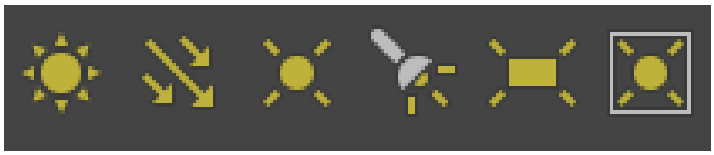


Slika 3.17. Dijelovi kostura u HumanIK-u [17]



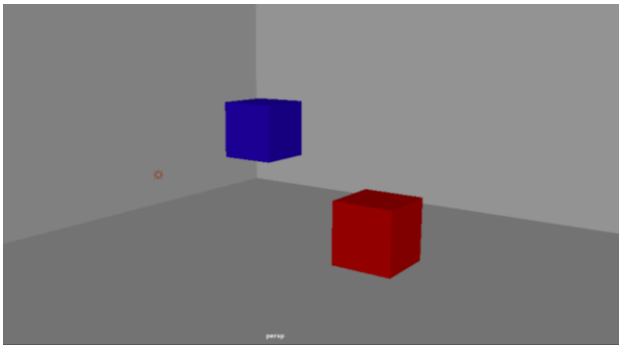
Slika 3.18. Pravilno složena hijerarhija kostura u Mayi

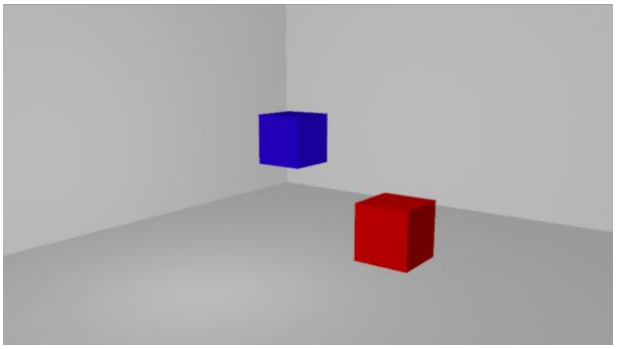
3.5. Svjetla



Slika 3.19. Vrste svjetla u Mayi

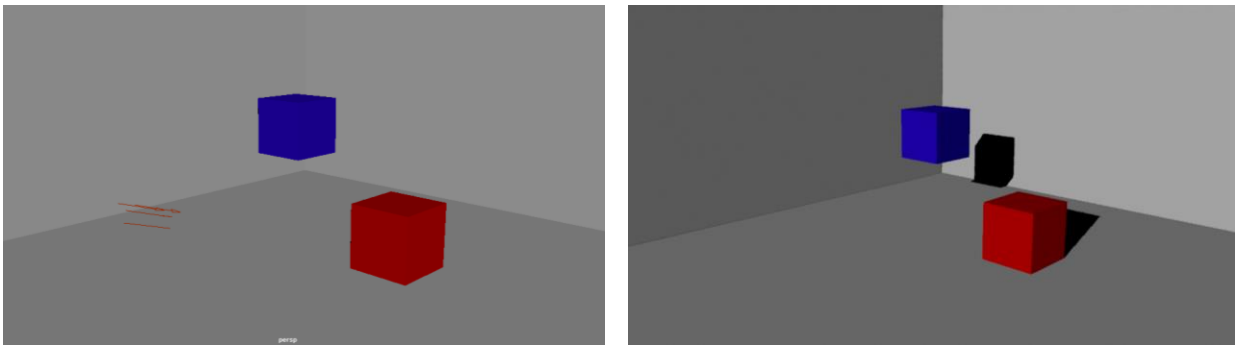
Ambient light – vrsta svjetla koja je u stvarnom životu široko rasprostranjena. U ovom slučaju, to je neizravno svjetlo koje je odbijeno ili se prenosi putem drugih objekata. Osvjetljava čak i područja koja nisu izravno osvjetljena nekim drugim izvorom svjetlosti. [4]





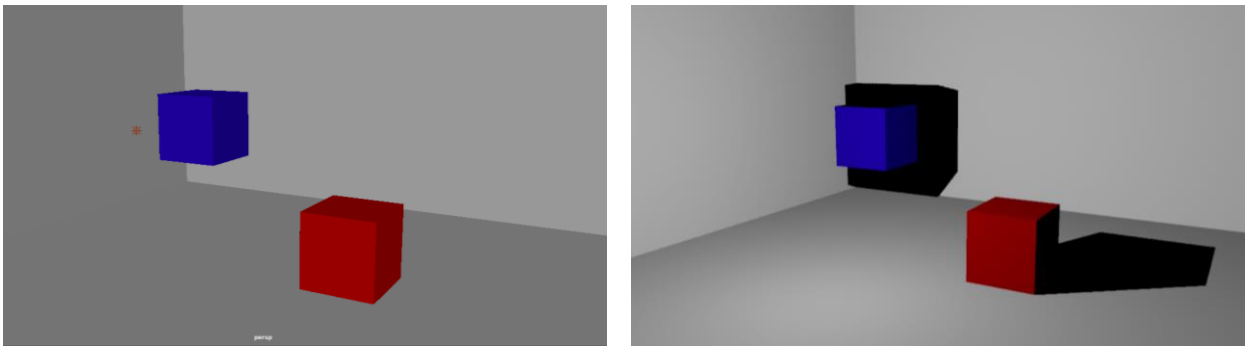
Slike 3.20. i 3.21. Ambient light (prije i poslije renderiranja)

Directional light – postavlja jedan vektor za svu svjetlost koju baca i svjetlo pada na svaki objekt iz istog kuta, bez obzira gdje se objekt nalazi. Sve sjene koje baca ovakva vrsta osvjetljenja padaju u istom smjeru i ortogonalna su projekcija svakog objekta. Nije bitno gdje je svjetlo postavljeno u odnosu na objekt koji osvjetljava, nego je važan smjer u kojem je svjetlost usmjerena, odnosno kut kojeg svjetlo koristi. [4]



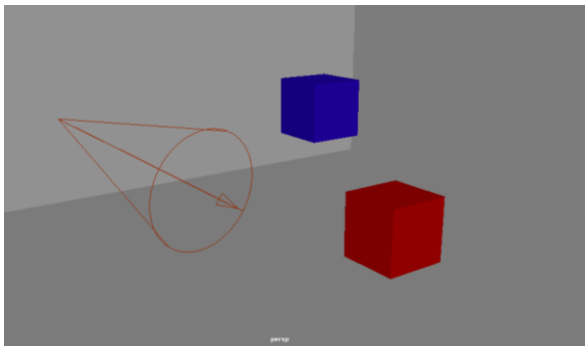
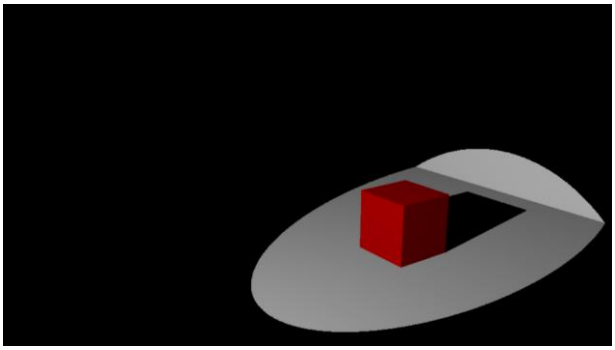
Slike 3.22. i 3.23. Directional light (prije i poslije renderiranja)

Point light – simulira zrake koje svijetle iz jedne neizmjerno male točke u prostoru. Emitira svjetlost jednako u svim smjerovima poput žarulje. [4]



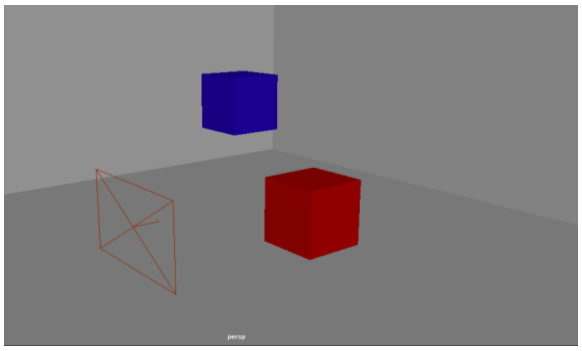
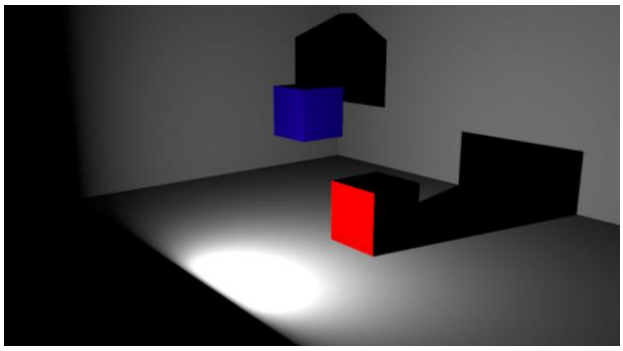
Slike 3.24. i 3.25. Point light (prije i poslije renderiranja)

Spot light – simulira svjetlost iz jedne točke, slično kao i point light, ali razlika je u tome što spot light ima ograničeno osvjetljenje u obliku stošca, odnosno izgleda kao snop svjetla. Ovo svjetlo ima dodatne kontrole i opcije koje ostala svjetla nemaju, kao npr. projiciranje slikovne mape iz svijeta ili stvaranje svjetlosne zrake koja izgleda kao da se probija kroz maglu. [4]

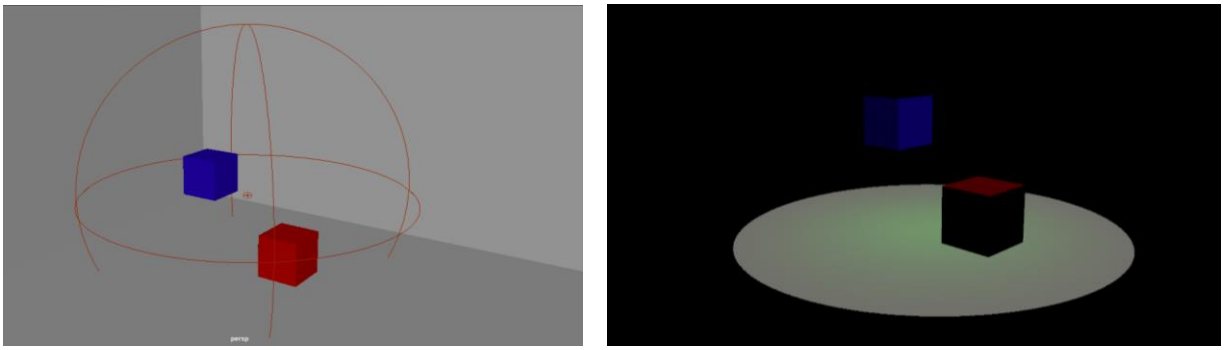
Slike 3.26. i 3.27. Spot light (prije i poslije renderiranja)

Area light – dvodimenzionalni pravokutni izvor svjetlosti. Upotrebljava se za simulaciju pravokutnih refleksija na prozorskim površinama. U usporedbi s drugim svjetlima, area light uzrokuje duže vrijeme renderiranja, ali zauzvrat daje veću kvalitetu svjetla i sjena. [4]

Slike 3.28. i 3.29. Area light (prije i poslije renderiranja)

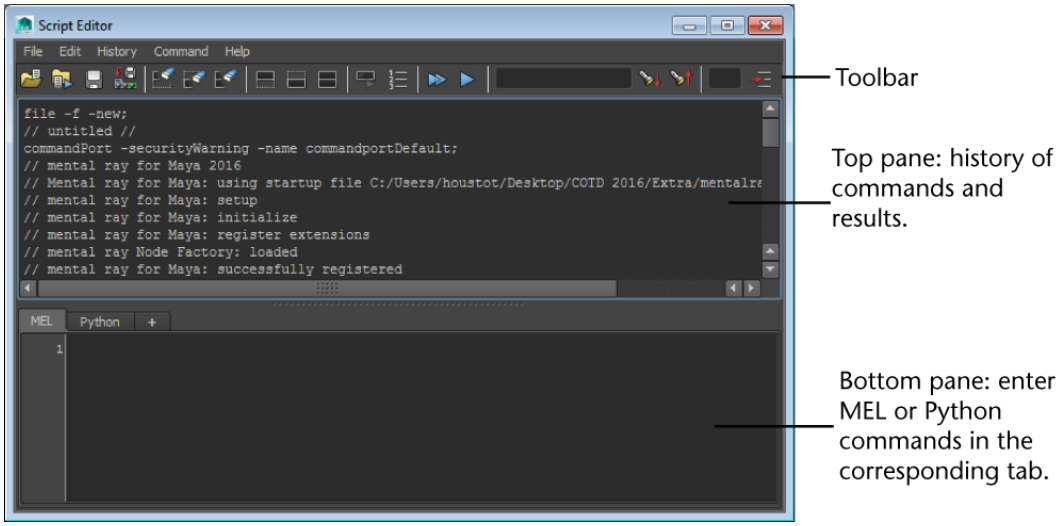
Volume light – prednost ovog svjetla je što imamo vizualni prikaz obujma svjetlosti (prostora unutar kojeg se nalazi). Svjetlo se može prikazati u više boja (gradijent), što je korisno za volumensku maglu. [4]



Slike 3.30. i 3.31. Volume light (prije i poslije renderiranja)

3.6. Programiranje

Maya koristi skriptni jezik MEL (Maya Emmbeded Language) koji je jednostavan za korištenje i omogućuje mijenjanje izvornih funkcija. Većina zadataka koje možemo obaviti putem GUIa (grafičko korisničko sučelje) Maye, može se postići i s MELom, kao i neke zadatke koji nisu dostupni u GUIu. MEL nudi metode kojima možemo ubrzati obavljanje kompliciranih i ponavljajućih zadataka te je korisnicima omogućeno redistribuirati određen set naredbi koje bi drugima mogle koristiti. Uz MEL, Maya podržava i Python, programski jezik koji se koristi za aplikacije, web dizajn i sl. [4]



Slika 3.32. Script Editor [18]

3.7. Konkurencija

Iako je Maya jedan od najpopularnijih i najkorištenijih programa u svijetu 3Da, postoje i drugi programi koji joj svakako konkuriraju. Neki su bolji za modeliranje, neki za teksturiranje ili animiranje, svaki od njih ima neke svoje prednosti i nedostatke, presudno je to što nama zapravo treba od svega navedenog. Jedni od većih konkurenata Mayi su 3ds Max i Blender.

3ds Max je također Autodeskov proizvod. Sadrži vrlo snažan set alata za modeliranje koji olakšava izradu projekta koliko god on težak bio. Njegovo sučelje je također relativno jednostavno za razumijevanje novim korisnicima te je dizajnirano tako da ima jednostavan i brz pristup svim najvažnijim alatima. Jedna od važnijih prednosti 3ds Maxa je to što može obrađivati veći broj podataka te poboljšavati performanse i vizualnu kvalitetu. Mana mu je, u odnosu na Mayu, to što se može koristiti samo na 64-bitnim Windowsima, ne podržava Linux ili Mac OS X. [14]

Blender, osim što je besplatan, osvojio je srca mnogih developera indie igara i hobista. Jedan od najboljih karakteristika Blendera su njegovi alati za modeliranje. Iako sva tri programa imaju sposobnost stvaranja istih asseta, poznato je da Blender ima vrlo intuitivan i lako razumljiv tijek modeliranja. U njemu postoji poseban Edit mode koji služi za uređivanje posebno za svaki objekt te se na taj način izbjegava označavanje objekata koji se ne žele uređivati. Selekcija je obrnuta od one u Mayi, npr. vertexi se označavaju desnom tipkom miša. Što se tiče operativnih sustava, radi podjednako dobro na Windowsima, Linuxu i Mac OS X-u. [13]

PROGRAM	PLATFORME	ULAZNI I IZLAZNI FORMATI (najvažniji)	CIJENA
Maya	Windows, Linux, Mac OS X	fbx, obj, mel, mb, ma, eps, ai, mov, dxf, jpg, png, bmp, gif	1 mjesec: 190$ 1 godina: 1505$ 2 godine: 2859.50$ 3 godine: 4063.50$ (rujan 2018.)
3ds Max	Windows	3ds, fbx, sxf, dwg, xml	1 mjesec: 190$ 1 godina: 1505$ 2 godine: 2859.50$ 3 godine: 4063.50$ (rujan 2018.)
Blender	Windows, Linux, Mac OS X	obj, fbx, jpg, png, gix, avi, mov	besplatno (rujan 2018.)

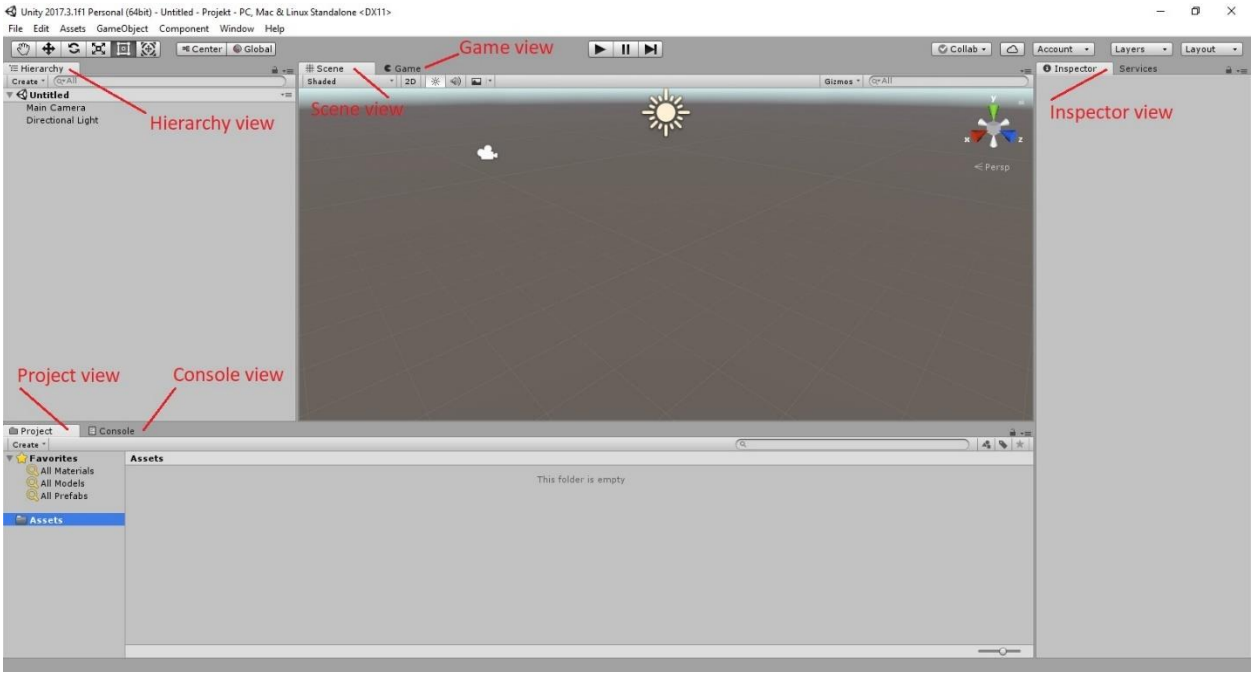
Tablica 3.1. Maya, 3ds Max i Blender - usporedba

4. Unity

Unity game engine je program razvijen od strane Unity Technologiesa koji je prvi put najavljen i objavljen u lipnju 2005. na Appleovoj konferenciji Worldwide developers kao OS X ekskluzivni game engine no proširio se i na ostalo računalno i mobilno tržište (iOS, Android, Windows, Playstation 3 i 4, Xbox One, Mac, itd.). Od 2018. engine podržava 27 platformi. Engine se može koristiti za izradu trodimenzionalnih i dvodimenzionalnih igara, kao i simulacije za stolna i prijenosna računala, kućne konzole, pametne televizore i mobilne uređaje. Nekoliko glavnih verzija Unitya je objavljeno od samog njegovog početka, a najnovija stabilna inačica je Unity 2018.2.6 koja je izašla 30. kolovoza 2018. Osim samostalnih game developera i velikih tvrtki, Unity je privukao i programere jer nudi kvalitetan programski jezik C#. Prethodno je podržavao i Boo, koji je uklonjen s izlaskom Unitya 5 te verziju JavaScript (zvana UnityScript) koja je uklonjena u kolovozu 2017. nakon izlaska verzije Unity 2017.1 u korist C#. Kada su u pitanju ulazni i izlazni formati, Unity podržava većinu programa za 3D modeliranje, uređivanje slika, videomontažu, itd. (Maya, Blender, 3ds Max, Adobe Photoshop, Fireworks).

Verzija koja je trenutno dostupna korisnicima je Unity 2018.2.6. Dostupan je u Personal verziji (za početnike, studente, hobiste koji žele početi s Unityem i istražiti mogućnosti) koja je besplatna, Plus verziji (za kreatore koji su ozbiljni u vezi ostvarenja svoje vizije) koja košta 35$ mjesečno ili 25$ mjesečno ako se korisnik pretplati na godinu dana i Pro verziji (za profesionalce koji trebaju potpunu fleksibilnost) koja košta 125$ mjesečno.

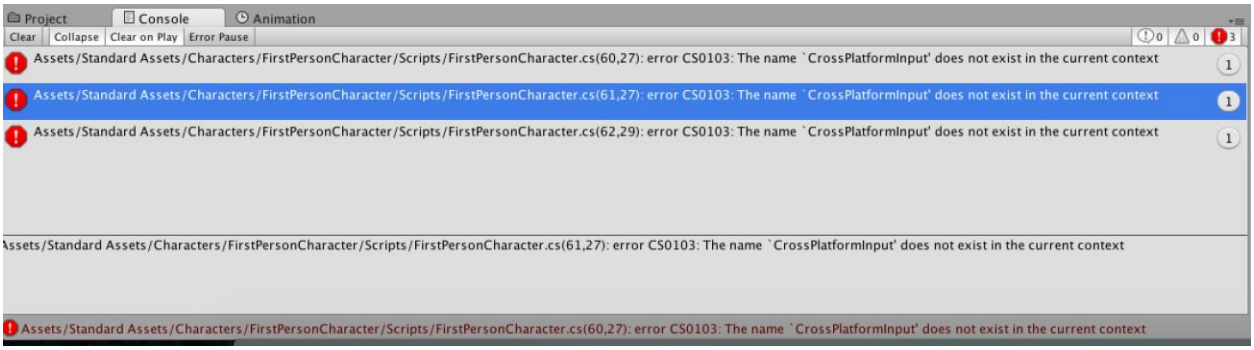
4.1. Korisničko sučelje – dijelovi Unitya



Slika 4.1. Korisničko sučelje Unitya

Project view – pruža nam strukturu datoteka naših komponenta igre, uključujući objekte, artove, skripte, itd. Ovaj prikaz ima dva stupca. Prvi stupac sadrži strukturu datoteka, kao i favorite, što pomaže u brzom pronalasku naših asseta. Drugi stupac prikazuje sadržaj datoteka iz prvog stupca. [14]

Console view – prikazuje poruke, upozorenja, greške prilikom kreiranja igre. Najčešće greške koje se javljaju u ovom prikazu uglavnom su vezane uz skriptu, što uvelike olakšava rješavanje nastalih problema. [14]



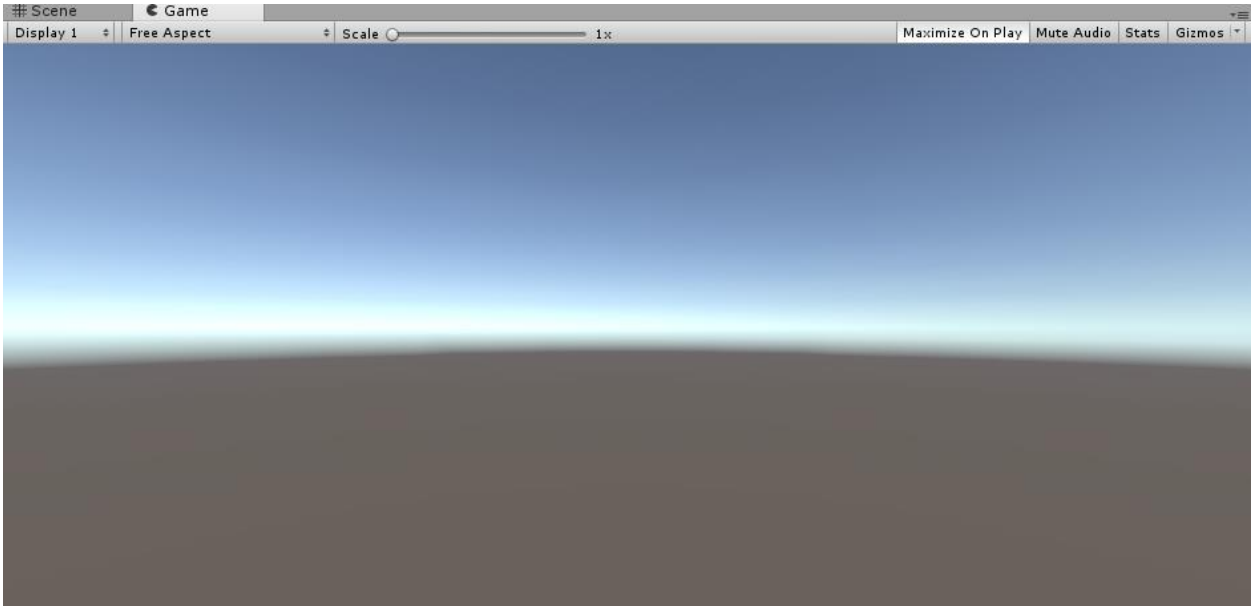
Slika 4.2. Console view [19]

Inspector view – kada je neki objekt selektiran, u ovom prozoru se pojavljuju detalji objekta. U većini slučajeva promjene na objektu mogu se izvršiti upravo u Inspectoru. [13]

Hierarchy view – prikazuje hijerarhijski popis svakog objekta na sceni. Ako neki objekt dodamo ili obrišemo sa scene, ista akcija se dogodi i u Hierarchy viewu. [13]

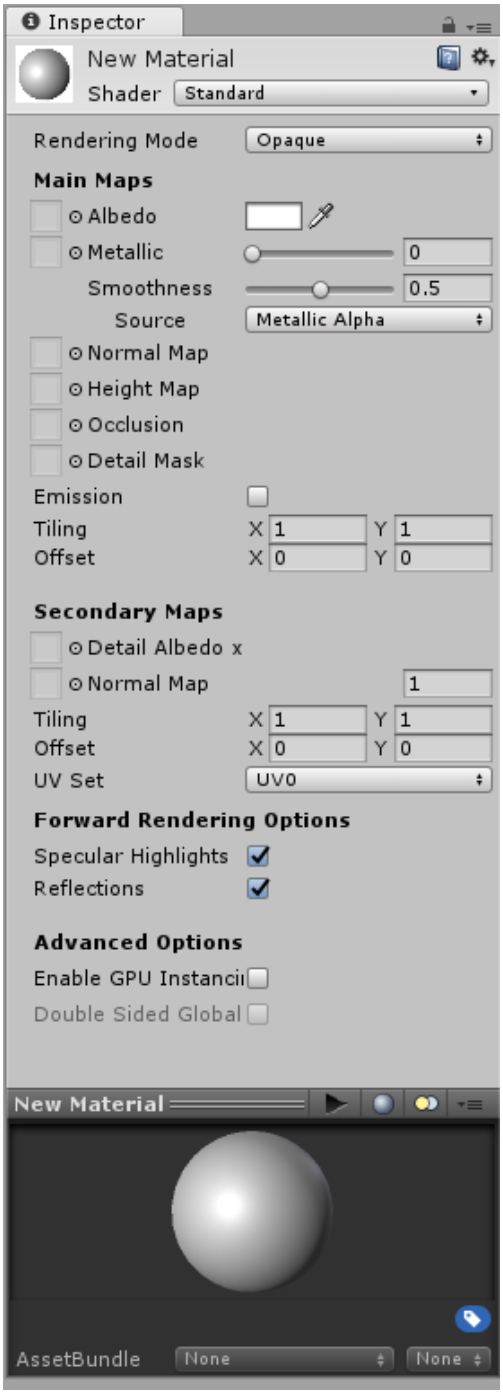
Scene view – prikazuje scenu koja je ekvivalentna levelu u igri. Na scenu stavljamo objekte poput likova (engl. characters), zgrada, terena, itd. Te iste objekte na sceni možemo premjestiti, rotirati, skalirati pomoću alata za preoblikovanje (engl. transform tools). [13]

Game view – ovaj prikaz je jedan od najboljih mogućnosti Unitya i on nam daje šansu da igru probamo unutar game enginea, što uvelike štedi vrijeme developerima s obzirom da ne moraju prvo compileati igru da bi je probali. Ako napravimo neke izmjene unutar enginea dok je igra pokrenuta, te promjene će se odmah vidjeti unutar igre, ali kad prekinemo igru, sve promjene koje smo napravili neće se spremiti. To je odličan način da eksperimentiramo s igrom, bez da pokvarimo skriptu ili neku drugu komponentu u igri. [13]



Slika 4.3. Game view

4.2. Materijali, shaderi i teksture

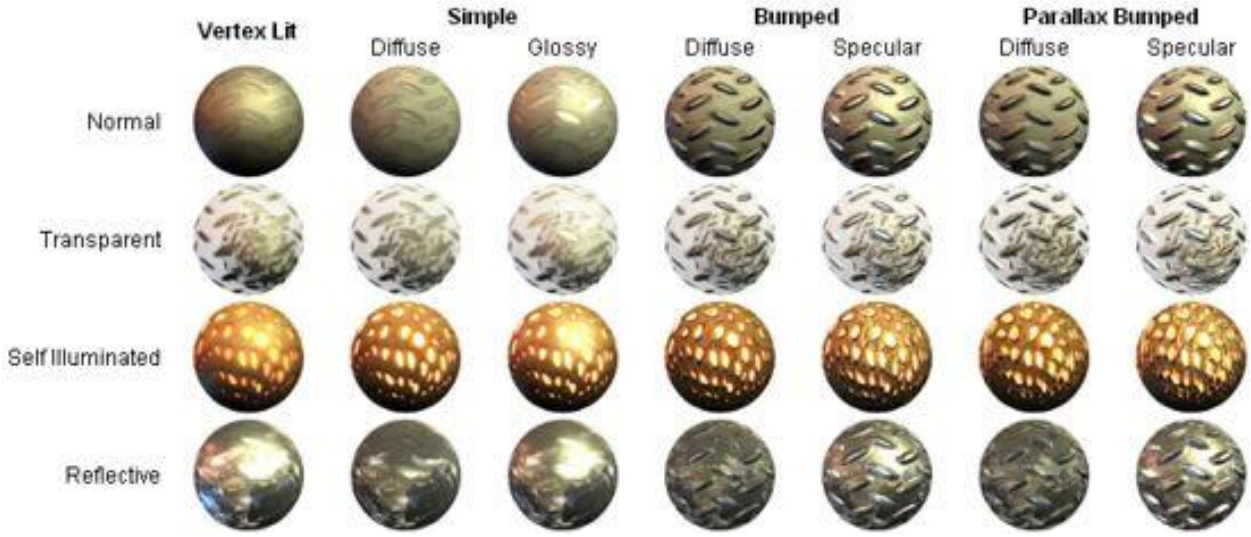


Slika 4.4. Standardni shader

Materijali određuju kako bi površine trebale biti renderirane, uključujući teksture koje koristi, boje, ponavljanje teksture (engl. tiling) i još mnogo toga. Dostupne opcije materijala kojima možemo manipulirati ovise o tome koji shader materijal koristi. [14]

Shaderi su male skripte koje sadrže matematičke izračune i algoritme za izračunavanje boje svakog renderiranog piksela na temelju konfiguracije materijala i svjetla na sceni. Za većinu normalnih renderiranja (kao što su prikazivanje likova, scene, okruženja, punih i transparentnih objekata) standardni shader je obično najbolji izbor. To je vrlo prilagodljiv shader koji je sposoban prikazati mnogo različitih tipova površina poprilično realno. Postoje i druge situacije u kojima se mogu koristiti različiti posebno napravljeni shaderi ili čak posebno napisani (programirani) shaderi, npr. tekućine, folije, refrakcijsko staklo, efekti čestica, crtani, ilustracijski ili neki drugi artistički efekti ili specijalni efekti kao što su noćni vid, rendgenski vid. [14]

Teksture su bitmap slike. Materijal može sadržavati reference na teksture tako da shader materijala može koristiti teksture prilikom izračuna površinske boje objekta. Uz temeljnu boju (Albedo) objekta, teksture mogu predstavljati i mnoge druge karakteristike materijala kao što je njegova refleksija ili hrapavost. [14]



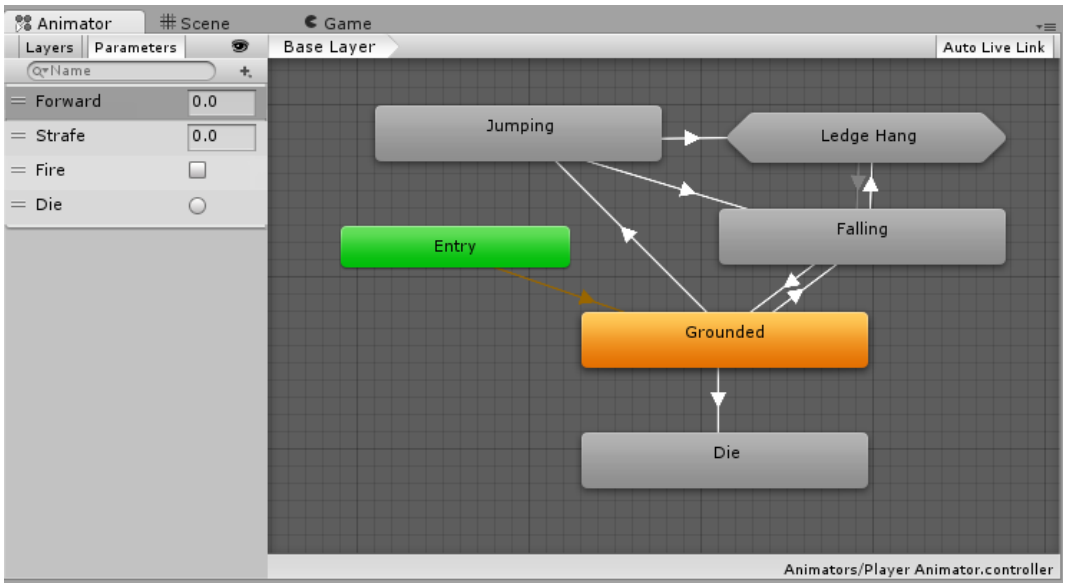
Slika 4.5. Vrste shadera i njihov izgled [20]

4.3. Animacija

Animacija se definira kao simulirani pokret koji se stvara prikazivanjem niza slika ili okvira (engl. frameova). U Unityu,animacija je prikazana slično metodama korištenima u filmovima, što uključuje prikaz određenog broja progresivnih okvira svake sekunde. Animacije se koriste kako bi oživjeli likove kojima želimo upravljati. Kad igrač koristi strjelice na tipkovnici ili neki drugi uređaj (npr. joystick) za pomicanje glavnog lika u igri, taj lik je animiran. Dakle, igra reagira na tipke koje stišćemo i ovisno o tome koju tipku smo stisnuli ta animacija se pokrene. Animacije se obično izrađuju u nekom od programa za 3D modeliranje kao što su Maya, 3ds Max ili Blender, ali moguće je stvoriti animaciju i unutar Unitya. [13]

Animacije stvorene u Unityu omogućuju promjenu položaja, rotacije, veličine, svojstva (boja materijala, intenzitet svjetla, jačinu zvuka), svojstva unutar vlastitih skripti (float, integer, enum, vektor i boolean varijable), vrijeme pozivanja funkcija unutar skripte. Animation Clips (animacijski isječci) jedni su od ključnih elemenata Unityevog animacijskog sustava. Kao što sam navela, omogućen je uvoz animacija iz drugih programa ili izrada animacije unutar Unitya. Animator Controller omogućuje organizaciju i održavanje skupa animacijskih isječaka i pridruženih animacijskih prijelaza za neki lik ili objekt. U većini slučajeva, normalno je imati više animacija i prijelaza između njih za određene uvijete/radnje u igri. Npr. moguće je prebaciti se iz

animacije hodanja u animaciju skakanja prilikom pritiska tipke space. Postoje slučajevi kada nam je potrebna samo jedna animacija, ali i tada moramo animacijski isječak staviti u controller kako bi ju mogli koristiti na nekom objektu u igri. [14]

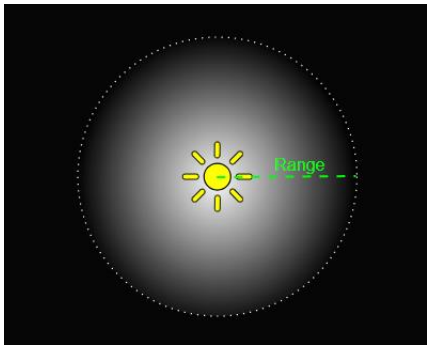


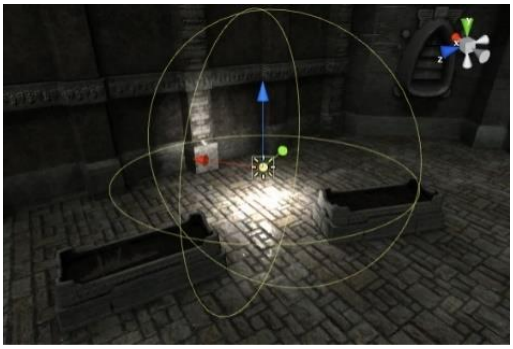
Slika 4.6. Animator prozor u Unityu [21]

4.4. Svjetla

Svjetla su bitan dio svake scene. Dok mreže i teksture definiraju oblik i izgled scene, svjetla definiraju boju i ugođaj 3D okruženja. Vjerojatnost je velika da ćemo koristiti više od jednog svjetla na sceni. Postoji nekoliko vrsta svjetla koji emitiraju boju koju im dodijelimo na različite načine. Osvijetljenost nekih svjetala ovisi o njihovoj udaljenosti od objekta ili drugačijim pravilima o kutu iz kojeg zrake padaju iz izvora. Osnovna svjetla u Unityu su: directional, spot, point i area light. Directional, spot i point svjetla se smatraju realtime svjetlima. To znači da oni bacaju svjetlo na cijelu scenu i ažuriraju svaki okvir (engl. frame). Ako mičemo svjetla ili objekte po sceni, ta svjetla će se automatski ažurirati. Te izmjene su odmah vidljive na Scene i Game prikazima (engl. view). [14]

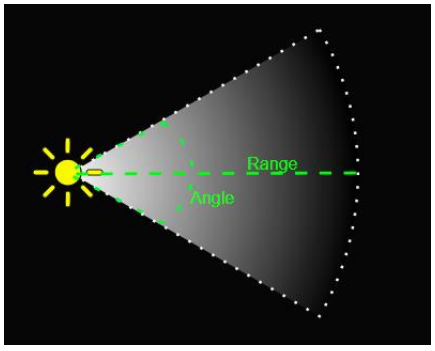
Point light – ovo svjetlo se nalazi u jednoj točki u prostoru i emitira svjetlost u svim smjerovima jednako. Intenzitet svjetla se može smanjiti čak i na nulu ako svjetlost udaljimo predaleko od objekta/scene. Intenzitet svjetla je obrnuto proporcionalan kvadratu udaljenosti od izvora. To je poznato kao kvadratni zakon reciprociteta (inverse square law) i slično je načinu na koji se svjetlo ponaša u stvarnom svijetu. Ovo svjetlo koristi se za simulaciju lampe, ali se može koristiti i kao iskra ili efekt nekakve eksplozije. [14]

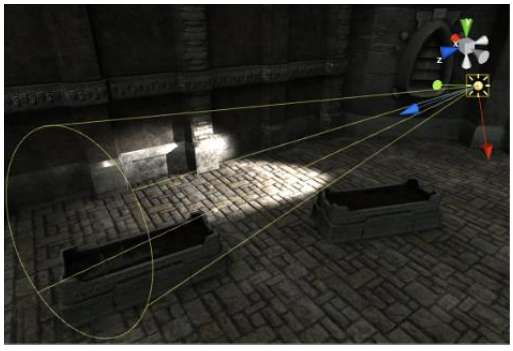




Slike 4.7. i 4.8. Point light [22]

Spot light – poput point lighta, spot light ima određenu poziciju i raspon do kojeg svjetlo pada. Međutim, spot light je ograničen kutom, što rezultira osvjetljenjem u obliku stošca. Intenzitet se smanjuje na rubovima stošca (oblika prema kojem svjetlo pada). Spot light se obično koristi za umjetne izvore svjetla kao što su svjetiljke, svjetla automobila, reflektori. [14]

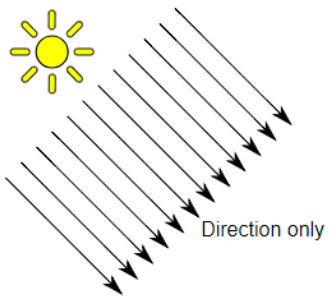


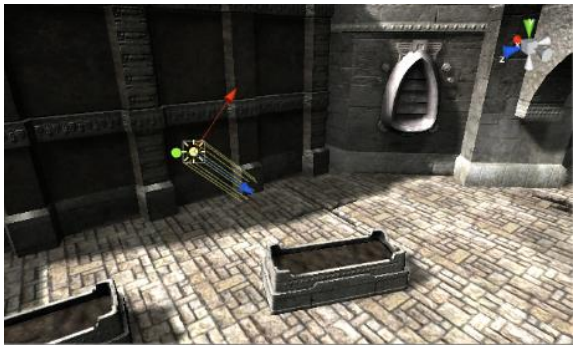


Slike 4.9. i 4.10. Spot light [22]

Directional light – ova vrsta svjetla je vrlo korisna za stvaranje efekata kao što je sunčeva svjetlost. Ponašajući se poput sunca, ovo svjetlo može se smatrati kao beskrajno udaljen izvor

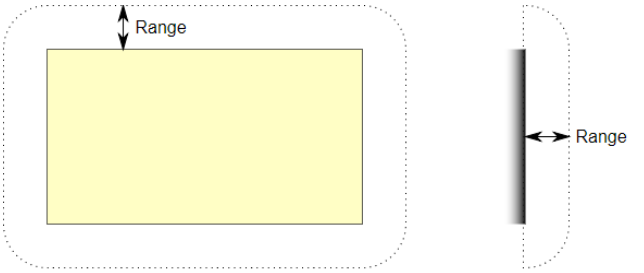
svjetlosti. Nema određenu poziciju izvora tako da se može postaviti bilo gdje na sceni. Svi objekti na sceni su osvjetljeni kao da svjetlost pada uvijek iz istog smjera. Nije definirana nikakva konkretna udaljenost na kojoj mora biti od objekta tako da intenzitet ovog svjetla nikad neće slabiti. Koristi se za simulaciju sunca ili mjeseca. Po defaultu, svaka nova scena u Unityu ima directional light. [14]

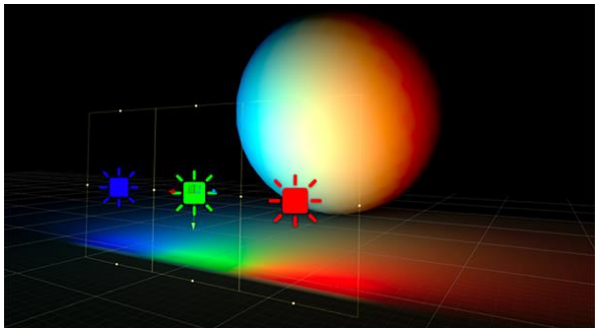




Slike 4.11. i 4.12. Directional light [22]

Area light – svjetlost određena pravokutnikom u prostoru. Emitira se u svim smjerovima jednako iz svoje površine, ali samo s jedne strane pravokutnika. Ne može se ručno namjestiti udaljenost ovog svjetla, ali intenzitet će se smanjiti prema kvadratnom zakonu reciprociteta na putu od izvora. Može simulirati manje izvore svjetlosti (kao osvjetljenje interijera kuće) s realističnijim efektom nego point light. [14]

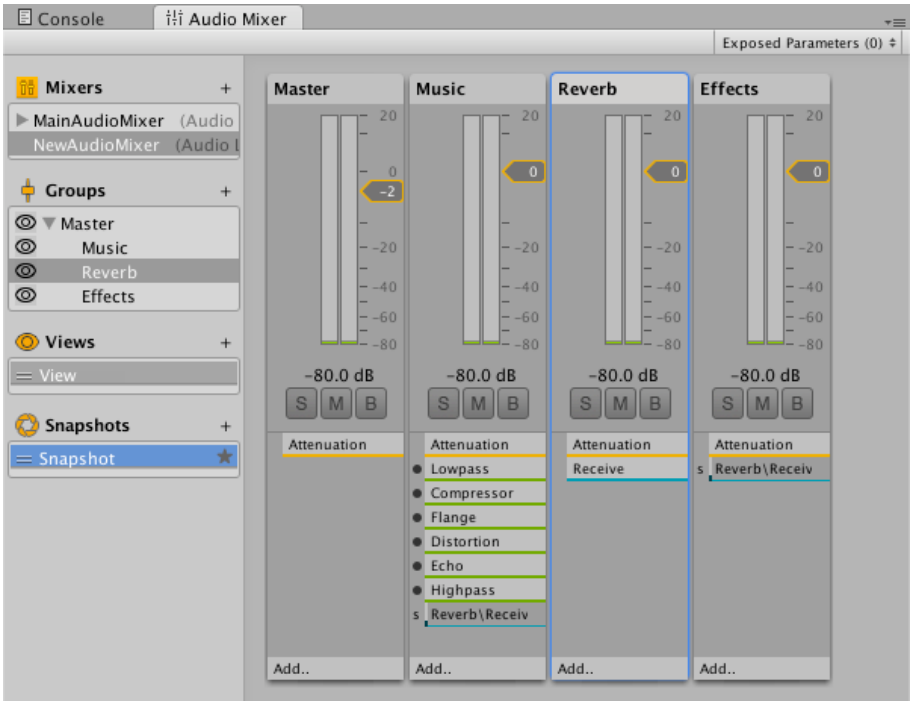




Slike 4.13. i 4.14. Area light [22]

4.5. Zvuk

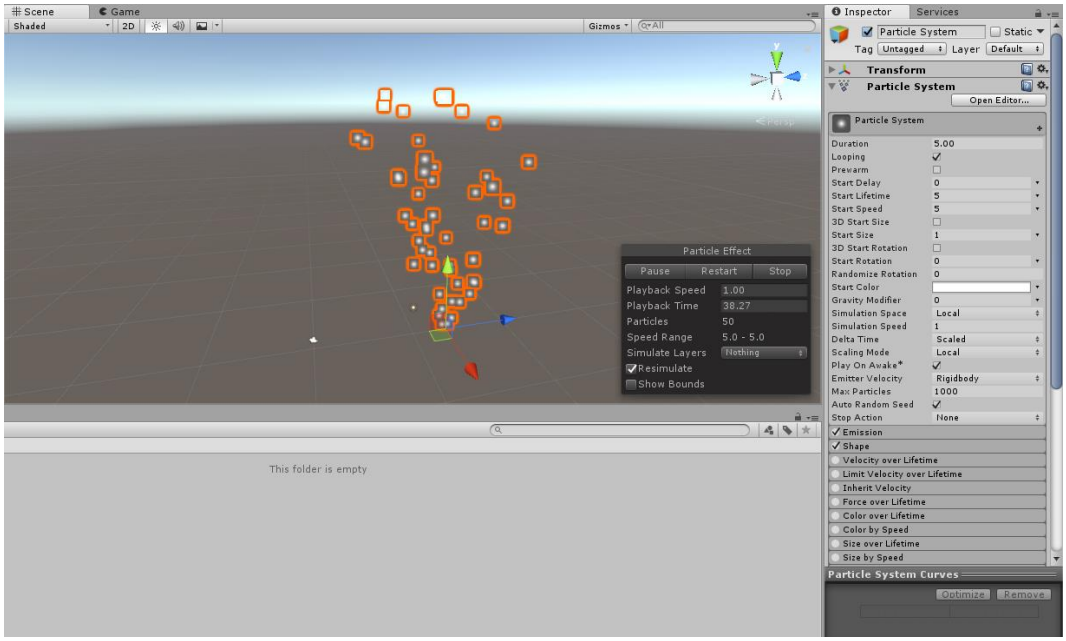
Igra bi bila nepotpuna bez zvuka, pozadinske glazbe ili zvučnih efekata. Unityev audio sustav je fleksibilan i snažan. Možemo uvesti većinu standardnih audio formata. Audio sustav ima sofisticirane značajke za reprodukciju zvuka u 3D prostoru te po izboru možemo dodati efekte poput jeke ili filtriranja. Unity također ima opciju snimanja zvuka preko bilo kojeg dostupnog mikrofona na računalu kojeg koristimo. U stvarnom životu zvukove emitiraju objekti i te zvukove čuju slušatelji. Način na koji čujemo zvuk ovisi o brojnim faktorima. Slušatelj može reći iz kojeg smjera zvuk dolazi i može dobiti osjećaj udaljenosti zvuka prema njegovoj jačini i kvaliteti. Izvor zvuka koji se brzo kreće (kao automobil koji prolazi ili bomba koja pada) će promijeniti visinu zvuka tijekom kretanja kao rezultat Dopplerovog efekta. Također, okolina će utjecati na način na koji se zvuk reflektira, tako da će glas unutar špilje imati odjek, dok na otvorenom neće. Da bi mogli simulirati efekt pozicije, Unity zahtijeva da zvuk potječe iz audio sourcea (izvor zvuka) koji je spojen na objekt. Zvukove koji se emitiraju prima Audio Listener (slušatelj zvuka) koji je spojen s nekim drugim objektom, najčešće glavnom kamerom. Unity potom može simulirati efekt udaljenosti i pozicije izvora zvuka od slušatelja i reproducirati zvuk prema slušatelju. Unity sadrži i Audio Mixer koji omogućuje miješanje različitih izvora zvuka i dodavanje efekata na njih. Svaki izvor zvuka ima opciju ponavljanja (engl. loop) kojom se mogu simulirati kratki zvukovi kao što je vatra, voda, vodopad i sl. Napredne postavke omogućuju funkcije prigušenja, prioriteta, postavljanja minimalne i maksimalne udaljenosti, itd. [14]



Slika 4.15. Audio Mixer [23]

4.6. Particle system

Unity ima impresivnu sposobnost renderiranja čestica (engl. particles) u igrama. Čestice predstavljaju dinamične prikaze magle, dima, vatre, iskre, prašine, tekućine, oblaka itd. One se mogu doživjeti kao više sitnih slika u pokretu. Za primjer se može uzeti vatromet. On se sastoji od stotine sitnih svjetala u pokretu. Emitiranje čestica počinje u jednoj točki te se nasumice kreću u različitim smjerovima, ovisno o postavkama koje su se podesile u Inspectoru. Kada čestica dođe do kraja zadanog vremena, uništava se. Čestice se mogu širiti prema raznim geometrijskim oblicima. Osim oblika širenja čestica, možemo podesiti i određene karakteristike kao što je brzina emitiranja, vijek trajanja, boja, veličina čestica, itd. [13]



Slika 4.16. Particle system – osnovne postavke

4.7. Grafičko korisničko sučelje (engl. GUI – Graphical user interface)

GUI (ili samo UI – engl. user interface) je zbirka vizualnih komponenti kao što su tekst, gumbi i slike koje olakšavaju interakciju korisnika sa softverom. Također, koriste se za davanje povratnih informacija igračima. U većini slučajeva, GUI igračima omogućuje interakciju s igrom. Bez GUIa, igrač ne bi imao nikakve vizualne naznake kako prelaziti igru. Možemo zaključiti da GUI ima dvije primarne svrhe: povratne informacije i kontrola. Povratne informacije su generirane od strane igre i upućene su igraču, dok igrač dobiva kontrolu nad igrom. [13]

Slika 4.17. UI igre u Unityu [24]

4.8. Skriptiranje

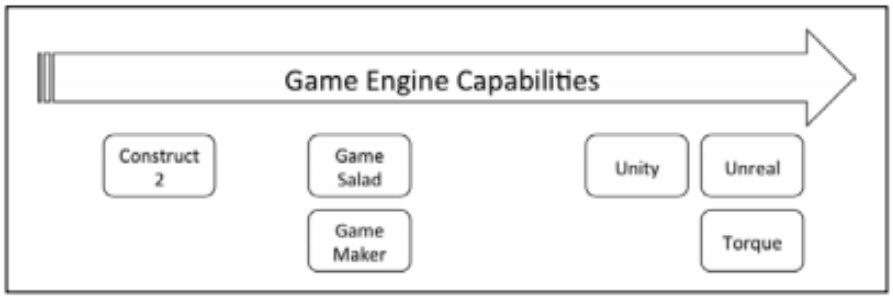
Skriptiranje je bitan sastojak u kreiranju svih igara. Čak i najjednostavnije igre trebaju skripte kako bi se mogle pokrenuti ili da bi se određene stvari u igri dogodile u određenom trenutku. Osim toga, skripte se mogu koristiti za izradu grafičkih efekata, mogu kontrolirati fizičko ponašanje objekta, itd. Skriptiranje je vještina koja zahtjeva određeno vrijeme i napor kako bi se usavršila. Ponašanje objekata kontroliraju komponente koje su dio njih. Iako komponente imaju široku primjenu, u jednom trenutku moramo otići korak naprijed kako bi implementirali vlastite značajke igranja. Unity omogućuje stvaranje vlastitih komponenti pomoću skripti koristeći jezik C#. C# je standardni jezik sličan C++u i Javi i koristi se u većini tutoriala, dokumentacija i primjera kodova te je zbog toga nadvladao jezike Boo i UnityScript (JavaScript) koji su se također prethodno koristili u Unityu. [14]

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

public class CameraFollow : MonoBehaviour {

    public Transform target;
    public float smoothing = 5f;

    Vector3 offset;

    // Use this for initialization
    void Start () {
        offset = transform.position - target.position;
    }

    // Update is called once per frame
    void FixedUpdate () {
        Vector3 targetCameraPosition = target.position + offset;

        transform.position = Vector3.Lerp(transform.position, targetCameraPosition, smoothing * Time.deltaTime);
    }
}
```

Slika 4.18. Primjer skripte

4.9. Konkurencija

Postoji nekoliko desetaka game enginea dostupnih za razvoj 2D i 3D igara. Pa zašto onda koristiti baš Unity? Unreal i CryEngine su među najboljim game engineima koji su dostupni te su ih upravo najveći studiji za razvoj igara koristili tijekom zadnjih godina. Modeli plaćanja i kompliciran način rada u njima bili su dovoljni da developeri indie igara i mali studiji za razvoj igara okrenu glavu na drugu stranu. Postoje i game enginei koji su laki za korištenje, poput Game Salad, Game Maker i Construct 2. U ovim engineima mogu se stvoriti jednostavne igre bez ijedne linije koda. To su sve enginei za razvoj 2D igara i zato su uskraćeni za mnogo mogućnosti. Sve te činjenice bi trebale staviti Unity u zlatnu sredinu između jednostavnog korištenja i imanja velikih mogućnosti za rad, ali Unity ipak više teži na stranu Unreal i CryEnginea. [13]



Slika 4.19. Game enginei poredani prema mogućnostima koje pružaju [25]

Sve prethodno navedeno bi trebalo ukazati da je Unity najbolji odabir za developere. Postoje i neki drugi razlozi koji idu u korist Unityu, kao npr. mogućnost programiranja sa C# jezikom, testiranje igara u zasebnom prikazu, sposobnost promjene igre za vrijeme igranja bez da te promjene utječu na prethodno spremljenu verziju (sve promjene napravljene za vrijeme igranja se poništavaju kada napustimo gameplay), itd. [13]

5. Maya - praktični dio

Većina modela koji su korišteni prilikom izrade igre napravljeni su u programu Autodesk Maya 2018. Svi modeli izrađeni su poligonalnim modeliranjem te je na većinu primijenjen materijal Lambert, dok je na svega jednom modelu korišten Blinn. Određeni modeli su teksturirani i izrađena im je UV mapa prilagođena teksturi koja je primijenjena. Nakon izrade svih modela, animiran je model vojnika koji je preuzet sa stranice Mixamo. Vojnik je već imao postojeći ljudski kostur što je uvelike olakšalo animiranje jer je korišten HumanIK. Svi modeli i animacije eksportani su u formatu .fbx jer na taj način se eksportiraju i pripadajući materijali i teksture skupa s modelima.

5.1. Izrada Scene

Za scenu igre prvo je izmodeliran teren (podloga) po kojoj će se glavni lik kretati pomoću alata Polygon Cube, Polygon Sphere i Polygon Cylinder. Osnovni oblici navedenih alata po potrebi su modificirani manipulacijom njihovim vertexima, facevima i edgevima. Također, na nekim objektima korištena je i opcija Edit Mesh -> Extrude. Od terena se oduzeo drugi Polygon Cube kako bi nastala udubina prema kojoj je kasnije napravljen vodopad tako da je primijenjena opcija Mesh -> Booleans -> Difference. Odabirom travnjaka na planini i pritiskom broja tri na tipkovnici dobije se Smooth mesh preview (zaglađeni pravilniji oblik) modela. Kako bi model ostao tako fino zaglađen potrebno ga je pretvoriti nazad u poligon (Modify -> Convert -> Smooth Mesh Preview to Polygons). Na teren je primijenjeno osam različitih materijala. Svi materijali su bili Lambert, samo drugih boja i postavki. Na vodopadu se na materijalu smanjio Transparency (prozirnost) kako bi se dobio dojam vode.

Potom je slijedila izrada drveća i kamenja. Drveće je izrađeno uz pomoć alata Polygon Sphere, Polygon Cylinder i Polygon Cone te su modificirani i extrudani po potrebi. Na svako drvo je primijenjen materijal Lambert, također uz male izmjene boja kako scena ne bi bila jednolična. Kamenje je napravljeno modificiranjem i extrudanjem kugle (alat Polygon Sphere).

Kako bi scena izgledala življe, dodana je logorska vatra i baklje. Oboje je napravljeno vrlo jednostavno; koristeći alate Polygon Cylinder (za drvo) i Polygon Sphere (za vatru). Za drvo je

korišten već postojeći materijal Lambert koji je korišten i na prethodno napravljenom drveću, a za vatru je korišten Lambert materijal narančaste boje sa smanjenim Transparencyem.

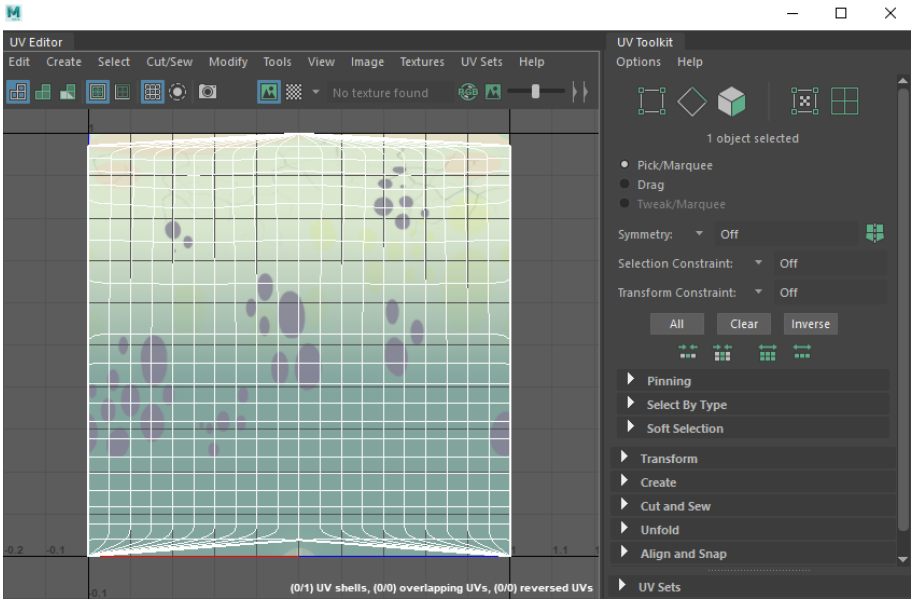
Za kraj, na scenu je dodan dvorac koji se može preuzeti besplatno na stranici Sketchfab (sketchfab.com) te je prema vlastitim željama preuređen (promijenjen materijal i uklonjeni neki dijelovi koji nisu bili potrebni na sceni). Dvorac je djelo dvojice rođaka koji se na stranici Sketchfab vode pod imenom BlackSpire. (link preuzetog dvorca: https://sketchfab.com/models/83bd872a94d944babcbd1aaf6febfaf1)

Slika 5.1. Prikaz renderirane scene u Mayi

Slika 5.2. Prikaz renderirane scene u Mayi 2

5.2. Izrada jaja

Posebno su izrađena jaja koja će se u igri morati skupljati kako bi se igra završila. Prilikom izrade korišten je alat Polygon Sphere. Dobivena kugla je modificirana tako da se manipuliralo s njenim facevima. Kad je dobiven željeni oblik jajeta, ono je bilo poprilično low poly (imalo je mali broj poligona). Primijenjen je Smooth Mesh Preview na jaje i konvertirano je opet u poligon. Kao materijal korišten je Lambert na koji je primijenjena željena tekstura. Budući da mesh (mreža) modela nije odgovarala teksturi, bilo je potrebno pozabaviti se s namještanjem mesha u UV Editoru. Kad je dobiven konačan pravilan izgled jajeta, jaje je kopirano tri puta i na novonastala tri jajeta stavljena je slična tekstura, samo druge boje.

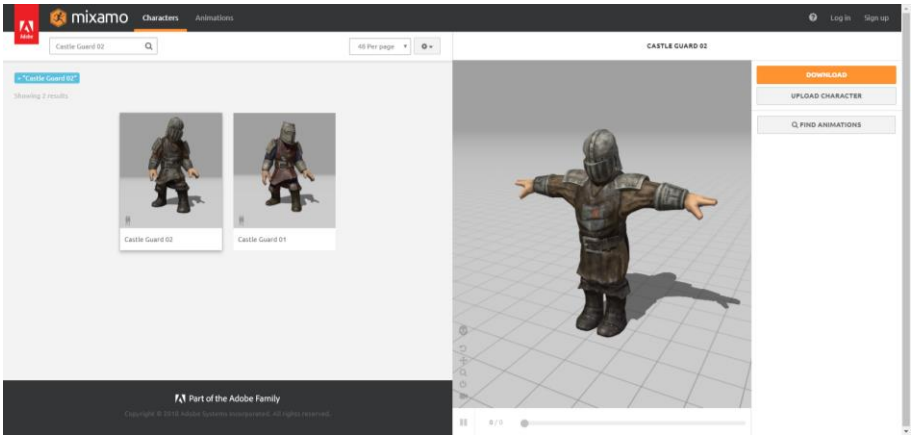


Slika 5.3. UV mapa za teksturu jajeta

Slika 5.4. Prikaz renderiranih jaja u Mayi

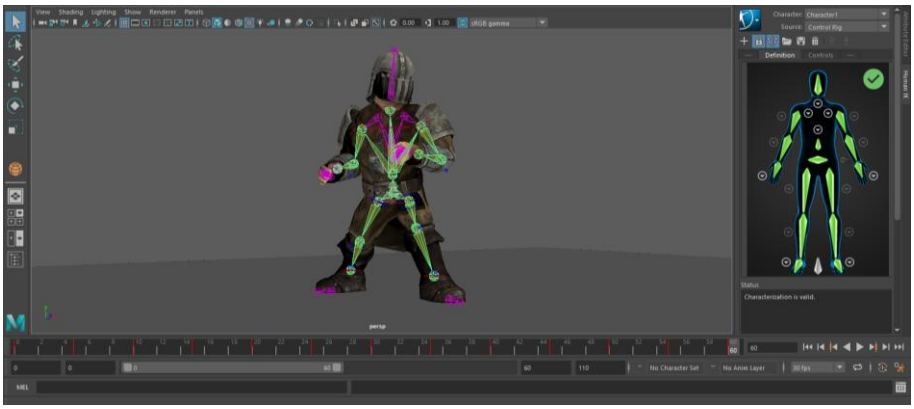
5.3. Animacija vojnika – neprijatelj

Kao što je navedeno, vojnik je besplatno preuzet sa stranice Mixamo (mixamo.com).



Slika 5.5. Vojnik na stranici Mixamo

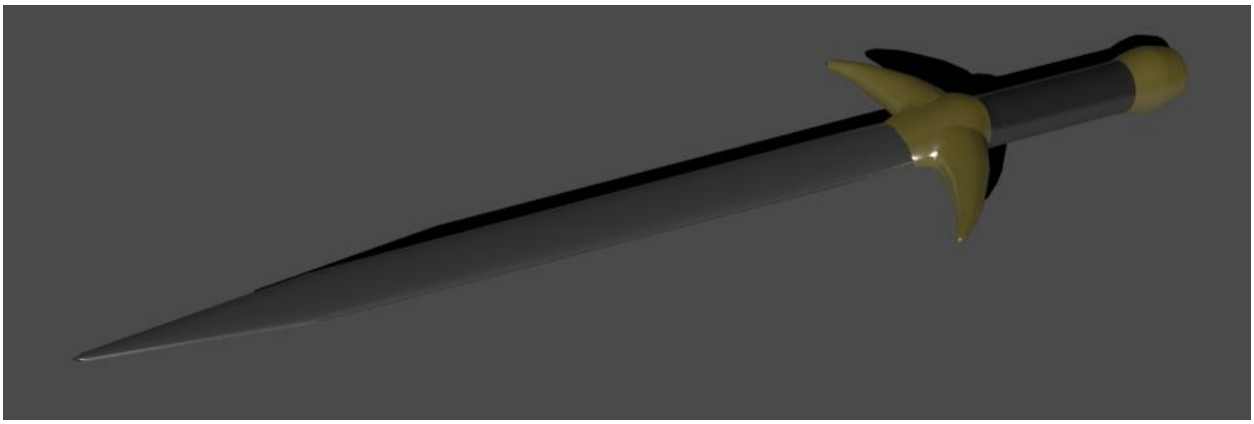
Pošto je vojnik već imao definiran ljudski kostur, njegove kosti su se mogle hijerarhijski povezati u HumanIKu. Princip animiranja je bio isti kao što je prethodno opisano; za određene dijelove tijela postavljeni su ključni okviri te je na taj način napravljena animacija za idle (besposlen, stoji na mjestu) i napad. Da bi animacija mogla biti pravilno eksportirana iz Maye, potrebno je bakeati sve okvire između onih ključnih tako da i oni postanu ključni okviri (Edit -> Keys -> Bake Simulation).



Slika 5.6. Definirana hijerarhija kostura vojnika u HumanIK, Maya

5.4. Izrada mača

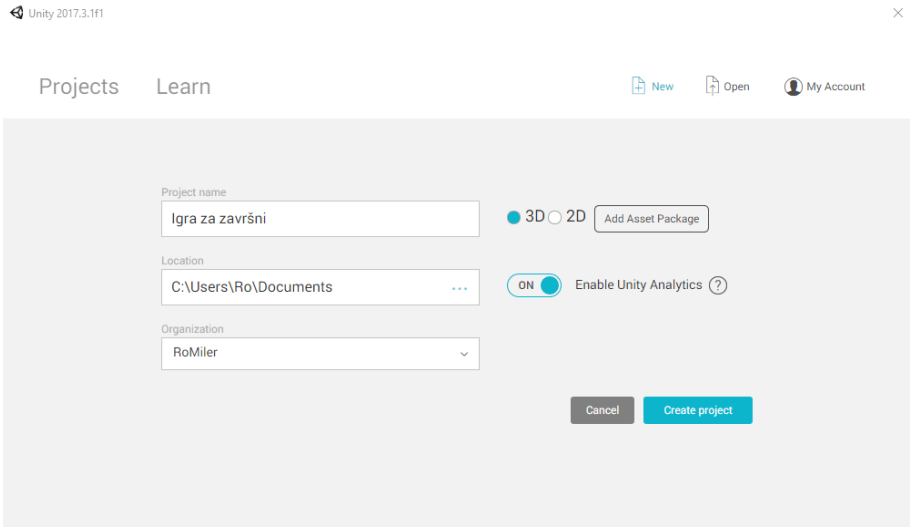
Pošto je vojnik preuzet bez ikakvog oružja, izmodeliran je mač uz pomoć alata Polygon Cube i Polygon Cylinder. Na mač je primijenjen tri puta materijal Blinn (sjajni materijal), svaki za drugačiju boju, sivu (srebrenu), tamniju sivu (tamniju srebrenu) i žutu (zlatnu). Mač se naknadno veže za kostur vojnika u Unityu.



Slika 5.7. Prikaz renderiranog mača u Mayi

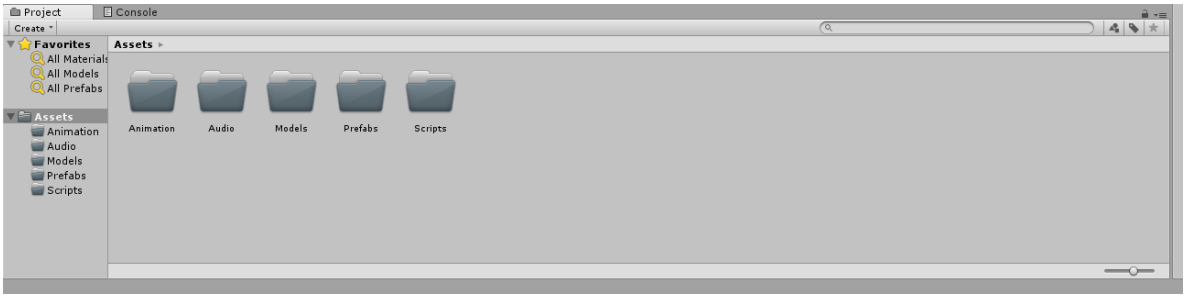
6. Unity - praktični dio

Prilikom pokretanja Unitya prvo se treba kreirati projekt. Nudi se mogućnost kreiranja 2D i 3D projekta te dodavanje paketa dodataka (Asset Package). Projekt se može imenovati, kao što se može odabrati i lokacija pohranjivanja projekta.



Slika 6.1. Kreiranje projekta u Unityu

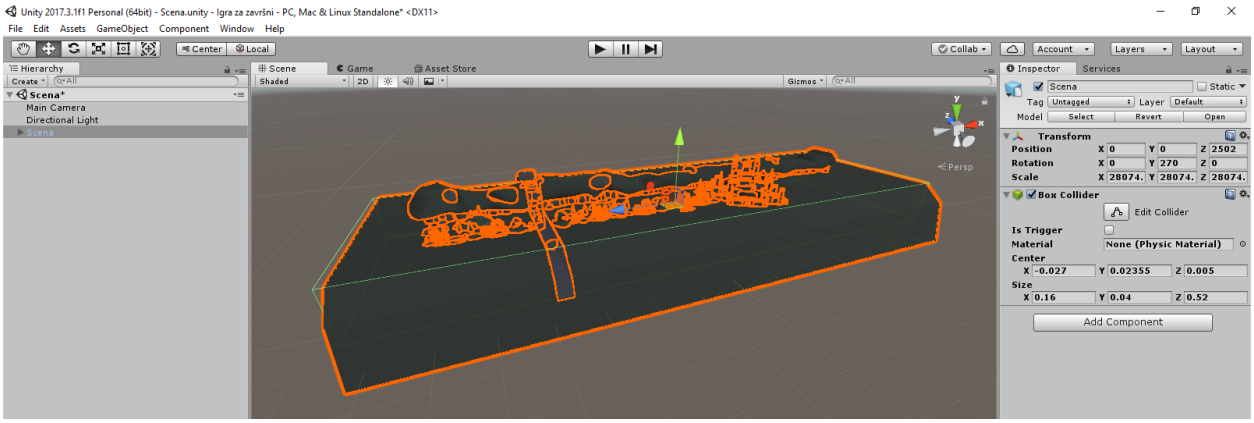
Kada je Unity učitao novonastali projekt, u prozoru Project su se kreirale neke osnovne mape (engl. folders) u koje će se razvrstati svi modeli, materijali, skripte, zvukovi, itd., koji će se koristiti za stvaranje igre.



Slika 6.2. Početni folderi u prozoru Project

6.1. Postavljanje scene

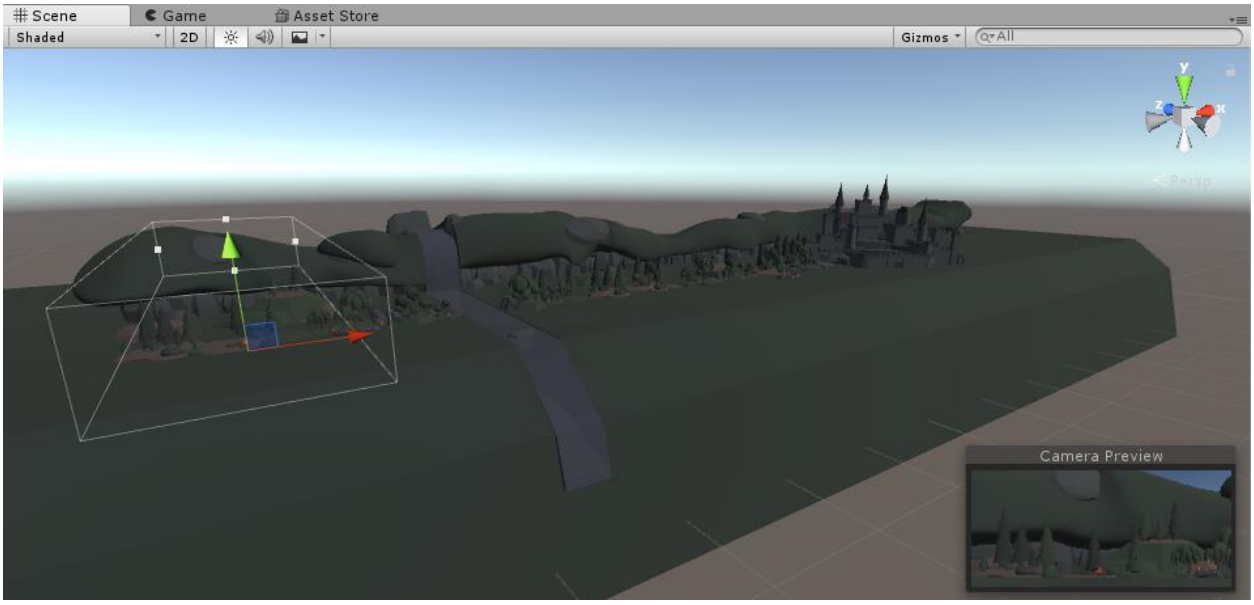
Na scenu je ubačen teren koji je napravljen u Mayi. Pošto je iz Maye teren eksportiran kao .fbx, zadržali su se svi materijali i boje tako da nije bilo potrebno ništa od materijala naknadno ubacivati. Na teren je dodana komponenta Box Collider (u prozoru Inspector Add Component -> Physics -> Box Collider) koja omogućuje glavnom liku, neprijateljima ili nekim drugim objektima da stoje na zemlji, odnosno da se mogu kretati po terenu, inače bi samo propali kroz teren i nastavili beskonačno padati. Tijekom daljnjeg kreiranja igre po potrebi će se mijenjati Collider.



Slika 6.3. Postavljena scena

6.2. Postavke kamere

U prozoru Hierarchy odabrana je kamera (Main Camera), na što su se otvorile postavke kamere u prozoru Inspector. Kamera je postavljena tako da joj projekcija bude ortografska (Orthographic), što znači da će joj pogled bit usmjeren na scenu tako da dobijemo dojam da se radi o 2D igri, tj. 2.5D igri, s obzirom da su modeli trodimenzionalni, ali je projekcija dvodimenzionalna. Također, dodana je skripta 'CameraFollow' (Add Component -> New Script) na kameru u kojoj će se u programu Visual Studio 2017 isprogramirati da kamera prati glavnog lika. Nakon napisanog koda u skripti, pojavile su se dodatne postavke koje možemo modificirati, konkretno, možemo postaviti Smoothing kamere (koliko glatko se kreće skupa s likom) te Target, kućicu u koju ubacujemo glavnog lika kojeg će kamera pratiti.



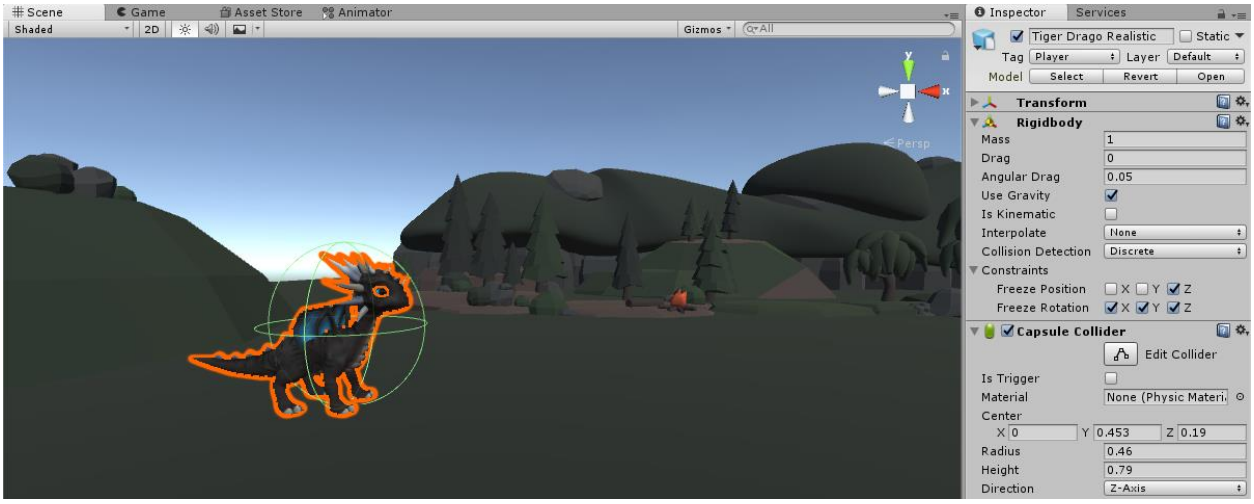
Slika 6.4. Postavljena kamera s ortografskom projekcijom

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

public class CameraFollow : MonoBehaviour {

    public Transform target;
    public float smoothing = 5f;

    Vector3 offset;

    // Use this for initialization
    void Start () {
        offset = transform.position - target.position;
    }

    // Update is called once per frame
    void FixedUpdate () {
        Vector3 targetCameraPosition = target.position + offset;

        transform.position = Vector3.Lerp(transform.position, targetCameraPosition, smoothing * Time.deltaTime);
    }
}
```

Slika 6.5. Skripta 'CameraFollow'

6.3. Postavljanje glavnog lika – zmaj

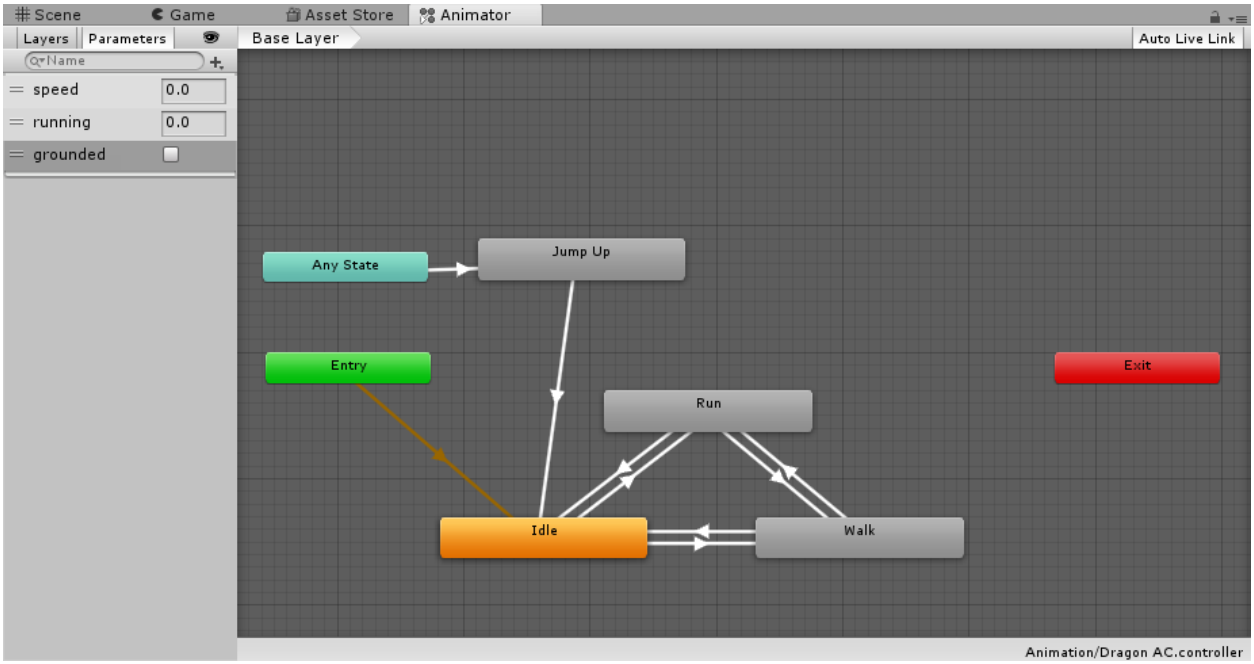
Za glavnog lika igre odabran je zmaj. Zmaj je kupljen na Unity Asset Storeu u paketu s još nekoliko različitih modela zmajeva, materijala za iste, animacijama, zvukovima, itd. (link paketa na Unity asset storeu: https://assetstore.unity.com/packages/3d/characters/creatures/little-dragons-tiger-64248). Na zmaja su dodane komponente Rigidbody i Capsule Collider. Pod Rigidbody je zamrznuta rotacija za sve tri osi i pozicija za z-os tako da se može kretati samo lijevo-desno (x-os) i gore-dolje (y-os). Capsule Collider služi kao zmajevo ograničenje i hitbox, u smislu da ga taj collider drži na površini terena jer se sudara sa colliderom terena, da se spriječi prolazak dalje ako se sudari s nekim drugim objektom koji također ima collider, da može pokupiti objekte koji će se moći skupljati te da označava granicu koju neprijatelj mora dirati da bi zmaj primio damage.



Slika 6.6. Postavljen glavni lik (zmaj) na scenu

Nakon toga je zmaju dodana komponenta Animator i pod polje Controller je ubačen Animator Controller 'Dragon AC' koji je posebno napravljen u prozoru Project (Create -> Animator Controller). U Animator su ubačene animacije Idle, Walk, Run i Jump Up (princip drag and drop iz prozora Project) kojima su dodani prijelazi (desni klik na animaciju -> Make Transition). Za prijelaze je bilo potrebno napraviti float parametre speed i running koji su određivali vrijednosti prema kojima se mijenjaju stanja, odnosno animacije. Također, bool parametar grounded je bio potreban za prijelaze u i iz animacije skoka (Jump up). Animacija Jump up je namještena tako da se može iz bilo kojeg stanja skočiti (prijelazu je dodan parametar grounded 'false'), ali uvijek odlazi

u idle nakon što dotakne tlo (grounded 'true'). Kako bi to sve funkcioniralo, promijenjen je Layer terena iz Default u Ground (novi Layer) i dodan je novi objekt (GameObject) na zmaja (označen je zmaj u Hierarchy prozoru te se odabire Create -> Create Empty Child) koji se zove checkGroundLocation. Taj objekt provjerava dira li zmaj pod.



Slika 6.7. Animator u kojem se nalaze animacije zmaja

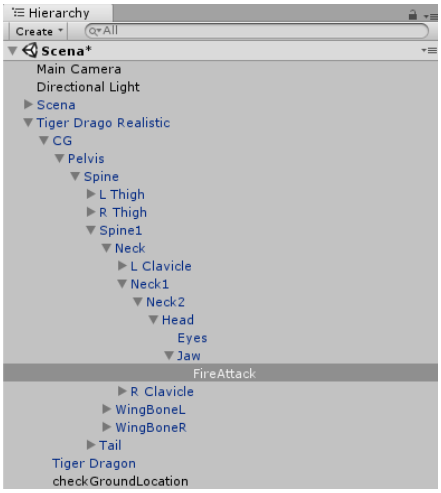
Nakon podešavanja svih animacija i njihovih prijelaza, dodana je skripta 'DragonController' na zmaja u kojoj je isprogramirano kretanje lijevo-desno, rotiranje prilikom mijenjanja smjera, kada prelazi iz jedne animacije u drugu, itd.

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

public class DragonController : MonoBehaviour {

    //movement variables
    public float walkSpeed;
    public float runSpeed;

    Rigidbody myRB;
    Animator myAnimator;

    bool facingRight;

    // Use this for initialization
    void Start () {
        myRB = GetComponent<Rigidbody>();
        myAnimator = GetComponent<Animator>();
        facingRight = true;
    }


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

    }

    void FixedUpdate(){

        float move = Input.GetAxis("Horizontal");
        myAnimator.SetFloat("speed", Mathf.Abs(move));

        float running = Input.GetAxisRaw("Fire3");
        myAnimator.SetFloat("running", running);

        if (running > 0){
            myRB.velocity = new Vector3(move * runSpeed, myRB.velocity.y, 0);
        }

        else{
            myRB.velocity = new Vector3(move * walkSpeed, myRB.velocity.y, 0);
        }

        if (move>0 && !facingRight) Flip();
        else if (move<0 && facingRight) Flip();
    }

    void Flip(){
        facingRight = !facingRight;
        Vector3 theScale = transform.localScale;
        theScale.z *= -1;
        transform.localScale = theScale;
    }
}
```

```
    //for jumping

    bool grounded = false;
    Collider[] groundCollisions;
    float groundCheckRadius = 0.2f;
    public LayerMask groundLayer;
    public Transform groundCheck;
    public float jumpHeight;
```

```
void FixedUpdate(){

    if(grounded && Input.GetAxis("Jump")>0){
        grounded = false;
        myAnimator.SetBool("grounded", grounded);
        myRB.AddForce(new Vector3(0, jumpHeight, 0));
    }

    groundCollisions = Physics.OverlapSphere(groundCheck.position, groundCheckRadius, groundLayer);
    if (groundCollisions.Length > 0) grounded = true;
    else grounded = false;

    myAnimator.SetBool("grounded", grounded);
    myAnimator.SetFloat("verticalSpeed", myRB.velocity.y);
```

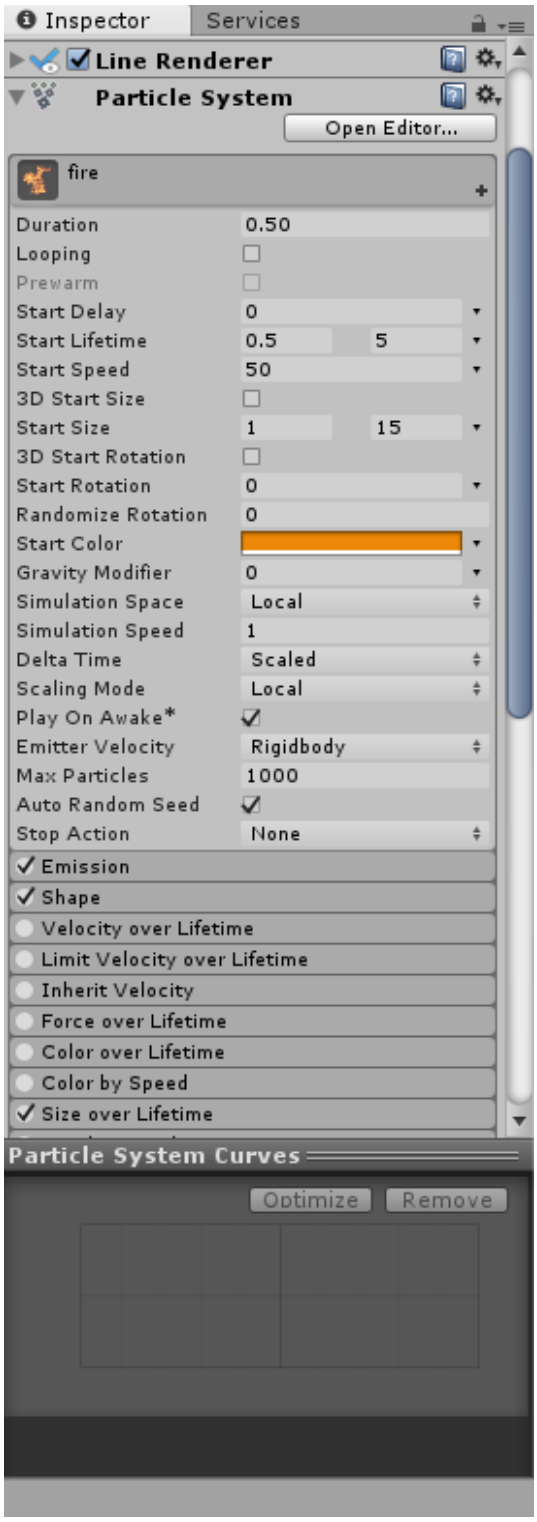
Slika 6.8. Skripta 'DragonController'

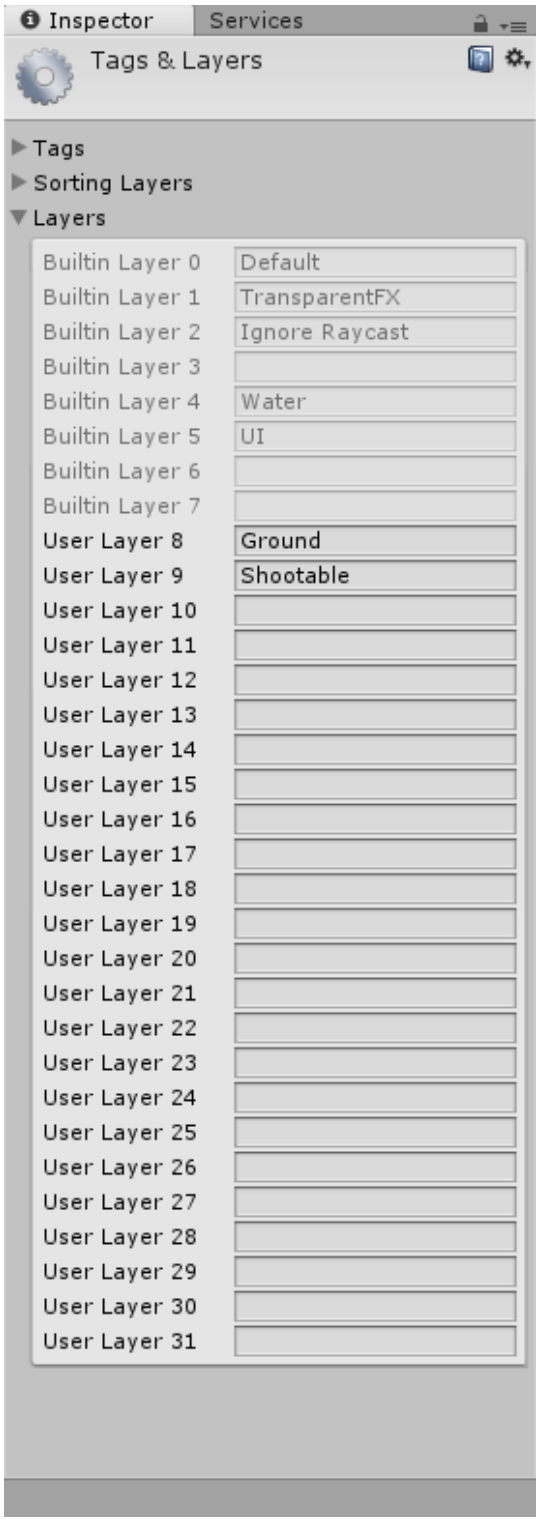


Slika 6.9. Objekt FireAttack povezan s ustima na zmajevom kosturu

Kao zmajev napad odabrano je puhanje vatre. Prvo se u prozoru Hierarchy pronađe Jaw (usta, vilica) pod kosturom zmaja te se na Jaw dodaje novi GameObject (Create -> Create Empty Child) koji je nazvan FireAttack. Objekt je smješten kod usta baš zato što je u pitanju puhanje vatre.

Nakon toga je napravljen novi zasebni GameObject (Create -> Create Empty) i nazvan je fire. Objektu fire se u Inspectoru dodaje nova komponenta Line Renderer (Add Component -> Effects -> Line Renderer). Ova komponenta je efekt koji će predstavljati zraku, odnosno liniju koja će određivati smjer i dužinu vatre. Zatim se dodaje komponenta Particle System (Add Component -> Effects -> Particle System). Za Particle System je napravljen posebni materijal. Za materijal je korištena slika plamena tako da kad se čestice raspršе, one će zapravo tvoriti jednu veliku vatru. Za stvari koje će se moći pogoditi i kojima će se moći raditi damage trebalo je napraviti novi Layer Shootable.





Slike 6.10. i 6.11. Postavke particle systema (vatre) i prikaz Layera (novi Layer Shootable)

Na taj isti objekt fire dodane su i dvije skripte, 'shootFire' i 'DestroyMe'. Jedna će nam dati da u Inspectoru unesemo koliki doseg će vatra imati te koliki damage će raditi neprijateljima, dok druga skripta daje mogućnost da unesemo koliki će biti vijek trajanja te vatre, odnosno nakon kojeg vremena će se uništiti, jer nebi imalo smisla da ostane zauvijek na sceni nakon napada.

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

public class shootFire : MonoBehaviour
{

    public float range = 150f;
    public float damage = 5f;

    Ray shootRay;
    RaycastHit shootHit;
    int shootableMask;
    LineRenderer fireLine;

    // Use this for initialization
    void Awake()
    {
        shootableMask = LayerMask.GetMask("Shootable");
        fireLine = GetComponent<LineRenderer>();

        shootRay.origin = transform.position;
        shootRay.direction = transform.forward;
        fireLine.SetPosition(0, transform.position);

        if (Physics.Raycast(shootRay, out shootHit, range, shootableMask))
        {
            //hit an enemy goes here
            fireLine.SetPosition(1, shootHit.point);
        }
        else fireLine.SetPosition(1, shootRay.origin + shootRay.direction * range);

    }

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

    }
}
```

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

public class DestroyMe : MonoBehaviour {

    public float aliveTime;

    // Use this for initialization
    void Awake () {
        Destroy(gameObject, aliveTime);
    }

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

    }
}
```

Slika 6.12. Skripta 'shootFire'

Ponovnim odabirom objekta fireAttack koji je dio kostura zmaja dodana je komponenta Audio Source. Ta komponenta sadrži zvuk koji će se čuti tijekom napadanja, no pošto je isključena opcija 'Play on Awake' kako se zvuk ne bi čuo odmah prilikom pokretanja igre, potrebno je bilo isprogramirati da se taj zvuk triggera samo kad napadnemo. Dodana je skripta koja će omogućiti upravo tom zvuku da se pokrene svaki put kad napadnemo i da odredimo koliki je delay (kašnjenje) između napada. Također, nastalo je polje Projectile koje označava ono što će se ispaliti nakon što pritisnemo tipku za napad. U tom slučaju, pod to polje se stavi prethodno složen objekt fire.

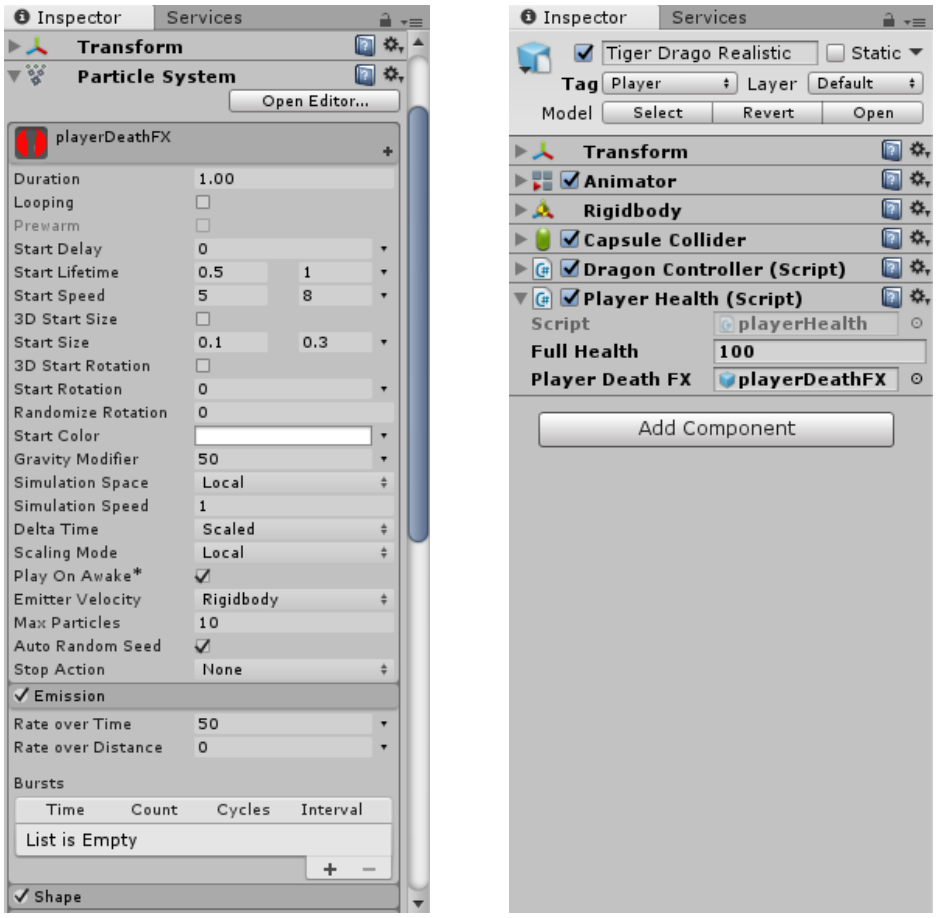
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class fireAttack : MonoBehaviour
{

    public float timeBetweenFires = 0.15f;
    public GameObject projectile;

    float nextFire;

    //audio
    public AudioClip fireSound;
    public float Volume;
    AudioSource fireAS;

    // Use this for initialization
    void Awake()
    {
        nextFire = 0f;

        //audio
        fireAS = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {
        DragonController myPlayer = transform.root.GetComponent<DragonController>();

        if (Input.GetAxisRaw("Fire1") > 0 && nextFire < Time.time)
        {
            nextFire = Time.time + timeBetweenFires;
            Vector3 rot;
            if (myPlayer.GetFacing() == -1f)
            {
                rot = new Vector3(0, -90, 0);
            }
            else rot = new Vector3(0, 90, 0);

            if (Input.GetButtonDown("Fire1"))
            {
                Instantiate(projectile, transform.position, Quaternion.Euler(rot));

                //audio
                fireAS.clip = fireSound;
                fireAS.PlayOneShot(fireSound, Volume); ;
            }

        }
    }
}
```

Slika 6.13. Skripta 'fireAttack'

Dodana je još jedna skripta na zmaja. Skripta se zove playerHealth i ona određuje koliko će zmaj imati healtha i kada health padne na nulu, zmaj umire. Nakon toga se zmaju promijeni u Inspectoru Tag iz Untagged u novi tag Player tako da se skripta može referencirati na zmaja preko tog taga. U Hierarchy prozoru dodaje se novi Particle System (Create -> Effects -> Particle System) koji je nazvan playerDeathFX. Na ovaj Particle System dodan je materijal crvene boje i namještene su postavke tako da kada zmaj umre, čestice se raspršе kao kapljice krvi. Ovaj efekt se dodaje na zmaja pod polje Player Death FX koje je nastalo nakon programiranja u skripti playerHealth.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class playerHealth : MonoBehaviour {
6
7       public float fullHealth;
8       float currentHealth;
9
10      public GameObject playerDeathFX;
11
12      // Use this for initialization
13      void Start () {
14          currentHealth = fullHealth;
15      }
16
17      // Update is called once per frame
18      void Update () {
19
20      }
21
22      public void addDamage(float damage)
23      {
24          currentHealth -= damage;
25          if(currentHealth <= 0)
26          {
27              makeDead();
28          }
29      }
30
31      public void makeDead()
32      {
33          Instantiate(playerDeathFX, transform.position, Quaternion.Euler(new Vector3(-90, 0, 0)));
34          Destroy(gameObject);
35      }
36  }
37
```

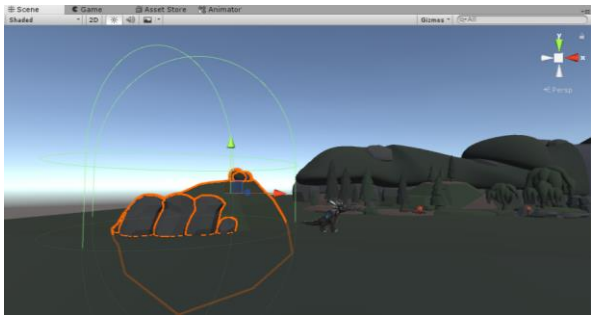
Slika 6.14. Skripta 'playerHealth'

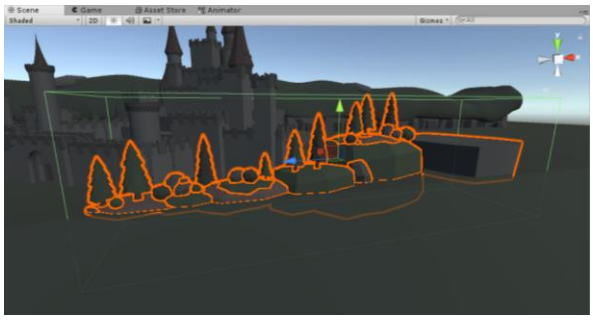


Slike 6.15. i 6.16. Postavke particle systema (krv) i nove postavke zmaja nakon ubačene skripte playerHealth

6.4. Update scene

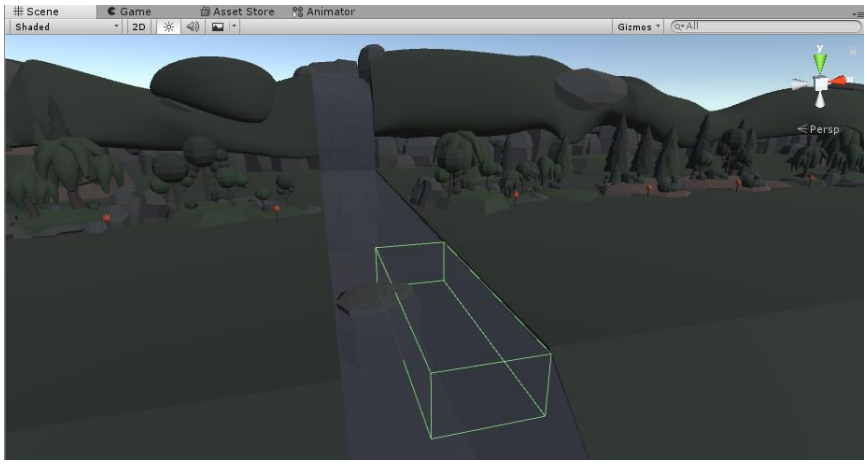
Na sceni je od već postojećih modela drveća i kamenja složen 3D obstacle (zapreka) koja će blokirati put zmaju i na taj način zmaj neće moći pasti s terena te je definiran početak i kraj. Na te zapreke su stavljeni samo Capsule i Box Collideri.

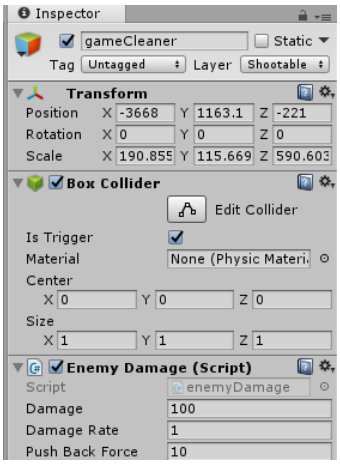




Slike 6.17. i 6.18. 3D obstacles (zapreke)

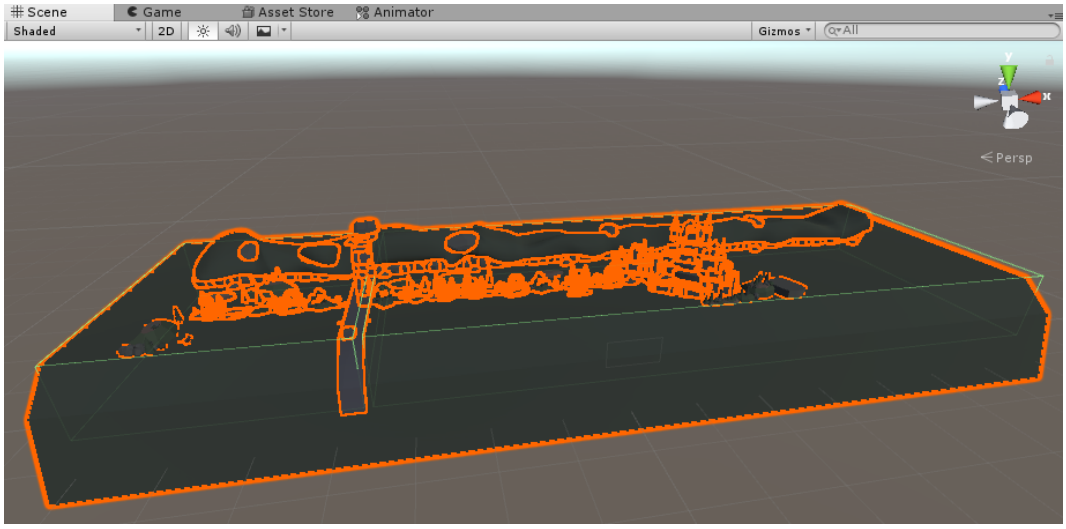
Kod vodopada je pomaknut kamen u lijevu stranu i u vodu je dodan objekt gameCleaner koji se sastoji od Box Collidera i na sebi ima skriptu enemyDamage. Ta skripta je napravljena za neprijatelje koji će se kasnije dodati na scenu, ali u ovom slučaju je korištena da objekt radi 100 damagea zmaju – koliko je zmajev maksimalan health, što znači da će zmaj odmah umrijeti ukoliko padne u vodu.





Slike 6.19. i 6.20. Prikaz pozicije i postavke objekta 'gameCleaner'

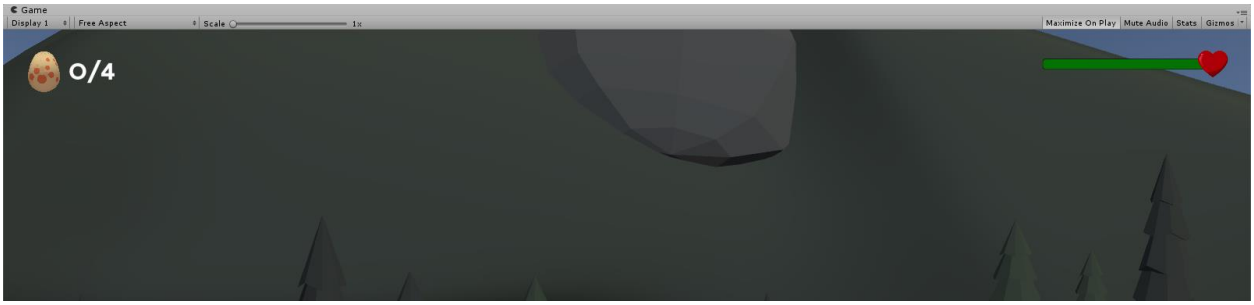
Na teren su stavljena ipak dva Box Collidera, tako da na vodopadu ne postoji niti jedan collider koji će omogućiti da se po njemu hoda, a i u tom slučaju gameCleaner ne bi funkcionirao jer bi samo prešli preko njega



Slika 6.21. Updateana scena

6.5. Korisničko sučelje - UI

U Hierarchy prozoru se kreira Canvas (platno) (Create -> UI -> Canvas) na koje su postavljeni elementi tako da dobijemo korisničko sučelje igre koje će igraču davati povratnu informaciju o tome koliko još zmaj ima healtha i koliko je jaja skupio. Canvasu se dodaje Slider u gornjem desnom kutu kojem je promijenjen izgled tako da zeleni dio označava health, a crveni dio koliko healtha nedostaje. Na Canvas se dodaje i slika playerDamageIndicatorImage (desni klik na canvas -> UI -> Image). Slika je transparentna u sredini s crvenim rubom i kada nešto radi damage zmaju, ta slika blica na ekranu i nakon nekog vremena opet nestane. U gornji lijevi kut dodana je slika jajeta i pokraj nje tekst koji prikazuje koliko je jaja skupljeno.



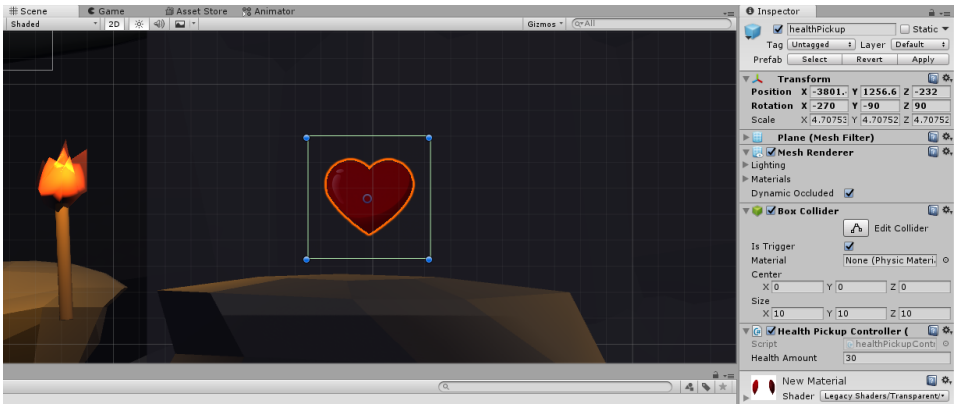
Slika 6.22. Prikaz izgleda elemenata korisničkog sučelja igre

6.6. Power up – srce

U prozoru Hierarchy odabere se Create -> 3D Object -> Plane. Nakon što se na sceni postavi plane na željeno mjesto, dodaje mu se prethodno namješten materijal koji se sastojao od transparentne slike srceta (isto srce kao i na slideru canvasa). Planeu se dodaje Box Collider na kojem se označi 'Is Trigger' što znači da će se nešto dogoditi kada se neki drugi Collider sudari s ovim. U ovom slučaju je napravljena skripta healthPickupController u kojoj je isprogramirano da kada se collider zmaja sudari s colliderom ovog planea, zmaju će narasti health. Ukoliko zmaj već ima maksimalan health on će i dalje moći pokupiti srce, ali ništa se s njegovim healthom neće dogoditi.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class healthPickupController : MonoBehaviour {
6
7       public float healthAmount;
8
9       // Use this for initialization
10      void Start () {
11
12      }
13
14      // Update is called once per frame
15      void Update () {
16
17      }
18
19      void OnTriggerEnter(Collider other)
20      {
21          if(other.tag == "Player")
22          {
23              other.GetComponent<playerHealth>().addHealth(healthAmount);
24              Destroy(transform.root.gameObject);
25          }
26      }
27  }
28
```

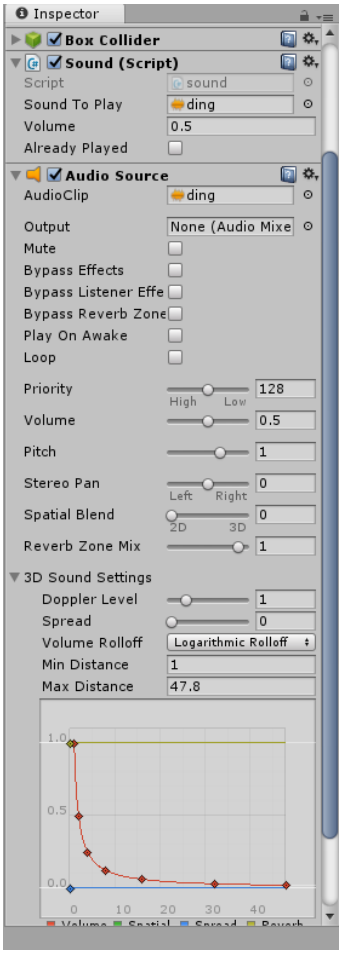
Slika 6.23. Skripta 'healthPickupController'



Slika 6.24. Postavke i izgled srca koje daje health

Na scenu se dodaje novi GameObject naziva holder. Taj objekt ima Box Collider kojem je, također, uključen Is Trigger, a u ovom slučaju kada se triggera, čuje se određeni zvuk. Smješten je na sceni točno na srce, tako da kad se srce pokupi, taj zvuk se triggera. Na objekt se dodaje Audio Source koji sadrži Audio Clip, tj. zvuk koji će se triggerati. Da bi to sve funkcioniralo, napravljena je skripta sound koja poziva taj Audio Source kada se triggera collider. Taj isti holder će se iskoristiti i za skupljanje jaja koja će ostavljati neprijatelji kada ih se ubije.

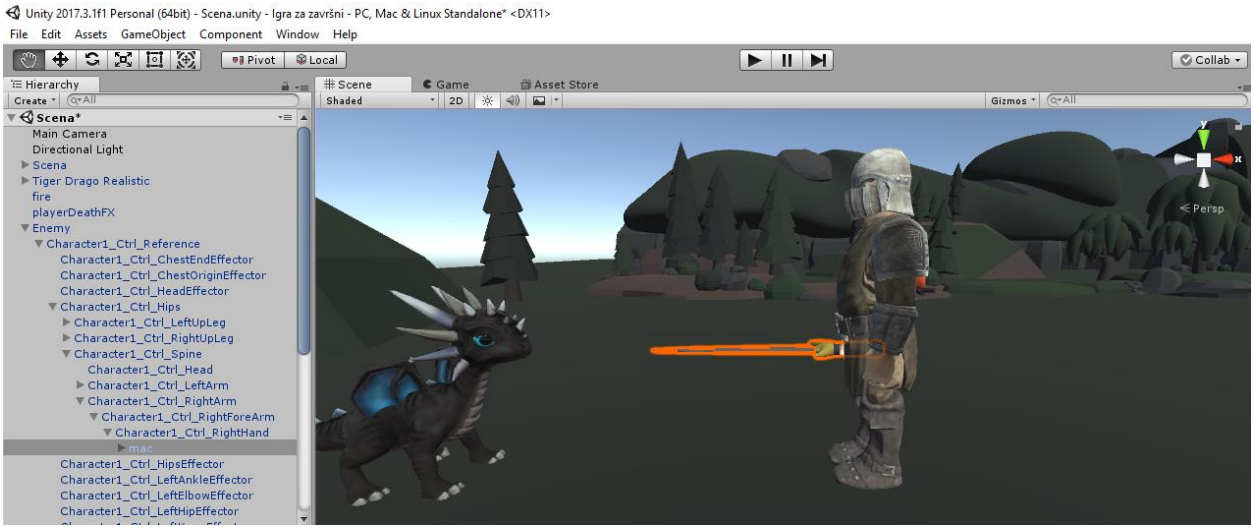
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class sound : MonoBehaviour {
    public AudioClip soundToPlay;
    public float Volume;
    AudioSource audio;
    public bool alreadyPlayed = false;

    // Use this for initialization
    void Start () {
        audio = GetComponent<AudioSource>();
    }

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

    }

    void OnTriggerEnter(Collider other)
    {
        if (!alreadyPlayed)
        {
            audio.PlayOneShot(soundToPlay, Volume);
            alreadyPlayed = true;
        }
    }


}
```



Slike 6.25. i 6.26. Skripta 'sound' i postavke zvuka na objektu (holder)

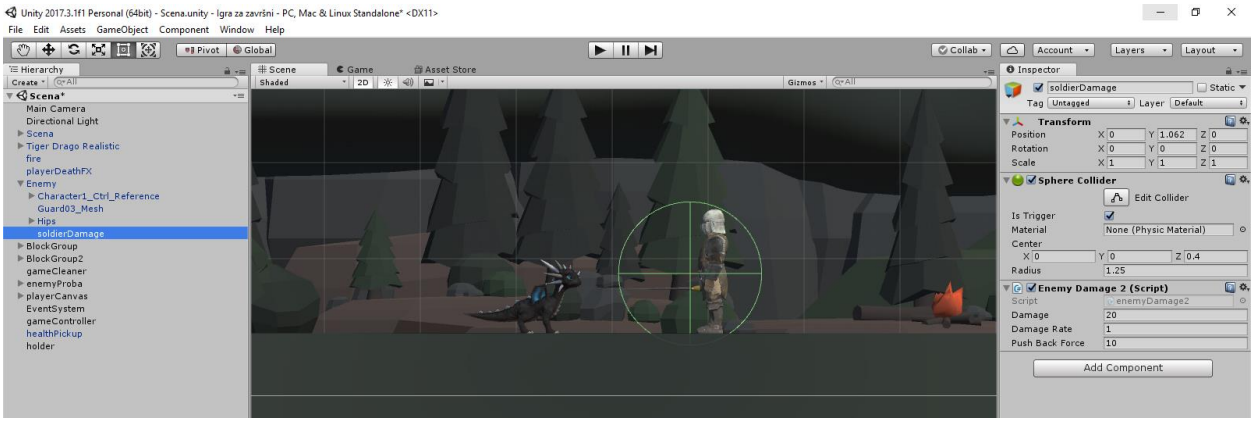
6.7. Neprijatelj

Na scenu je stavljen model vojnika i dodana mu je komponenta Rigidbody na kojoj je zamrznuta rotacija i pozicija kao i na zmaju. Zatim mu se dodaje Capsule Collider koji će mu omogućiti da stoji na terenu bez propadanja. Layer mu je promijenjen u Shootable jer je to objekt kojem zmaj može raditi damage. Također, dodan mu je mač prethodno napravljen u Mayi i povezan je s njegovim kosturom tako da je ubačen pod čvor (engl. node) desne ruke.



Slika 6.27. Mač povezan s kosturom vojnika (desnom rukom)

Nakon toga se vojniku dodaje Empty Child GameObject koji je nazvan soldierDamage. Ovom objektu je dodan Sphere Collider koji se može triggerati i skripta enemyDamage. Svrha ovog objekta je da kada se sudari s njegovim colliderom, vojnik zmaju radi damage.



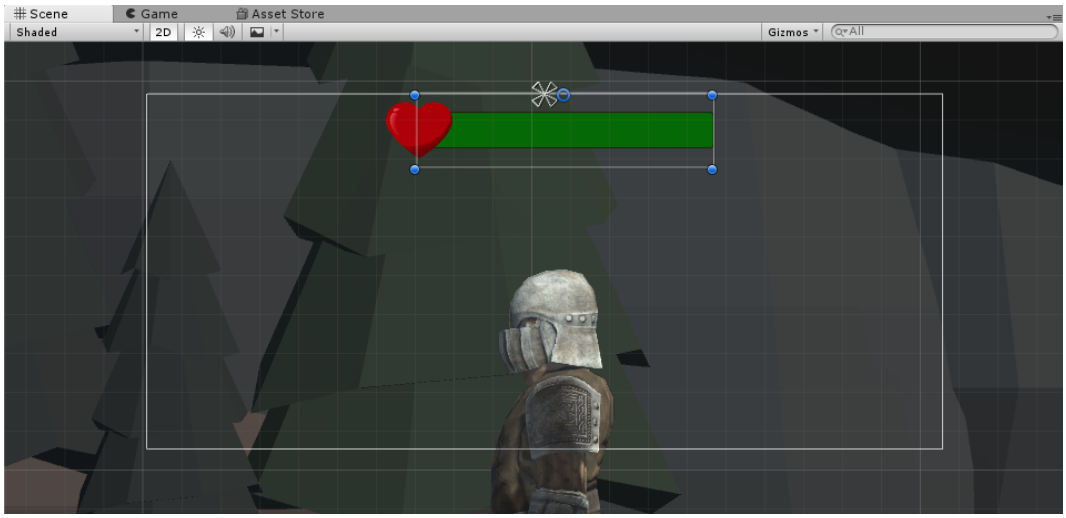
Slika 6.28. Objekt 'soldierDamage' s colliderom

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

public class enemyDamage2 : MonoBehaviour
{

    public float damage;
    public float damageRate;
    public float pushBackForce;

    float nextDamage;

    bool playerInRange = false;

    GameObject thePlayer;
    playerHealth thePlayerHealth;

    // Use this for initialization
    void Start()
    {
        nextDamage = Time.time;
        thePlayer = GameObject.FindGameObjectWithTag("Player");
        thePlayerHealth = thePlayer.GetComponent<playerHealth>();
    }

    // Update is called once per frame
    void Update()
    {
        if (playerInRange) Attack();
    }


    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            playerInRange = true;
        }

    }

    private void OnTriggerExit(Collider other)
    {
        if (other.tag == "Player")
        {
            playerInRange = false;
        }
    }

    void Attack()
    {
        if (nextDamage <= Time.time)
        {
            thePlayerHealth.addDamage(damage);
            nextDamage = Time.time + damageRate;

            pushBack(thePlayer.transform);
        }
    }

    void pushBack(Transform pushedObject)
    {
        Vector3 pushDirection = new Vector3(0, pushedObject.position.y - transform.position.y, 0).normalized;
        pushDirection *= pushBackForce;

        Rigidbody pushedRB = pushedObject.GetComponent<Rigidbody>();
        pushedRB.velocity = Vector3.zero;
        pushedRB.AddForce(pushDirection, ForceMode.Impulse);
    }
}

```

Slika 6.29. Skripta 'enemyDamage'

Zatim se dodaje novi Empty Child GameObject nazvan soldierDetection. Ovaj objekt ima samo Box Collider koji može biti trigeran i kada se zmajev collider sudari s njim, vojnik će detektirati zmaja i krenut će u napad.

U Hierarchy prozoru se kreira novi Canvas enemyFloatingHUD čija je svrha da prikazuje trenutni health vojnika. U postavkama Canvasa Render Mode se postavi na World Space tako da ovaj canvas ne ide preko cijelog ekrana kao onaj zmajev, nego će ovaj biti umanjen i bilo gdje na sceni gdje ga se postavi, a u ovom slučaju će lebdjeti iznad vojnikove glave. Canvasu se dodaje Slider i namješta se tako da izgleda kao i onaj zmajev. Canvas se u Hierarchy prozoru prebacuje pod vojnika tako da hijerarhijski bude child vojniku.



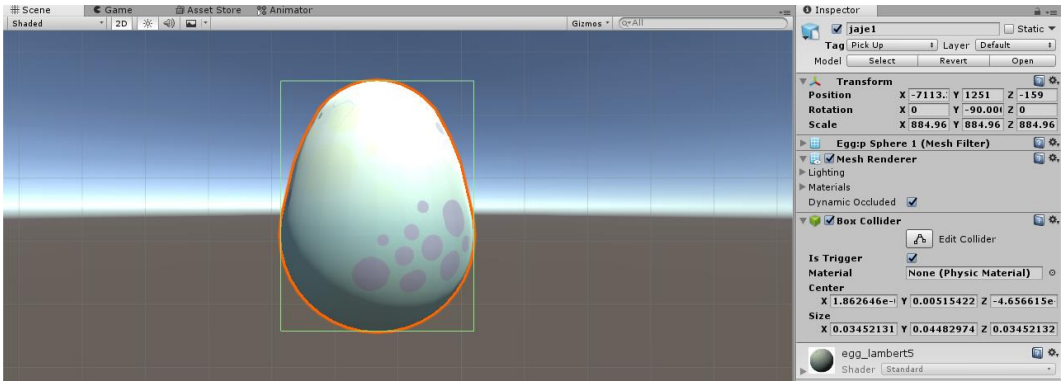
Slika 6.30. Lebdeći slider vojnikovog healtha

Da bi taj slider funkcionirao, vojnik prvenstveno mora imati health, stoga se na vojnika dodaje skripta enemyHealth.

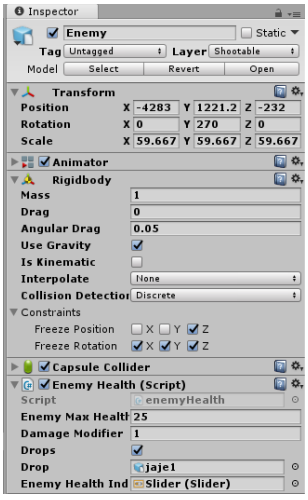
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class enemyHealth : MonoBehaviour {

    public float enemyMaxHealth;
    public float damageModifier;
    public bool drops;
    public GameObject drop;

    float currentHealth;

    public Slider enemyHealthIndicator;

    // Use this for initialization
    void Start () {
        currentHealth = enemyMaxHealth;
        enemyHealthIndicator.maxValue = enemyMaxHealth;
        enemyHealthIndicator.value = currentHealth;
    }

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

    }

    public void addDamage(float damage)
    {
        enemyHealthIndicator.gameObject.SetActive(true);
        damage = damage * damageModifier;
        if (damage <= 0f) return;
        currentHealth -= damage;
        enemyHealthIndicator.value = currentHealth;
        if (currentHealth <= 0)
        {
            makeDead();
        }
    }


    void makeDead()
    {
        //trun off movement
        //create ragdoll

        Destroy(gameObject.transform.root.gameObject);
        if (drops) Instantiate(drop, new Vector3(transform.position.x, transform.position.y + 35, transform.position.z), transform.rotation);
    }
}
```

Slika 6.31. Skripta 'enemyHealth'

Kada zmaj ubije vojnika, vojnik će ostaviti jaje. Na scenu se dodaje novi GameObject jaje kojem se dodaje Box Collider koji se triggera. Jajetu se mijenja Tag u Pick Up (što znači da će ga zmaj moći pokupiti i pritom će se u gornjem lijevom kutu UI-a promijeniti broj skupljenih jaja). Napravljena su četiri jajeta različitih boja tako da svaki vojnik ostavi drugačije jaje.



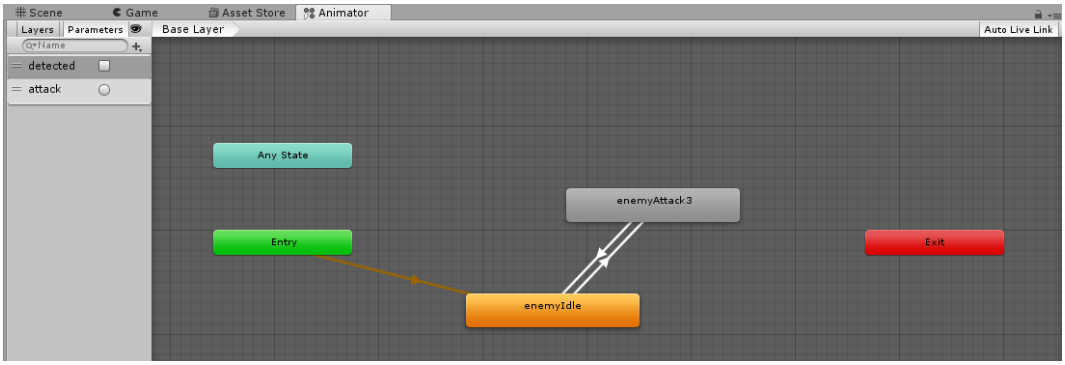
Slika 6.32. Postavke jajeta



Slika 6.33. – Nove postavke nakon ubacivanja skripte 'enemyHealth' na vojnika

Kad se vojniku dodala skripta enemyHealth, pojavile su se neke nove opcije koje se mogu mijenjati, tako da je stavljeno da vojnik ostavi jaje kao loot. Također, ubačen je slider kao indikator njegovog healtha.

Na vojnika se dodaje i komponenta Animator i pod Controller se ubacuje Soldier AC (Animator Controller). U Animatoru se drag and dropaju animacije Idle i Attack te se radi prijelaz između njih. Prijelazima se dodijele parametri detected (Bool) i attack (Trigger) koji će poslije trebati kod pisanja skripte. Isprogramirat će se tako da je vojnik u Idleu sve dok se zmaj ne sudari s njegovim detection colliderom. Tada vojnik krene napadati zmaja. Kad se napusti područje tog collidera, vojnik se vraća nazad u Idle.



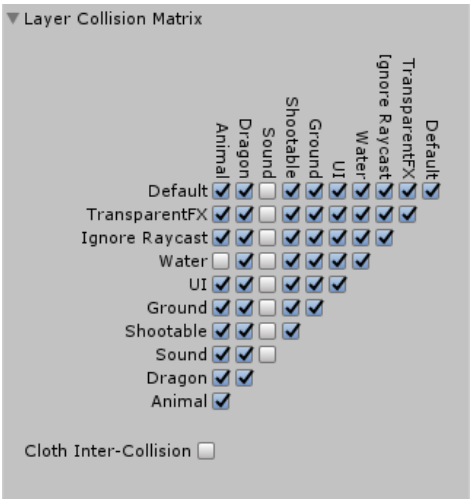
Slika 6.34. Animator (prozor) koji prikazuje animacije vojnika

Skriptu soldierController dodaje se na child objekt soldierDetection jer kao što je navedeno, kada se zmaj sudari s tim colliderom, vojnik detektira zmaja i krene ga napadati.

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

public class soldierController : MonoBehaviour {

    public float detectionTime;
    float startAttack;
    bool firstDetection;

    public float attackSpeed;
    public float idleSpeed;
    public bool facingLeft = true;

    float moveSpeed;
    bool attacking;

    Rigidbody myRB;
    Animator myAnim;
    Transform detectedPlayer;

    bool Detected;

    // Use this for initialization
    void Start () {
        myRB = GetComponentInParent<Rigidbody>();
        myAnim = GetComponentInParent<Animator>();

        attacking = false;
        Detected = false;
        firstDetection = false;
        moveSpeed = idleSpeed;
    }

    void FixedUpdate () {
        if (Detected)
        {
            if (!firstDetection)
            {
                startAttack = Time.time + detectionTime;
                firstDetection = true;
            }
        }

        if(!attacking && Detected)
        {
            if (startAttack < Time.time)
            {
                moveSpeed = attackSpeed;
                myAnim.SetTrigger("attack");
                attacking = true;
            }
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if(other.tag == "Player" && !Detected)
        {
            Detected = true;
            detectedPlayer = other.transform;
            myAnim.SetBool("detected", Detected);
        }
    }

    void OnTriggerExit(Collider other)
    {
        if(other.tag == "Player")
        {
            firstDetection = false;
            if (attacking)
            {
                myAnim.SetTrigger("attack");
                moveSpeed = idleSpeed;
                attacking = false;
                Detected = false;
            }
        }
    }
}

```

Slika 6.35. Skripta 'soldierController'

6.8. Update layera za zmaja i zvuk



Slika 6.36. Layer collision matrix – zvuk će se trigerati samo na zmaja

Kako bi objekt holder sa zvukom na sceni triggerao samo zmaj, bilo je potrebno napraviti neke izmjene s njihovim Layerima. Zmaju se promijeni Layer u Dragon, a holderu se stavi Layer Sound. Zatim se odabire Edit -> Project Settings -> Physics. U Inspectoru se otvore postavke i pod odjeljkom Layer Collision Matrix se isključi mogućnost triggeranja zvuka za sve Layere osim Layere Dragon i Animal.

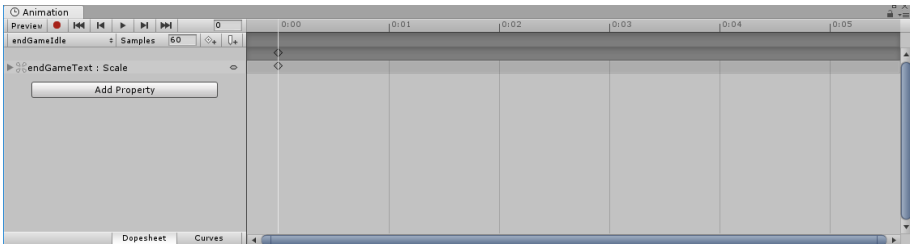
6.9. Pozadinska glazba

Slika 6.37. Postavke zvuka na objektu 'gameController'

Na scenu se dodaje GameObject pod nazivom gameController. Njemu se dodaje komponenta Audio Source koja sadrži Audio Clip, odnosno pjesmu koja će se cijelo vrijeme reproducirati u pozadini igre. U ovom slučaju nije važno gdje je ovaj objekt smješten na sceni jer mu je opcija Spatial Blend postavljena na 2D, što znači da će se zvuk jednako čuti bilo gdje u bilo kojem trenutku. Da je Spatial Blend bio postavljen na 3D, zvuk bi se čuo samo na distanci koju odredimo u daljnjim postavkama jer je taj zvuk onda ograničen na 3D prostor.

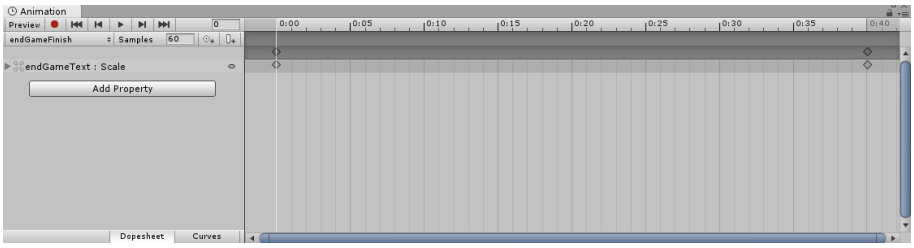
6.10. Losing i winning conditions

Svaka igra bi trebala imati neke uvjete koji se trebaju ispuniti da bi izgubili ili pobijedili. U ovom slučaju, na Canvas (playerCanvas) koji se odnosi na zmaja dodaje se novi UI element, tekst GAME OVER. Na tekst se dodaje Animator i pod Controller se ubacuje endGameText (Animator Controller). U ovom slučaju se koristite animacije napravljene u Unityu. Otvori se prozor Animation (Window -> Animation) i doda se svojstvo skaliranja (Scale). Prva animacija je endGameIdle i ima svega jedan ključni okvir (engl. keyframe) za koji je namješteno da je Scale teksta postavljen na 0, što znači da će tekst biti nevidljiv za vrijeme igranja.



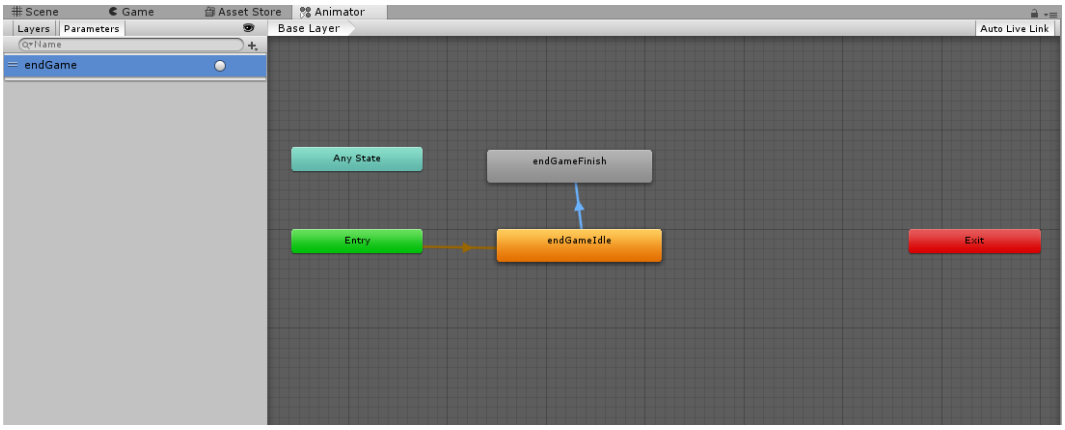
Slika 6.38. Animacijski prozor animacije 'endGameIdle'

Druga animacija je endGameFinish za koju se isto koristi svojstvo skaliranja (Scale). Ona ima dva ključna okvira (engl. keyframe). Prvi keyframe je postavljen na nultoj sekundi i Scale je postavljen na 0. Drugi keyframe je postavljen na četrdesetoj sekundi i Scale je postavljen na 1. Na taj način se dobije animacija zumiranja teksta.



Slika 6.39. Animacijski prozor animacije 'endGameFinish'

Te dvije animacije dodaju se u Animatoru i na prijelaz između njih dodaje se parametar endGame (Trigger).



Slika 6.40. Animator (prozor) koji prikazuje animacije teksta (endGameText)

Kada izgubimo u igri, na ekranu će se pojaviti poruka GAME OVER i igra će se resetirati na početak. Da bi se igra resetirala, na objekt gameController koji reproducira glazbu u pozadini dodaje se skripta restartGame.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class restartGame : MonoBehaviour {
6
7       public float restartTime;
8       bool resetNow = false;
9       float resetTime;
10
11      // Use this for initialization
12      void Start () {
13
14      }
15
16      // Update is called once per frame
17      void Update () {
18          if(resetNow && resetTime <= Time.time)
19          {
20              Application.LoadLevel(Application.loadedLevel);
21          }
22      }
23
24      public void restartTheGame()
25      {
26          resetNow = true;
27          resetTime = restartTime + Time.time;
28      }
29  }
30
```

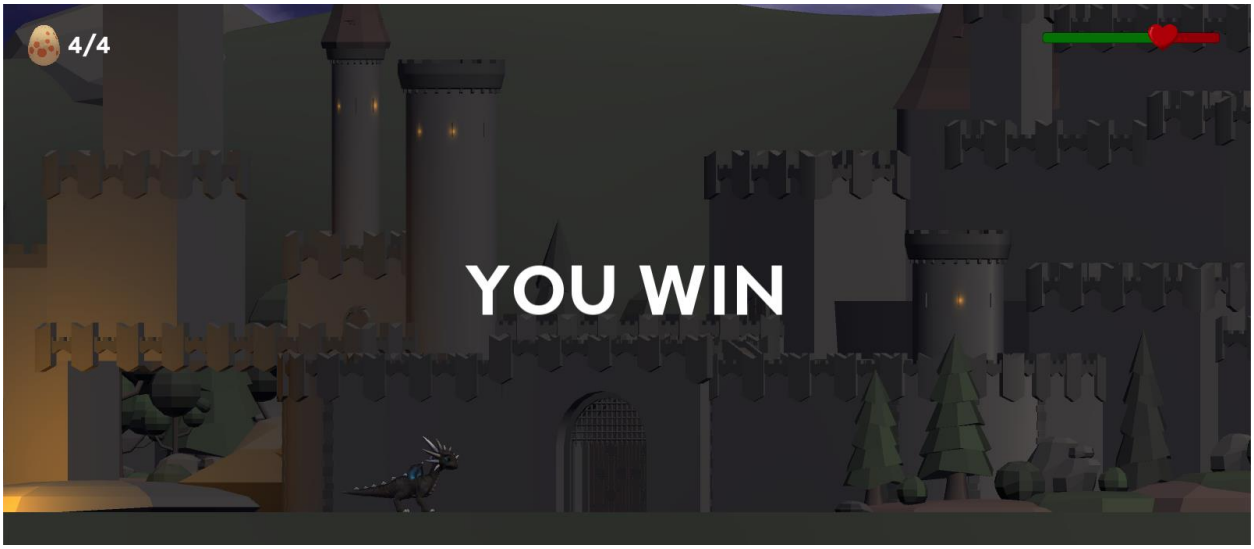
Slika 6.41. Skripta restartGame

Slika 6.42. Tekst GAME OVER kada izgubimo u igri

Da bi pobijedili u igri, moraju se skupiti sva četiri jajeta. Stoga, na zadnje jaje koje vojnik ostavi ubaci se skripta pickAndWin koja funkcionira slično kao i ona namijenjena za slučaj gubljenja, samo što se tekst mijenja iz GAME OVER u YOU WIN.

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

public class pickAndWin : MonoBehaviour {

    //HUD
    public Text endGameText;
    public restartGame theGameController;

    // Use this for initialization
    void Start () {

    }

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

    }

    private void OnTriggerEnter(Collider other)
    {
        if(other.tag == "Player")
        {
            endGameText.text = "YOU WIN";
            Animator endGameAnim = endGameText.GetComponent<Animator>();
            endGameAnim.SetTrigger("endGame");
            theGameController.restartTheGame();
        }
    }
}
```

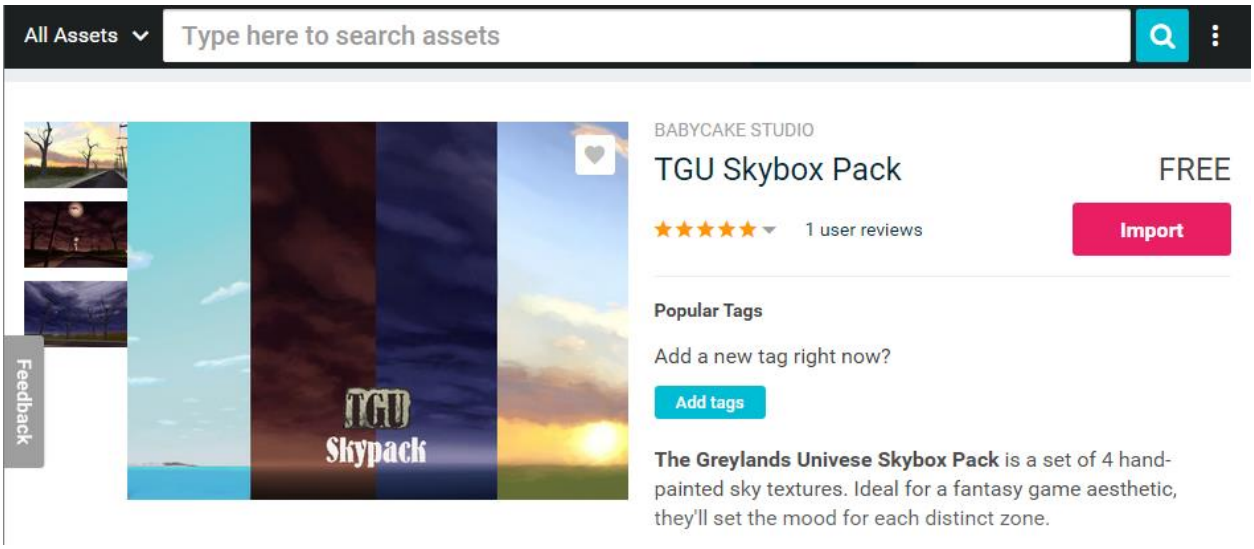
Slika 6.43. Skripta 'pickAndWin'



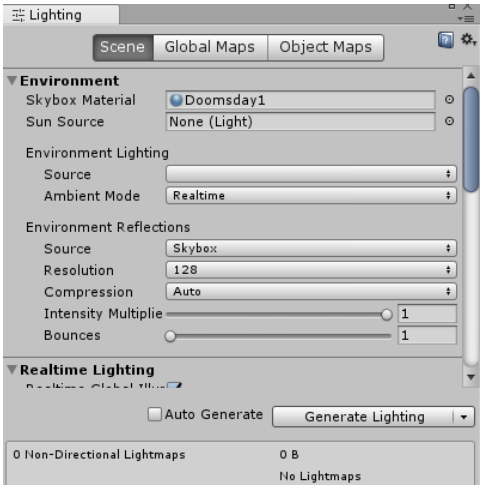
Slika 6.44. Tekst YOU WIN kada pobijedimo u igri

6.11. Nebo

Za nebo se preuzme besplatni asset TGU Skybox Pack s Asset Storea i importira se u Unity.



Slika 6.45. Unity asset store – TGU Skybox Pack koji sam iskoristila za promjenu izgleda neba



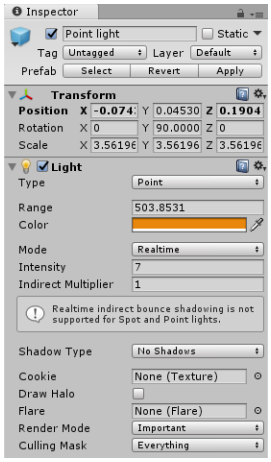
Slika 6.46. Lighting prozor u kojem se promijeni Skybox Material u novi skinuti Doomsday1

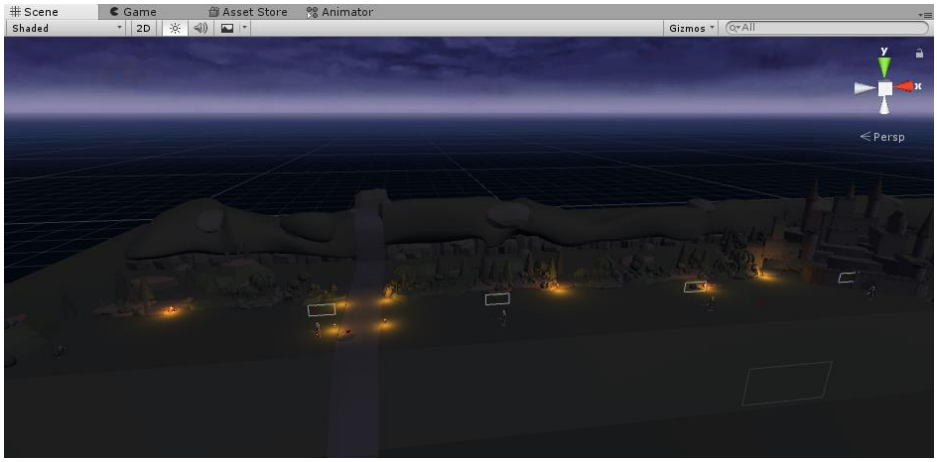
Zatim se u izborniku Window odabire Lighting -> Settings i u novootvorenom prozoru se pod Skybox Material umetne nebo Doomsday1.

Directional Lightu se u Inspectoru smanji Intensity na 0.45 jer se želi zamračiti scena, ali ne prejako jer se onda neće dobro vidjeti ostali objekti.

6.12. Konačni izgled scene

Na scenu se postavi zmaj na početku, zatim se postavi četiri vojnika na različitim mjestima. Svaki od vojnika će ostaviti jaje drugačije boje. Ubaci se i dva srceta (koja vraćaju health zmaju) te point lightovi koji su smješteni blizu logorske vatre i baklji i daju dojam da vatra gori i obasjava okolinu, ili u slučaju dvorca, da izgleda kao da su upaljena svjetla na prozorima. Na mjesta gdje su srca i gdje će vojnik ostaviti jaja postave se holderi sa zvukom koji se reproducira kad pokupimo iste.



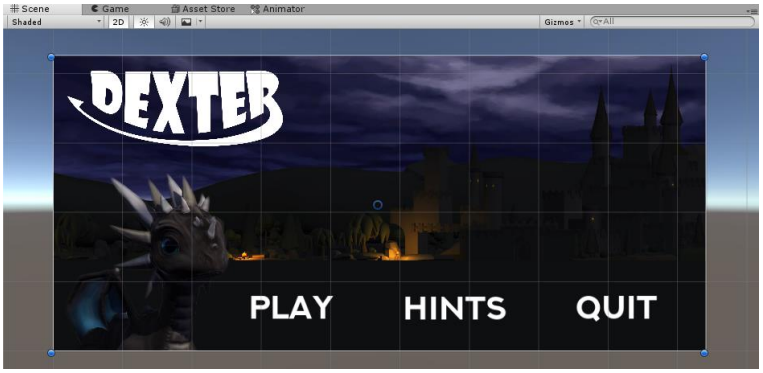


Slike 6.47. i 6.48. Postavke point lighta za scenu i konačna složena scena s point lightovima

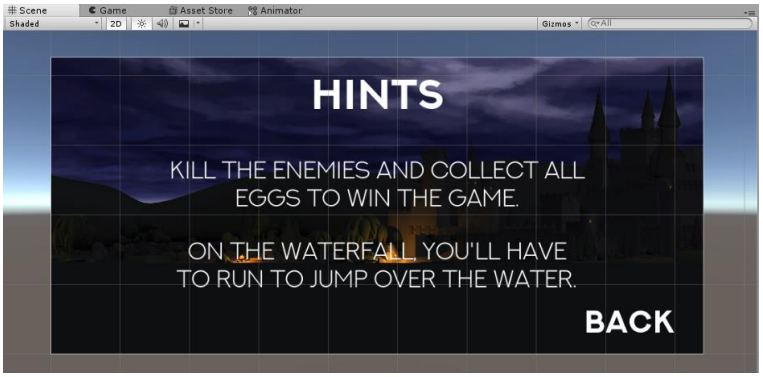
6.13. Glavni izbornik (Main menu)

Za kraj je ostala izrada glavnog izbornika iz kojeg će se moći pokrenuti igra, vidjeti hintovi (naputci za igranje) ili izaći iz igre. Napravi se potpuno nova scena, File -> New Scene i imenuje se Main. Njoj se u Hierarchy prozoru dodaje UI -> Canvas kojem se dodijele element Panel i tri gumba (engl. button) – Play, Hints i Quit i imenuje ga se MainCanvas. Panel u ovom slučaju predstavlja pozadinsku sliku izbornika.

Nakon toga se još jednom napravi novi Canvas i imenuje ga se TipsCanvas. Taj Canvas određuje izgled izbornika Hints kada se klikne na gumb Hints. Sastoji se od Panela s pozadinskom slikom, tri elementa teksta (jedan za naslov i dva za naputke) te gumba Back koji vraća natrag na glavni izbornik.



Slika 6.49. Main canvas – glavni izbornik



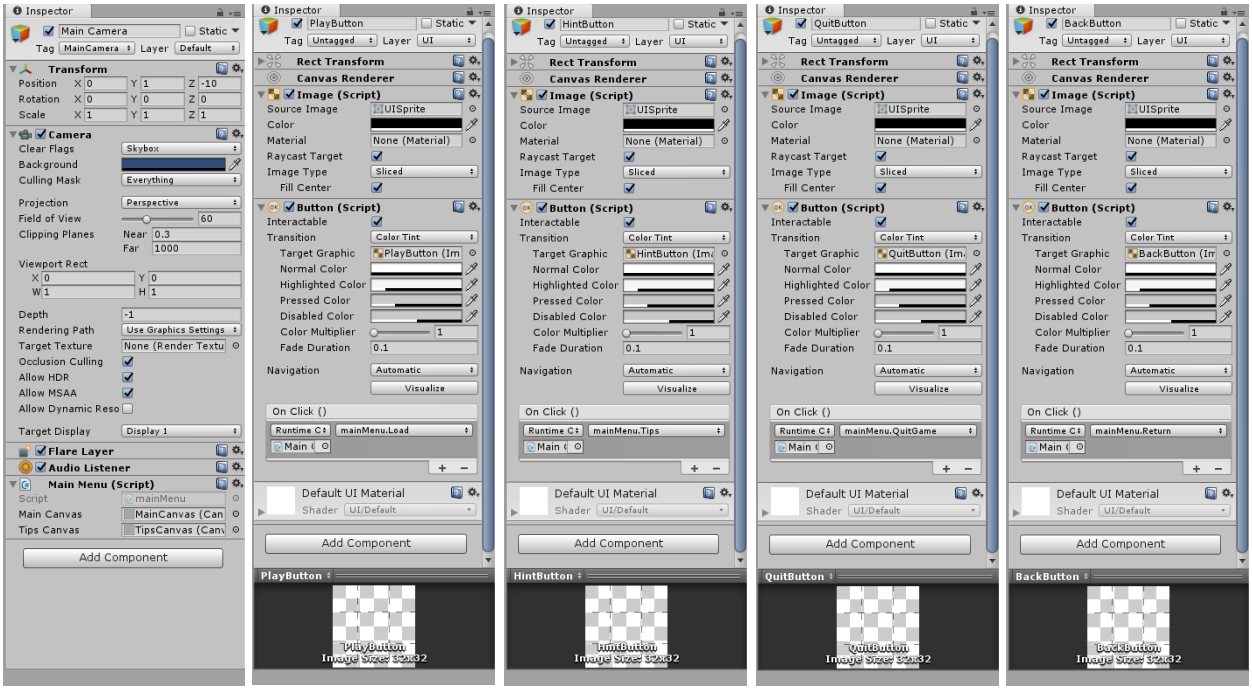
Slika 6.50. Tips canvas – sporedni izbornik s hintovima

Na glavnu kameru (Main Camera) ove scene dodaje se skripta mainMenu. Ta skripta određuje kada će se koji Canvas (Main ili Tips) prikazivati te kada stisnemo na gumb Play da se igra pokrene ili kada stisnemo na gumb Quit da izađemo iz igre, odnosno u slučaju gumba Back, da se vratimo nazad u glavni izbornik.

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

public class mainMenu : MonoBehaviour {

    public Canvas MainCanvas;
    public Canvas TipsCanvas;

    private void Awake()
    {
        TipsCanvas.enabled = false;
    }

    void Update()
    {
        if (Input.GetKey("escape"))
        {
            Debug.Log("Quit!");
            Application.Quit();
        }
    }

    public void Tips()
    {
        TipsCanvas.enabled = true;
        MainCanvas.enabled = false;

    }

    public void Return()
    {
        TipsCanvas.enabled = false;
        MainCanvas.enabled = true;

    }

    public void Load()
    {
        Application.LoadLevel(1);
    }

    public void QuitGame()
    {
        Debug.Log("Quit!");
        Application.Quit();
    }
}
```

Slika 6.51. Skripta 'mainMenu'

Na glavnoj kameri su se nakon ubacivanja skripte pojavila dva nova polja u koja se ubacuju Main i Tips Canvasi. Nakon toga se za svaki gumb mora dodati odgovarajuća funkcija pod odjeljkom On Click() koja je navedena u skripti mainMenu, npr. za gumb Play je bila namijenjena funkcija Load, za gumb Quit funkcija QuitGame, itd.



Slike 6.52., 6.53., 6.54., 6.55. i 6.66. Ubačena skripta na glavnu kameru scene Main i odabrane pripadajuće funkcije za svaki gumb

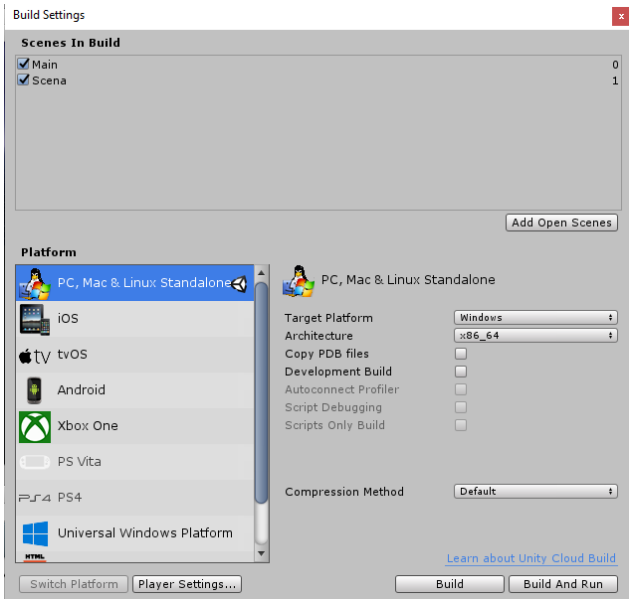
Vrati se nazad na početnu scenu gdje je igra složena i ubaci se novi Canvas nazvan Quit Canvas koji ide na vrh hijerarhije u Hierarchy prozoru. Ovaj Canvas sadrži samo jedan gumb Quit na koji se ubaci skripta za izlaženje iz igre.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class inGameQuit : MonoBehaviour {
6
7       void Update()
8       {
9           if (Input.GetKey("escape"))
10          {
11              Debug.Log("Quit!");
12              Application.Quit();
13          }
14      }
15
16      public void QuitGame()
17      {
18          Debug.Log("Quit!");
19          Application.Quit();
20      }
21  }
22
```

Slika 6.67. Skripta 'inGameQuit'

Slika 6.68. Konačan izgled igre u prozoru Game

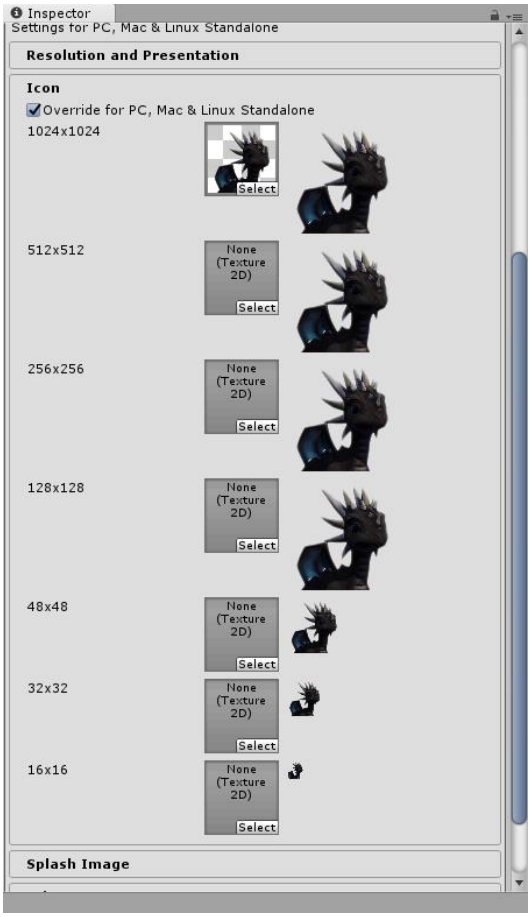
6.14. Izrada aplikacije (igre)

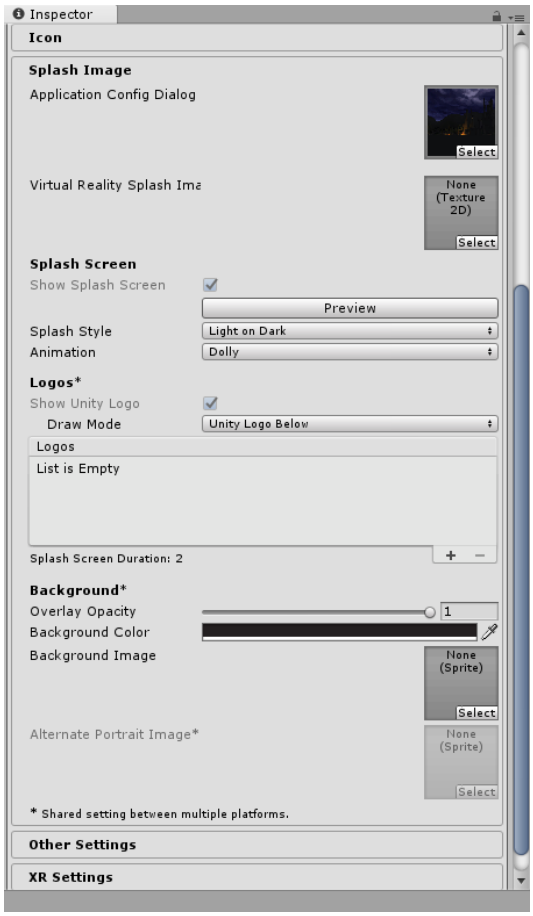


Slika 6.69. Build Settings

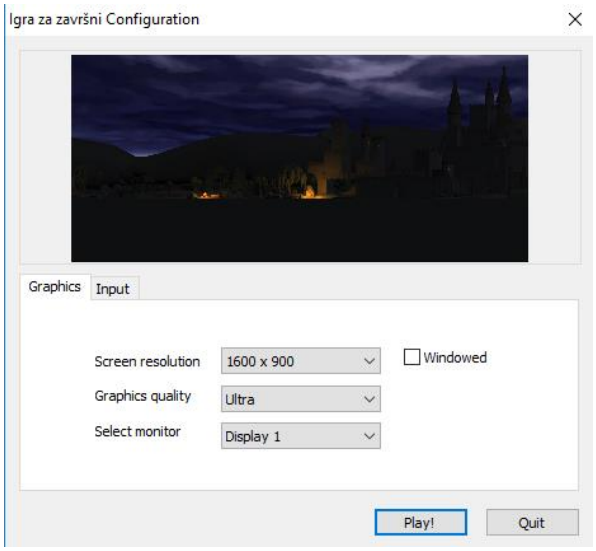
Završni korak je pretvoriti ovaj projekt u stvarnu aplikaciju. Odabirom na File -> Build Settings otvara se prozor u kojem možemo vidjeti koje scene će se pojaviti u aplikaciji. U ovom slučaju važno je da Main scena (glavni izbornik) ima oznaku 0 jer je to početna scena za koju želimo da se otvori.

Klikom na Player Settings u prozoru Inspector otvaraju se dodatne postavke. U ovom slučaju sve što se od postavki promijeni je to što se dodaje slika zmaja kao ikona igre te se pod Splash Image dodaje slika koja je korištena kao pozadina na glavnom izborniku. Ta slika će se prikazati na Configuration prozoru igre nakon što aplikacija bude puštena.





Slike 6.70. i 6.71. Ikona i splash image igre



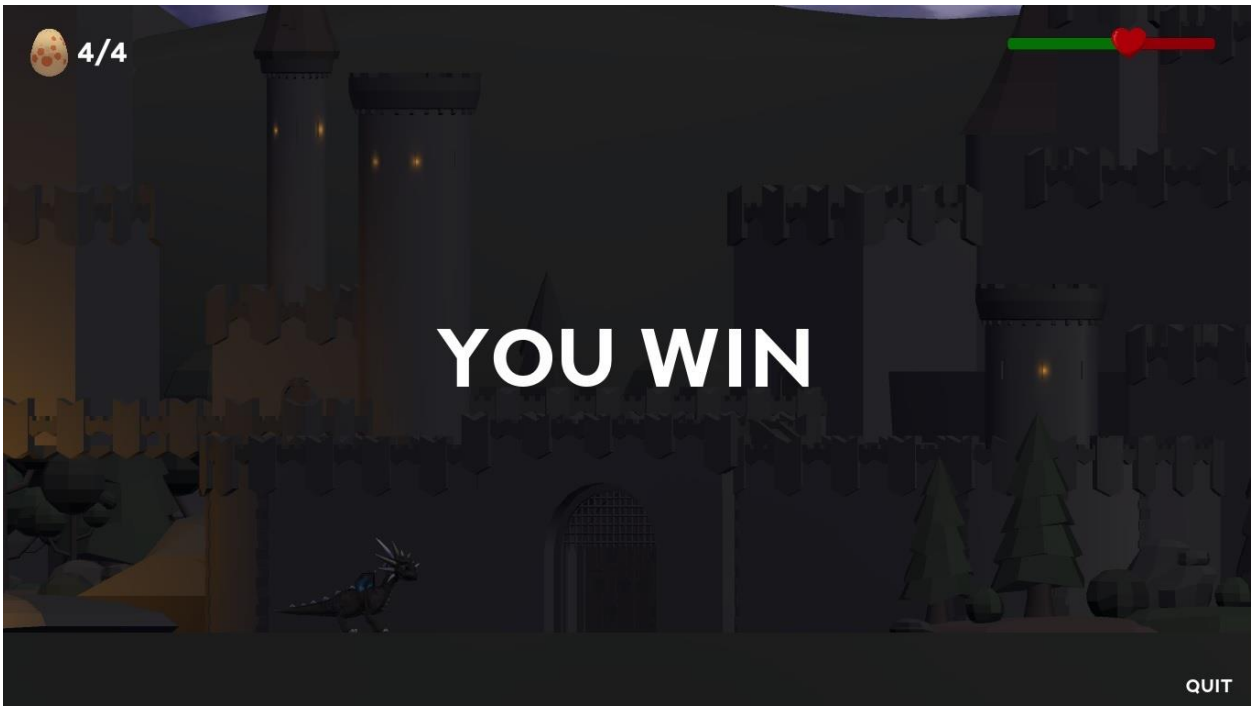
Slika 6.72. Konfiguracija igre

Kada su se podesile sve postavke, klikne se na Build i projekt se pretvara u aplikaciju.

Slika 6.73. Igra Dexter

Slika 6.74. Igra Dexter 2

Slika 6.75. Igra Dexter 3



Slika 6.76. Igra Dexter 4

7. Zaključak

U ovom radu korištena su dva programa kako bi se stvorila jedna velika cjelina – igra. Korisničko sučelje, princip rada, osnovne mogućnosti i alati su opisani za oba programa. Cilj je bio, ne samo izraditi igru, nego i pobliže se upoznati s ovim programima i njihovim alatima. Rad u Mayi je bio lakši s obzirom da smo se već susreli s tim programom na fakultetu, tako da je veći izazov bio Unity s kojim sam se učila raditi od početka.

Većina modela napravljena je u Autodesk Mayi. Svaki od njih je rađen poligonalnim modeliranjem te je po potrebi još modificiran s nekim drugim alatima. Također, animiranje i teksturiranje su uglavnom odrađeni u Mayi. Razlog zašto to sve nije odrađeno u Unityu je taj što Maya ipak ima više mogućnosti kad je u pitanju manipuliranje s 3D objektima. S obzirom da sam se prvi put susrela s Unityem, moram potvrditi da je vrlo lagan za korištenje i vrlo brzo se nauče neke osnovne stvari i korištenje nekih osnovnih alata. Ima poprilično jednostavno sučelje koje olakšava snalaženje u samom programu. Najzahtjevniji dio Unitya je bio programiranje, odnosno skriptiranje, s obzirom da se po prvi put susrećem i sa C# jezikom. Ipak, prednost C# je ta što postoji jako puno tutoriala i primjera kodova koji su mi uvelike olakšali programiranje. Jedna od najkorisnijih stvari koje se mogu naučiti u Unityu je izrada korisničkog sučelja koja je poprilično jednostavna, a omogućuje korisniku da preko njega na neki način komunicira s igrom.

Danas je vrlo lako napraviti igru na osobnom računalu s obzirom na sve što nam se nudi, od tutoriala, knjiga, foruma, itd. Ipak, kvaliteta i izgled (grafika) igre uvelike ovise o znanju, uloženom vremenu, broju ljudi koji rade na njoj ili čak i jačini osobnog računala. Kao što se može vidjeti na ovom primjeru, ne mora se sve izraditi striktno u jednom programu, nego su mogućnosti poprilično velike kraj svih softvera i enginea koji nam se nude jer svaki od njih ima neke svoje prednosti koje možemo iskoristiti. Da je izrada igara dovoljno lak proces, možemo vidjeti prema povećanom broju razvijenih igara u posljednjih nekoliko godina.

U Varaždinu, _____ _____

Datum Potpis

8. Literatura

[1] M. Klappenbach: What is a Platform Game?, Lifewire, 2. veljače 2018.
Dostupno na: https://www.lifewire.com/what-is-a-platform-game-812371, rujan 2018.

[2] J.Willard, L.Braun, S.Ansari-Tadi, D.Esposto: History of Platform Games, Multimedia 3K03/McMaster University
Dostupno na: http://joshwillard.com/platformgames/?page_id=4, rujan 2018.

[3] Autodesk, Features
Dostupno na: https://www.autodesk.eu/products/maya/features, rujan 2018.

[4] Autodesk Knowledge Network
Dostupno na: https://knowledge.autodesk.com/, rujan 2018.

[5] Andrija Bernik (prosinac 2010.): Vrste i tehnike 3D modeliranja, Tehnički Glasnik: Časopis Veleučilišta u Varaždinu. Varaždin, Hrvatska str. 45 - 47.
Dostupno na: http://hrcak.srce.hr/index.php?show=clanak&id_clanak_jezik=127863, rujan 2018.

[6] Andrija Bernik, Kelnarić D. (srpanj 2011.): Vrste i tehnike 3D teksturiranja, Tehnički Glasnik: Časopis Veleučilišta u Varaždinu. Varaždin, Hrvatska str. 30 - 33.
Dostupno na: http://hrcak.srce.hr/index.php?show=clanak&id_clanak_jezik=127725, rujan 2018.

[7] D. Vusić, Z. Sabati, A. Bernik: 3D modeliranje u primjerima 1., Sveučilište Sjever, Varaždin, 2014.

[8] D. Vusić, A. Bernik, R. Geček: 3D modeliranje u primjerima 2., Sveučilište Sjever, Varaždin/Koprivnica, 2016.

[9] Maya 2014 Basic Animation & The Graph Editor, Wellesley College
Dostupno na:
https://www.wellesley.edu/sites/default/files/assets/departments/art/files/maya2014-basicanimationthegrapheditor.pdf, rujan 2018.

[10] Autodesk Maya 2018, What's New in Maya 2018
Dostupno na: http://help.autodesk.com/view/MAYAUL/2018/ENU/?guid=GUID-EAF7F00F-0380-4E8E-94F9-D484CAD5505A, rujan 2018.

[11] G. Maestri: The differences between Maya and Maya LT, Lynda, 31. srpanj 2018.
Dostupno na: https://www.lynda.com/Maya-tutorials/differences-between-Maya-Maya-LT/408407/423316-4.html, rujan 2018.

[12] Pluralsight, 3ds Max, Maya LT or Blender – Which 3D Software Should I Choose for Asset Creation
Dostupno na: https://www.pluralsight.com/blog/film-games/3ds-max-maya-lt-blender-3d-software-choose-asset-creation, rujan 2018.

[13] E. Lavieri: Getting Started with Unity 5, Packt Publishing, Birmingham – Mumbai, 2015.

[14] Unity Documentation, Unity User Manual
Dostupno na: https://docs.unity3d.com/Manual/index.html, rujan 2018.

9. Popis slika

Prilozi:

DVD