Izrada akcijske igre u programskom alatu Unity

Mitrić, Jerko

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jerko Mitrić

Izrada akcijske igre u programskom alatu Unity

ZAVRŠNI/DIPLOMSKI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Jerko Mitrić

Matični broj: 43293/14–R

Studij: Informacijski sustavi

Izrada akcijske igre u programskom alatu Unity

ZAVRŠNI/DIPLOMSKI RAD

Mentor:
Dr. sc. Mladen Konecki

Varaždin, rujan 2018.

Jerko Mitrić

Izjava o izvornosti

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

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

_____

Sažetak

Izrada akcijske igre nalik DOOM-u, te upoznavanje sa enginom u kojem će se izrađivati. Detaljan opis Unitya i svih njegovih alata. Uspoređivanje engina s konkurencijom. Upoznavanje sa logikom izrade First Person Shooter-a, te općenite logike izrade igrica.

Ključne riječi: Unity, akcijska igra, FPS, algoritmi, 3D modeli

Sadržaj

1. Uvod

Kako bismo shvatili kako napraviti računalnu igru potrebno je istražiti i razumjeti logiku koja stoji iz izrade računalnih igara. Nakon savladavanje često jednostavne logike potrebno je upoznati se sa programom i programskim jezikom za izradu projekta. Moj projekt je jednostavan računalna igra sa elementima kretanja, pucanja, atributa igrača te naravno neprijatelja. Računalnu igra se može napravit na razne načine, no u današnje vrijeme većina industrije koristi „game engine“ iliti program specifičan za izradu računalnih igara. Naime pojam računalna igra ne odnosi se isključivo na igre koje se igraju na stolnom računalu već na sve igre koje se pokreću na nekom uređaju (Playstation, Nintendo). U ovom projektu je korišten Unity game engine. Tokom izrade projekta se nailazi na mnoge prepreke koje vrlo često otvaraju dodatne probleme ili otvaraju nove ideje za poboljšanje same igre. Probleme ćemo detaljnije opisati u nastavku ovog rada.

2. Povijest razvoja računalnih igara

Ideje prvih računalnih igara su se počele razvijati 60-ih godina prošlog stoljeća. Naime jedna prvih ikada računalnih igara je bila „Spacewar!“, koju je napravio Steve Russell 1962. godine. Steve Russell je bio student na sveučilištu Massachusetts Institute of Technology (MIT). Jedan od problema na koje su naišli je bila nemogućnost računanja trigonometrijskih funkcija. Trigonometrijske funkcije su bile potrebe za izračun putanje svemirskog broda. U pomoć im je priskočio Alan Kotok koji je već imao napisan potreban kod. (Graetz, J. M., kolovoz 1981.)

Spacewar je bila vrlo jednostavna računalna igra, ali za vrijeme računala sa vakumskim cijevima je to bio velik poduhvat. Prvo računalo na kojem se igrao Spacewar je bio PDP-1.

Slika 1 - Spacewar na PDP-1 računalu (Izvor: https://en.wikipedia.org/wiki/Spacewar!)

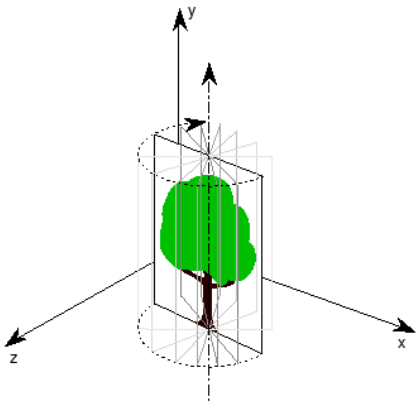
Početkom sedamdesetih započinje i era prvih igraćih automata. Atari izdaje igru Pong 1972. godine. Pong je razvio Nolan Bushnell. U nekoliko sljedećih godina igre na automatima su pokorile svijet. Sredinom 80-ih se razvijaju različite osobne konzole za video igre. Nintendo je pionir u razvoju igara za konzole sa igrama kao što su Mario Bros, Tetris, Super Mario 64 i Pokemon.

Slika 2 - Nintendo Entertainment System (Izvor: https://en.wikipedia.org/wiki/Nintendo_Entertainment_System)

Doba konzola kao što su Playstation i GameBoy će trajati sve do 1995. godine. Njihovu vladavinu će prekinuti 3dfx-ova grafička kartica Voodoo Graphics PCI. Prva cjenovno prihvatljiva 3D grafička kartica. Voodoo je omogućila poboljšanje tadašnjih 2D kartica te nije mogla raditi bez njih. Naravno kasnije iteracije su omogućile samostalnu uporabu. Sve do danas računala su uvijek jača nego konzole.

U istom periodu nastaju prve velike „first person shooter“ igrice. Jedna od najvećih ikad je bila Wolfenstein 3D, izdan 1992. godine. Kompanije iz Wolfensteina je bila id Software. Naime Wolfenstain nije bila prava 3D igra kakve očekujemo danas. To je bila 2D igra koja je pametnim „prečacem“ stvarala dojam da je 3D igra, taj prečac danas znamo kao billboarding. Billboarding je tehnika okretanja plošnih objekata prema kameri. Objekt na sebi ima teksturu te tako dobivamo dojam lažnog 3D objekta.

Slika 3 – Wolfenstein 3D (Izvor: https://www.imdb.com/title/tt0304947/mediaviewer/rm483801088)



Slika 4 – Billboarding (Izvor: http://www.bluevoid.com/opengl/sig00/advanced00/notes/node74.html)

Prvi pravi 3D engine je bio Quake Engine. Debitirao je sa igrom „DOOM“ 1993. godine. Quake je napisan u C programskom jeziku te koristi OpenGL API i Direct3D. Tokom izrade Quake enginea nisu imali na umu samo jednu igricu već pravi razvojni alat koji će služiti dulji niz godina. U nekoliko sljedećih godina izdaju više Quake iteracija na kojima su izrađene jedne od najpoznatijih igara ikada, kao Call of Duty franšiza.

3. Unity

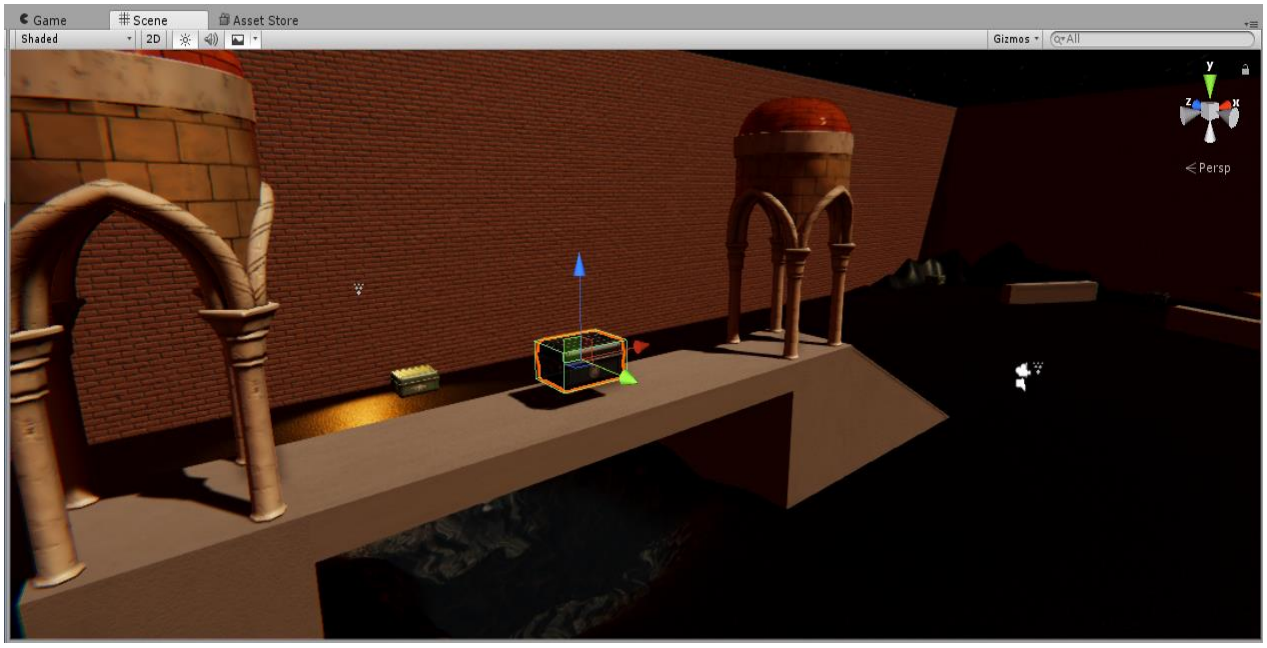
Unity je program za izradu računalnih igara. Jedna od ogromnih prednosti nad ostalim engine-ima jest razvojna podrška za velik broj različitih platformi. U Unityu možemo napraviti igru za iOS, Android, Windows, Playstation 3 i 4, Xbox One, Mac, Linux itd. Unity je razvila tvrtka Unity Technologies. Najavljen je i izdan 2005. godine. Zbog jednostavnog i razumljivog UI-a, korištenja skriptnih jezika i drag-drop funkcionalnosti Unity postaje jedan od najpoznatijih engina na svijetu. Još jedna velika prednost Unity-a nad ostalim game engine-ima je besplatna licenca. To daje prednost studentima i ostalim malim proizvođačima da krenu prototipirati svoje projekte bez velikog novčanog troška. Unity je podržavao UnityScript (Javascript za Unity), Boo i C#. U verziji 2017.1 više ne pruža podršku UnityScript-u,a podrška za Boo je maknuta još 2014 godine. Takvu odluku su donjeli jer je više od 85.4% projekata, većina dokumentacije i tutoriala koristilo C#.
Unity ima veliku podršku različitih programa kao što su Blender, Adobe Ps, Fireworks itd.

3.1. Unity korisničko sučelje

3.1.1. Scene prozor

Otvaranjem Unitya susrećemo se početnim prozorom. Početni prozor sadrži puno bitnih informacija u izradi projekta.

Većinski dio prozora čini Scene prozor. Na njemu vidimo grubi pregled kako naša igra izgleda. Pomoću njega postavljamo i modificiramo objekte u svijetu igre. U gornjem desnom kutu se nalazi alat koji služi za brzo pomicanje kamere ili mijenjanje perspektive. U Scene prozoru možemo birati točno koje komponente želimo vidjeti u danom trenutku, kao što je osvijetljene, itd.



Slika 5 - Scene prozor

3.1.2. Inspektor

Inspektor je moćan alat većine game enginea. Inspektor nam omogućava detaljnije promjene nad objektima. Kao primjer možemo uzeti lokaciju objekta u svijetu igre. Do sada smo u Scene prozoru mogli postaviti objekt gdje želimo, ali ne s velikom točnošću. U inspektoru iznos lokacije je točan čak na 6. decimalu. U inspektoru vidimo svaku osobinu nekog objekta, kao što su osvijetljene, skripta, collider itd.

3.1.3. Hijerarhija

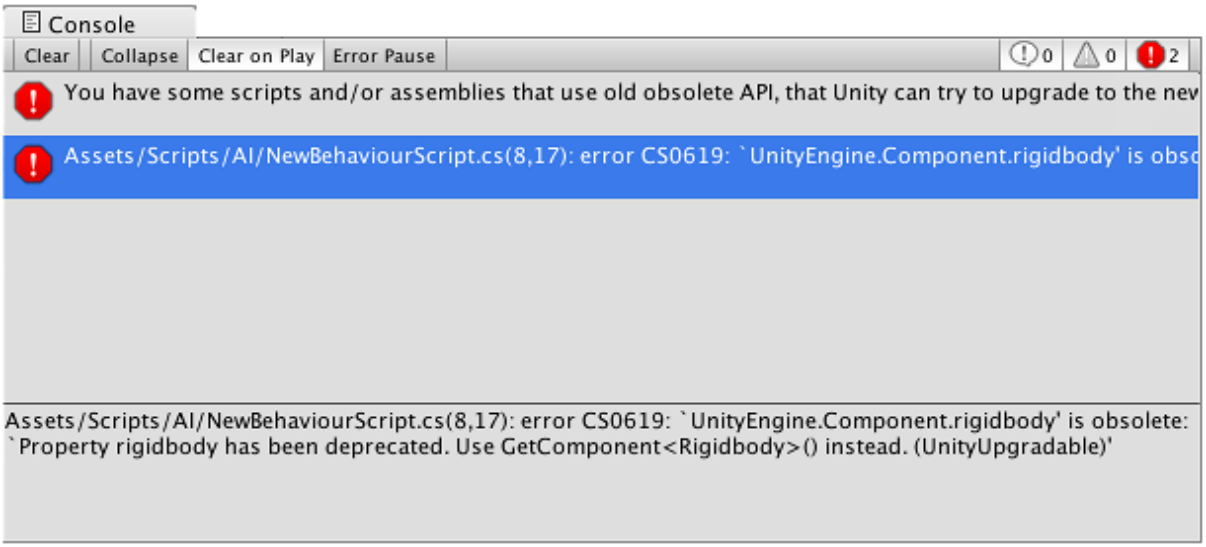
Hijerarhija sadrži sve objekte koji se nalaza u sceni koju sada pregledavamo. Pošto se u game developmentu često nalaze objekti unutar objekata. Vrlo lagano se možemo pogubit u redoslijedu utjecaja jednih na druge, taj problem je riješen hijerarhijom. Klikom na kompleksniji objekt izlistava se popis svih podobjekata do kojih je vrlo teško doći u Scene prozoru.

3.1.4. Projektni prozor

U projektnom prozoru vidimo sve assete i skripte koje možemo koristi u projektu. S lijeve strane imamo hijerarhijski prozorčić, a s desne popis svih mapa ili objekata za odabranu mapu. Svaka vrsta objekta označena je pripadajućom ikonom. Klikom desnog klika na mišu možemo stvoriti bilo koju vrstu objekta.

3.1.5.Konzola

Prozor konzole prikazuje poruke, upozorenja i greške koje se događaju u igrici. Svaku pojedinu vrstu poruka može se onesposobiti ako ju ne želimo vidjeti. Konzola je jedan od najbitnijih elementa bilo koje programskog alata pa tako i u Unityu.



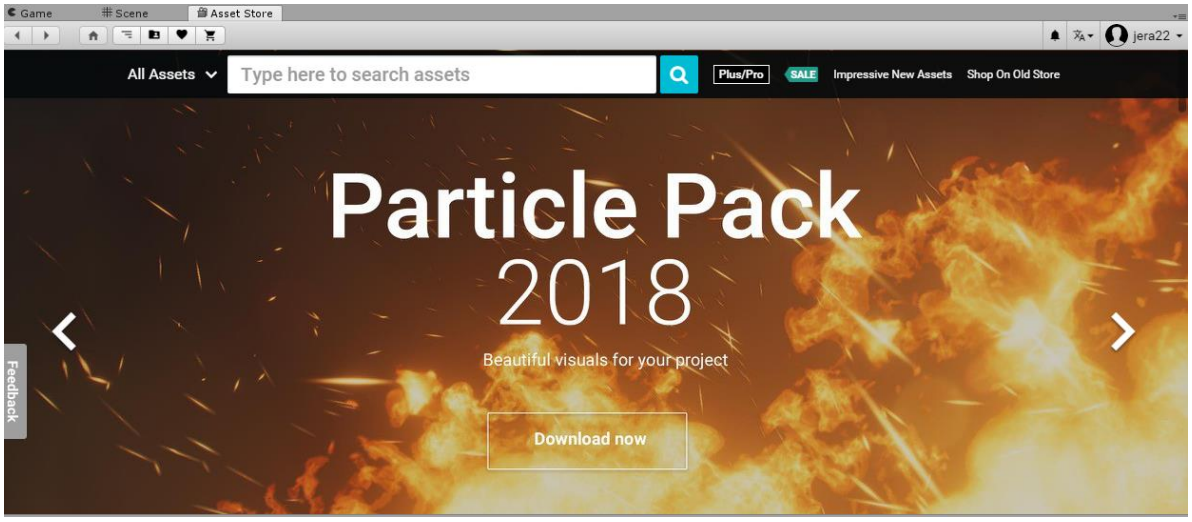
Slika 6 - Ispis grešaka u konzoli (Izovr: https://docs.unity3d.com/Manual/Console.html)

3.1.6.Game prozor

U game prozoru se prikazuje što vidi kamera glavnog igraća, na taj način dobivamo jasnu sliku što će korisnik vidjeti tokom igranje igre.

3.2. Unity Asset

Asset je sredstvo koje koristimo u našim projektima. U Unityu asset se odnosi na 3D, 2D modele, skripte, animacija, dodatke, dodatne alate itd. Unity Asset trgovina (Unity Asset Store) je web stranica na kojoj se nalaze svi asseti koje proizvodi zajednica ili kompanija. Prednost Unityeve trgovine je jako velik izbor besplatnih, visoko kvalitetnih asseta. Naravno zajednica može prodavati svoje proizvode na trgovini bez posebnih uvjeta ili poreza.

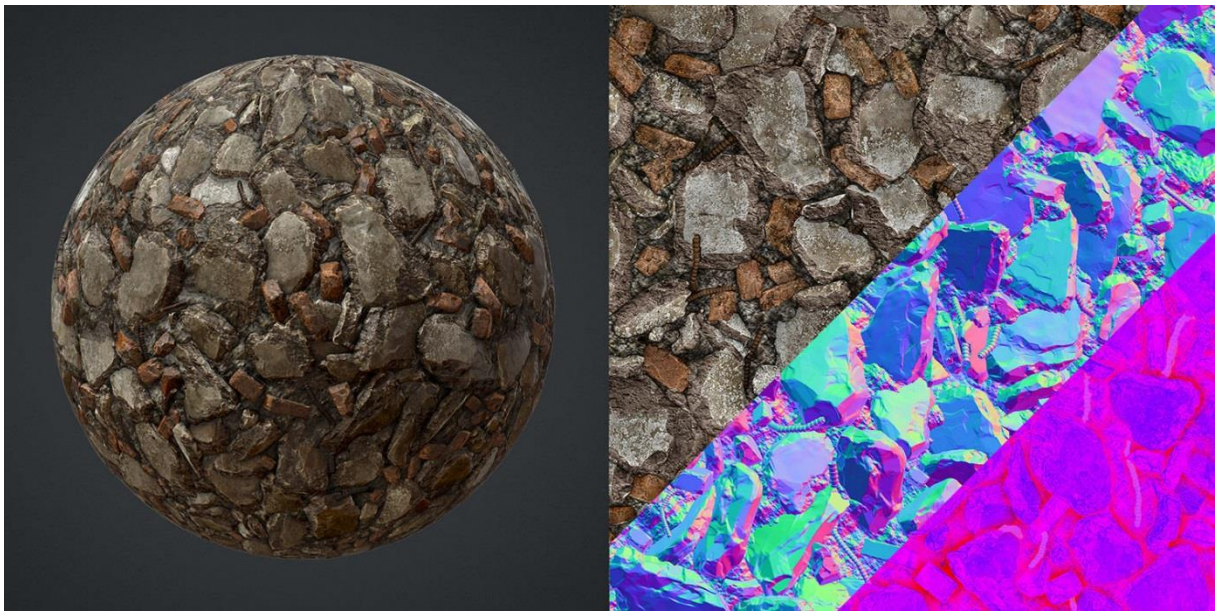


Slika 7 - Asset trgovina (Izvor: https://assetstore.unity.com/)

Assetima možemo pristupiti putem jednostavnog sučelja ugrađenog u Unity koji omogućuje preuzimanje i uvoz sredstava izravno u projekt. Trgovini pristupamo klikom na gumb Asset Store koji se nalazi iznad prozora scene.

3.3. Teksture

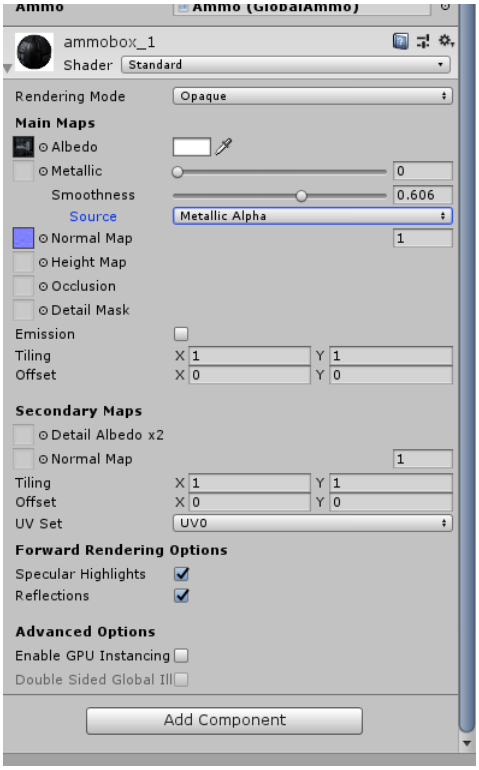
Tekstura je površinski izgled plohe ili objekta. U izradi video igara najčešće se koristi obična slika koja je prevučena preko mrežaste površine nekog 3D objekta. Pozicioniranje tekstura se najčešće odrađuje u istom programu u kojem je napravljena mrežasta površina 3D modela.



Slika 8 - Tekstura visoke kvalitete (Izvor: https://www.artstation.com/artwork/588ez)

U Unityu teksture postavljamo na objekt pomoću materijala (Materials). Materijali koriste specijalizirane programe koje nazivamo shaderi kako bi prikazali teksture na objektima. Shaderi omogućavaju prikazivanja osvijetljena i različitih efekata bojanja kako bi se simulirala nepravilna ili sjajna površina.

Unity pruža veliku fleksibilnost kod izrade i modificiranja tekstura. Svakoj postavljenoj teksturi u materijal možemo mijenjati način na koji je renderirana. Postavke kao metallic ili glatkoća su osnovni dio enginea za razliku od konkurencije koja se mora oslanjati na drugačije programe.



Slika 9 - Svojstva materijala u inspektoru

3.4. Logika i skriptiranje

Skriptiranje u računalnim igricama je najbitniji dio tokom izrade proizvoda. Bez skripti odnosno programiranja ne bi bilo moguće povezati sve dijelove igre u jedan završeni proizvod. Bitan aspekt igara jest njena ideja i logika. Prije početka izrade projekta moramo imati jasnu viziju kako ona želi izgledati i što bi sve igrač mogao raditi u njoj. Kao primjer logike ćemo uzeti ovaj projekt.

Logika u „First Person Shooter“ (dalje u radu FPS) igricama je relativno jednostavna, ali može postati komplicirana vrlo brzo dodavanjem naizgled jednostavnih mehanika igre. Kao primjer neuobičajene mehanike igara u kojima se puca je usporavanje vremena. Nakon odluke o implementaciji takve mehanike bitno se mijenja način razmišljanja i developmenta igre.

Svaka FPS igrica mora imati barem tri osnovne mehanike (kretnja, pomicanje kamere, pucanje). Prvu skriptu susrećemo kod pomicanja glavnog lika. Na sreću Unity već ima nekolicinu takvih skripti te ih možemo slobodno koristiti.

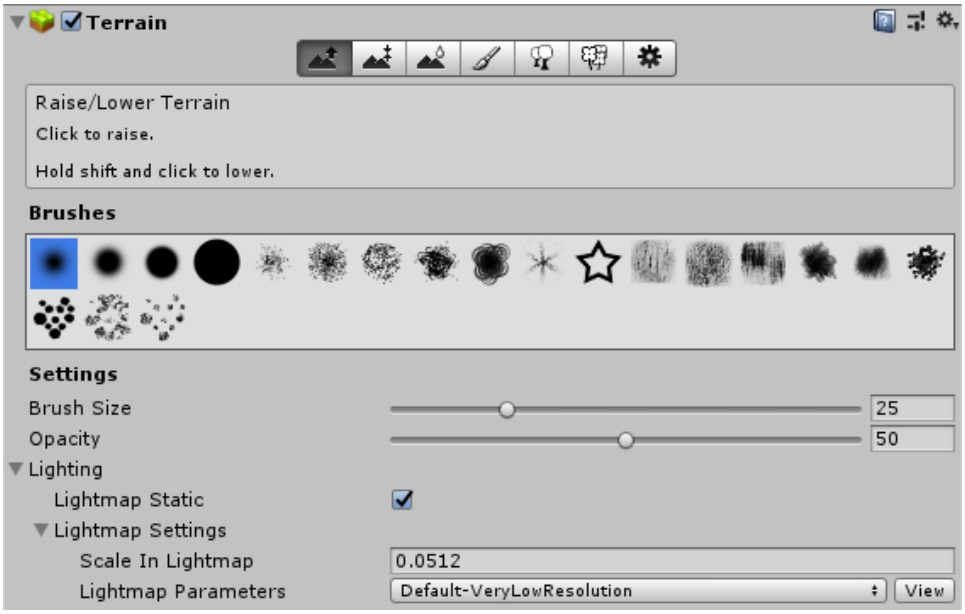
3.4.1.First Person Controller (Script)

First person controller skripta nam omogućava pokretanje i uvid u perspektivu glavnog lika. Pomicanjem miša lijevo desno rotiramo kameru po xy osi, a ako pomičemo miš gore dolje dodajemo i z komponentu kod rotacije. Pritiskom zadane tipke (najčešće W i S) pomičemo glavnog lika linearno po x osi. Pritiskom A i D tipki pomičemo lika po y osi. Kao i u svakoj modernoj igrici a i realnom svijetu glavni lik ima mogućnost skakanja ili čučnja. Pritiskom „space“ tipke lik se pomiče po z osi. U FPS skripti imamo nekoliko svojstava koje možemo mijenjati kako bi skok izgledao što realnije npr. Jump speed, Gravity multiplier i Stick to ground force. Kako bi sve kretnje imale dojam realizma potrebno je imati i akustični podražaj. Skakanje i trčanje potkrijepljeni su pripadajućim zvukovima. To znači da svaki puta kada glavni lik skoči i prizemlji se reproducirati će se zvučni efekt. Kasnije ćemo zvukove i kako ih koristimo objasniti detaljnije.

Slika 10 - Svojstva First Person Controllera

3.5. Teren i Navmesh

Unityev sistem terena nam omogućava izradu i dodavanje ogromnih pejzaža. Da bi smo dodali objekt terena u scenu, trebamo odabrati na alatnoj traci Game Object. Tu nam se otvara niz različitih vrsta objekata koje možemo napraviti. U ovom slučaju trebamo 3D objekt koji ima naziv Terrain. Nakon što se doda asset terena prikazuje se već prije zadani objekt. Početni teren je jedna velika ravna ploha koju možemo modelirati kako god poželimo. Unity ima odličnu podršku za takvu vrstu zadatka kroz širok spektar alata.

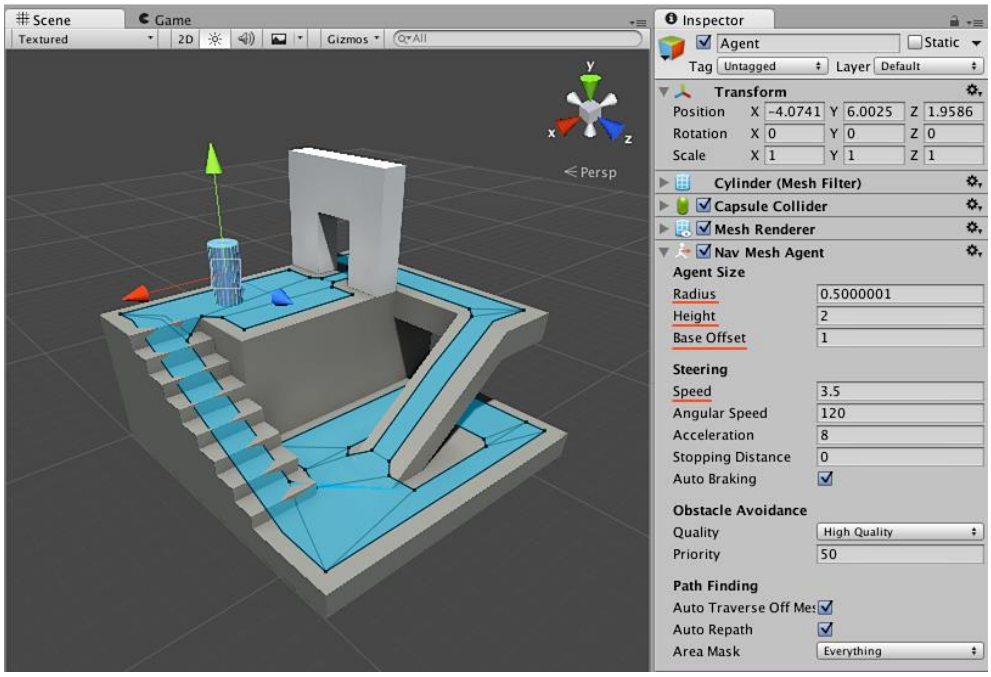


Slika 11 - Alati za uređivanje terena

Svaki alat ima opciju za biranje vrste kista, veličine kista i jačine kista osim alata za drveće. Pogledamo li sliku 11. alati podsjećaju na slične programe za obradu slika. Stvaranje terena možemo shvatiti baš kao „slikanje“. Za izradu uvjerljive okoline u igrici treba se uložit izniman trud u izradu terena. Naravno najbolje rješenje za popunjavanje praznog prostora na terenu su drveća. Unityev alat za postavljanje drveća je vrlo sličan ostalim alatima, ali s nekoliko dodatnih opcija. Naime postavljanje drveća je zapravo dodavanje velikog broja 3D objekata koji su najčešće interaktivni. Moramo imati puno veću kontrolu nad takvim alatom zbog velikog utjecaja na izvođenje naše igrice. Promjenom svojstva Tree Density utječemo na gustoću raspodijele drveća. Naravno ako bi svako drvo izgledalo isto, a tek bilo iste visine i širine, ne bi dobili dojam realizma. Kako bi se riješio taj problem postoji svojstvo Tree Height i opcija Lock Width to Height. Ta dva rješenja omogućavaju kaotičniju raspodjelu drveća i nasumičnu visinu i širinu istih.

Slika 12 - Postavke alata za drveće

Navigacijska mreža je zbirka dvodimenzionalnih konveksnih poligona (poligonska mreža) koja definira koja su područja terena prohodna. Najčešće se koristi kako bi računalno kontrolirani likovi znali po kojim dijelovima terena mogu pristupiti, a kojima ne. Na terenu se mogu nalaziti neprohodni dijelovi kao što su lava, drveća ili druge prepreke. U tom slučaju računalno kontroliran lik mora izračunati kako doći do cilja zaobilaskom tih prepreka. U slučaju ovog projekta računalno kontrolirani likovi su neprijatelji, a cilj je doći do glavnog igraća. Način na koji je to izvedeno bit će objašnjeno kasnije u radu.

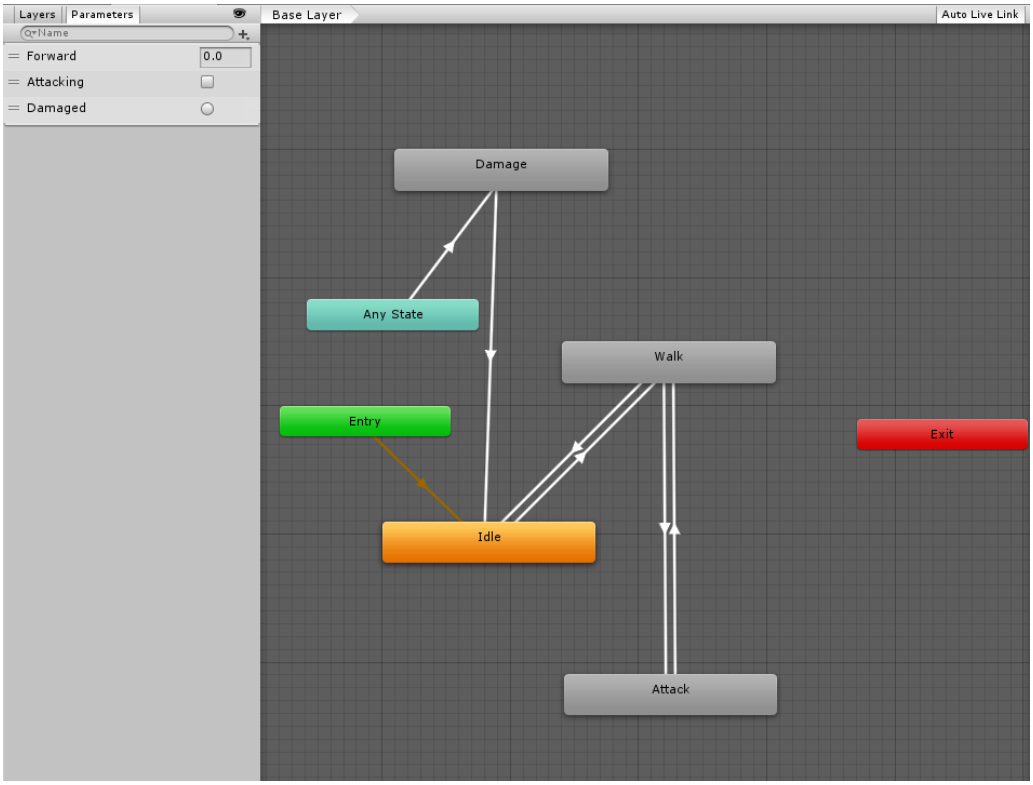


Slika 13 - Primjer Nav Mesha (Izvor: https://docs.unity3d.com/Manual/nav-CreateNavMeshAgent.html)

3.6. Animacije

Unity ima bogat i sofisticiran sustav animacija. Osnovni dio čine animacijski isječci. Animacijski isječak sadrži informacije kako bi se neki objekta trebao ponašati (mijenjati poziciju, rotaciju ili druga svojstva) tokom nekog vremena. Animacijski isječak je specifična animacija za neki model npr. trčanje humanoida ili čučanj. Animator je kontroler koji omogućava spajanje animacijskih isječaka u skup animacija koje taj model može napraviti. Kao primjer možemo uzeti kretnje nekog humanoida. U Animatoru će se nalaziti animacija za hodanje, trčanje, skakanje, čučanj, kretanje u čučnju itd. Svima animacija u animatoru pristupamo nekim uvjetom. Da bi se izvela animacija skakanja mora bit pritisnuta tipka „space“.

Cijeli tijek rada u Unityu se svodi na animacijske isječke. Njih možemo stvarati izravno u Unityu, Mayi ili pomoću snimanja pokreta (motion capture). Kako bi složili sve u jednu funkcionalnu cjelinu koristimo Animator controller koji povezuje sve animacijske isječke i slaže ih u cjelinu koja izgleda kao dijagram toka. Animator se ponaša kao stroj stanja koji prati koji animacijski isječak treba biti pokrenut. Kao primjer smo već naveli animaciju skakanja. Animatorova prednost je mogućnost miješanja dva animacijska isječka te se time dobiva fluidna tranzicija između animacija.

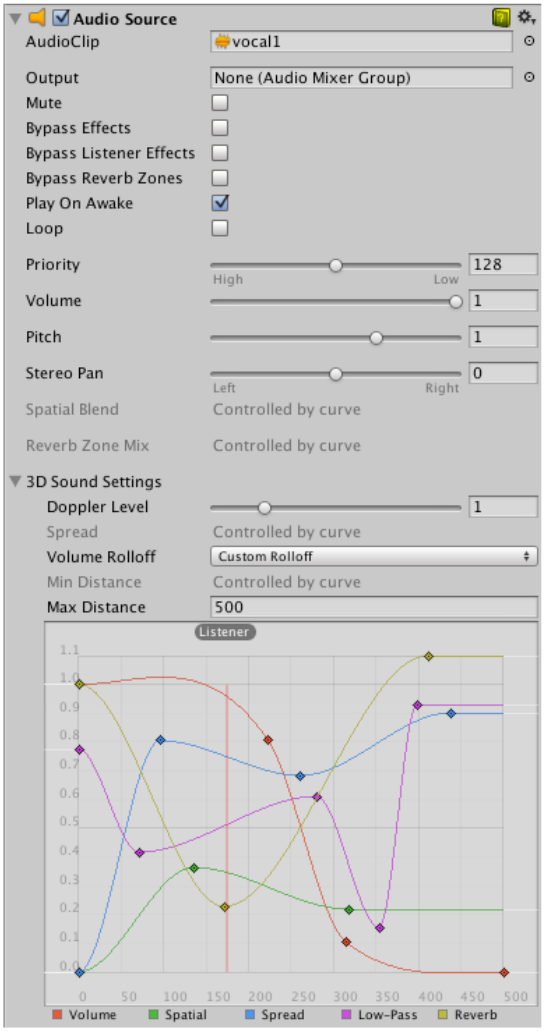


Slika 14 – Animator

3.7. Zvuk

Izrada vizualnih efekta je samo dio kod izrade igara. Zvuk igra veoma bitnu ulogu u izradi računalnih igara. Zvukovi u igrama stvaraju atmosferu, povratne informacije itd. Svaka kretnja i akcija u igrama treba biti potkrepljena zvučnim efektom. Time dobivamo dojam realnosti i težine te akcije.

Zvučne efekt u Unityu dodajemo kao assete. Na bilo koji objekt u igrici možemo dodati zvučni izvor (Audio Source). Svaki izvor ima nekoliko svojstava.



Slika 15 - Svojstva izvora zvuka (Izvor: https://docs.unity3d.com/Manual/class-AudioSource.html)

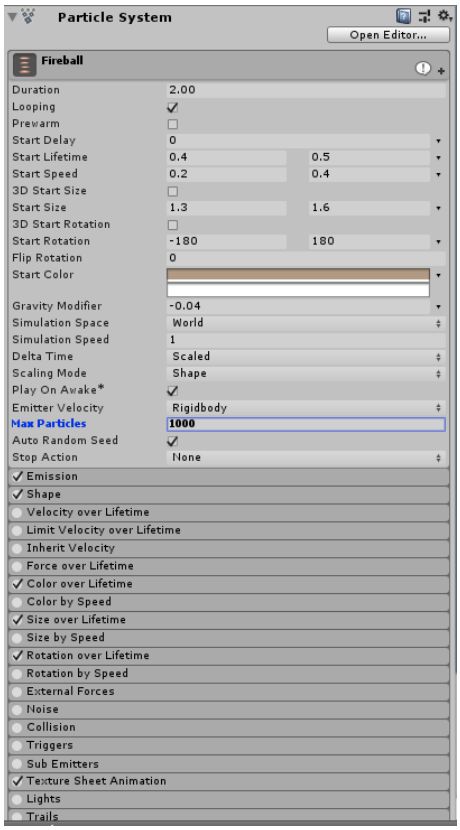
Zvučni izvor reproducira audio zapis u zadanoj sceni, zapis se može reproducirati na zvučnog slušatelja (Audio Listener) ili putem audio miksera. Zvučni izvor može reproducirati

bilo koju vrstu zvučnog zapisa i može se konfigurirati za reprodukciju kao 2D, 3D ili kao mješavina (SpatialBlend). [1] (https://docs.unity3d.com/Manual/class-AudioSource.html)

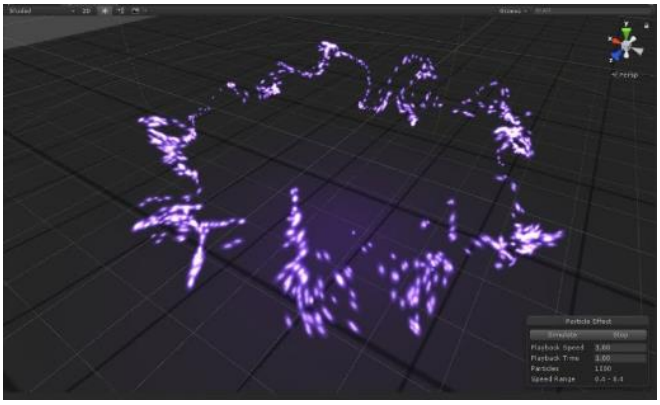
Zvučni slušatelj se ponaša kao mikrofon. On prima zvuk od bilo kojeg zvučnog izvora koji se nalazi u sceni i reproducira ih kroz zvučnike računala. U većini slučajeva taj slušatelj će se nalaziti u glavnoj kameri. Ako je zvučni izvor udaljen od slušatelja na njega se dodaju različiti efekti kako bi se simulirali razni efekti npr. Dopplerov efekt. Svaka scena može imati samo jednog slušatelja.

3.8. Čestice

Čestice su male, jednostavne sličice koje se prikazuju i pomiču u velikom broju. Sustavom čestica možemo imitirati brojne efekte koje nalazimo u svijetu. Kao primjer možemo uzeti dim ili vatru. Dim možemo imitirati na način da velik broj malih sliči dima, magle, oblaka reproduciramo sa velikim brojem izmjena. Svaka čestica ima predodređenu dužinu života, inače je to nekoliko sekundi, te je podložna promjeni veličine, rotacije, boje itd. tokom svog života.



Slika 16 - Svojstva čestica



Slika 17 - Čestice

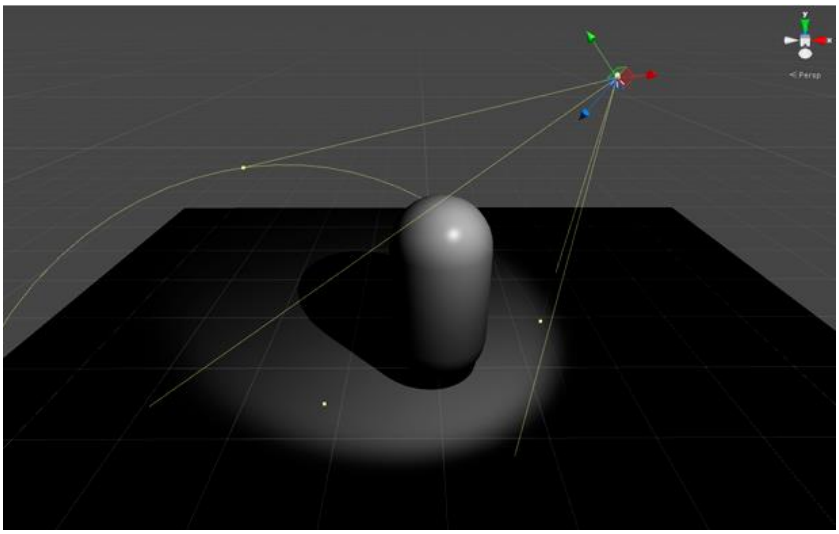
Čestice se mogu širiti u razne geometrijske likove kao što su kugla, stožac, kvadrat, itd. Čestice se dodaju klikom na glavni izbornik u GameObject -> Effects-> Particle System.

3.9. Osvjetljenje

U Unityu možemo smatrati da postoje dvije vrste osvjetljenja, u realnom vremenu i „precomputed“ osvijetljene. Svaka vrsta pojedinačno ili kombinacija njih može napraviti realističnu scenu.

3.9.1. Osvjetljenje u realnom vremenu

Osvjetljenje u realnom vremenu (Realtime Lighting) je osvjetljenje koje je dinamično i ažurira se sa svakim „frameom“. Directional, spot i point pripadaju toj vrsti svijetla. Pomicanjem objekata u sceni utječemo na svijetlo i sjenu koju baca. Nažalost u Unityu realno osvjetljenje ne podržava svjetlosnu reflekciju i refrakciju. U pomoć priskače unaprijed izračunato osvjetljenje.



Slika 18 - Osvjetljenje u realnom vremenu (Izvor: https://unity3d.com/learn/tutorials/topics/graphics/choosing-lighting-technique)

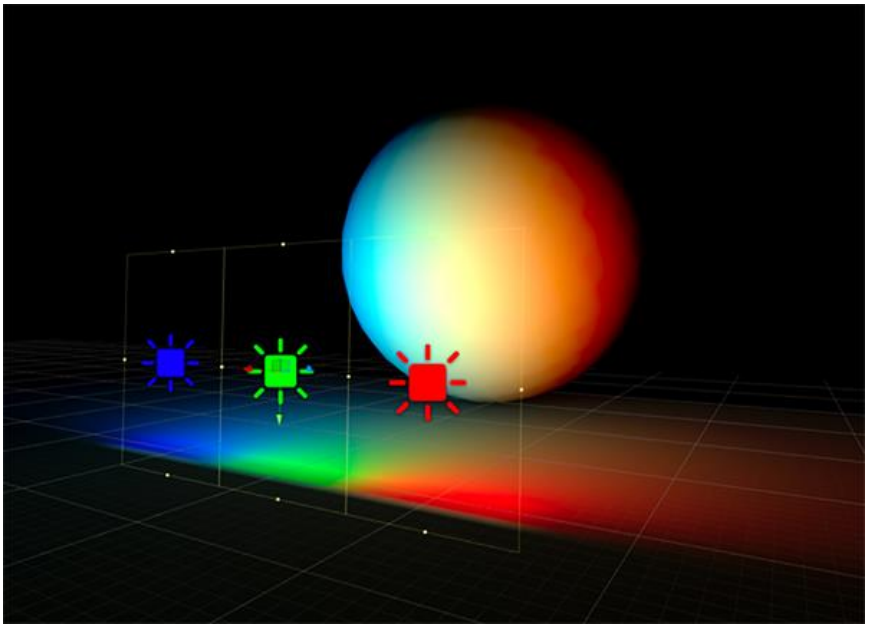
3.9.1.1. Vrste izvora svjetla

Directional - je izvor svjetla koji utječe na sve objekte u sceni. Postavljena su daleko od terena te zbog toga se koristi kao simulacija sunca.

Point – je izvor svjetla iz jedne točke. Svjetlo se širi radijalno te intenzitet opada linearno s udaljenošću. Nakon zadane udaljenosti svijetlo će imati intenzitet 0. Takva vrsta izvora se koristi kao simulacija za svjetiljke, vatru, eksplozije

Spot – je izvor svijetla stožastog oblika. Kao i point light ima ista svojstva, ali ne obasjava sve oko sebe već je ograničeno. Spot izvor se najčešće koristi kao simulacija ručne svjetiljke, svjetla auta itd.

Area – Area izvor je ograničen kvadratnim oblikom. Svjetlo se emitira jednako po cijeloj površini, ali samo s jedne strane površine. Pošto Area izvor obasjava objekt iz nekoliko izvora odjednom pod različitim kutovima dobivamo mekše sjenčanje.



Slika 19 - Area izvor (Izvor: https://docs.unity3d.com/Manual/Lighting.html)

3.9.2. Unaprijed izračunato osvjetljenje (Baked lighting)

Kada „ispečemo“ svjetlost efekti osvjetljenja se izračunavaju i zapišu se na teksture koje se zalijepe preko objekta. Na taj način se dobiva efekt da je objekt osvjetljen, ali ne dinamično. To znači ako bi pomakli taj objekt njegova sjena bi ostala na istoj poziciji. Naveli smo da nedostatak realno osvjetljenja je to što nema reflekcije i refrakcije. U baked osvjetljenju je to moguće. Time ostvarujemo uvjerljiviju scenu i bolji performance, ali sa statičnim objektima.

3.9.3. Unaprijed izračunato realno osvjetljenje

Unaprijed izračunato realno osvjetljenje uzima najbolje iz oba svijeta. Pošto ne možemo postići uvjerljivo osvjetljenje sa svjetlosti u realnom vremenu, a baked svijetlost ima problema sa pomicanjem objekata kao što je običan dan-noć ciklus tu uskače kombinacija obje solucije. Svjetlost u realnom vremenu stvara velik teret grafičkoj kartici. Pred izračunom svjetlosti za nekoliko objekata u sceni štedimo na velikom broju resursa, a za neke koji imaju mogućnost interakcije sa glavnim likom ćemo i dalje koristiti osvjetljenje u realnom vremenu.

Slika 20 - Primjer unaprijed izračunatog realnog osvjetljenja (Izvor: https://unity3d.com/learn/tutorials/topics/graphics/choosing-lighting-technique)

3.10. Kamera

Kao što se kamere koriste u filmovima da prikažu priču gledateljima tako kamera u Unityu prikazuje što vidi glavni lik u svijet u kojem se nalazi. Kako bi igra uopće bila funkcionalna moramo imati barem jednu kameru u sceni, naravno možemo ih imati više. Dobar primjer više kamera su igre u kojima imamo podijeljen ekran kako bi dva igrača mogla igrati. Kamere se mogu animirati kao i 3D objekti da se pokreću neovisno o igraču te time dobivamo dramatičniji efekt.

3.11. Usporedba Unitya s drugim engineima

3.11.1. Unity

Prednost Unitya nad ostalim engineima nije njegova moć ili mrežne performanse, već njegova jednostavnost i pristupačnost korisniku. Unity pruža velik raspon alata i značajki. Kako smo i već naveli u radu, Unity podržava development za velik raspon operativnih sustava. Korisnik u Unityu može raditi igru za Windows platformu te ju može objaviti za drugi operativni sustav s minimalnim preinakama. Unity podržava development za 25 platformi dok konkurenti kao Unreal 4 i CryEngine podržavaju oko 10. Unity ima veoma veliku zajednicu ljudi koji pridonese razvoju platforme. Korisnik može pridonijeti zajednici izradom asseta, tutoriala ili izravno pomoći na nekom projektu. Unity podržava velik broj različitih formata, te ih bez problema možemo implementirati u svoj projekt.

3.11.2. Unreal Engine 4

Unreal je napravila američka kompanija Epic Games. Unreal je grafički superiorni engine od Unitya. U zadnjih nekoliko mjeseci je doživio velik napredak u mrežnom djelu enginea. Naime njihova glavna igra Fortinite je imala razočaravajuće mrežne performanse te ih je to potaknulo da posvete više resursa za razvoj mrežnog sustava. Epic Games je i više nego uspio u tom pothvatu. Takav uspjeh je napravio veliku marketinšku reklamu za engine.

Velika prednost Unreala je njegov dinamičan sustav osvjetljenja i novi sustav čestica. Sustav može podržati čak milijun čestica u jednoj sceni.

Unreal je besplatan kao i Unity, ali je potrebno platiti 5% profita od prodaje igre. Naravno u današnjem svijetu ništa nije besplatno pa tako i Unreal je potrebno plaćati ako prihodi prelaze 3000$ po kvartalu.

Unreal je standardizirao revolucionaran način razvijanja igara. Blueprint sistem omogućava izradu video igara koristeći samo blueprinte. Time omogućavaju i ljudima koji nisu upoznati s programiranje da razvijaju svoje igrice. Ali tako revolucionarno rješenje je onemogućilo izradu igara za starije platforme.

Slika 21 - Showcase Unreala 4 (Izvor: https://wall.alphacoders.com/big.php?i=271762)

4. Izrada igre u Unityu

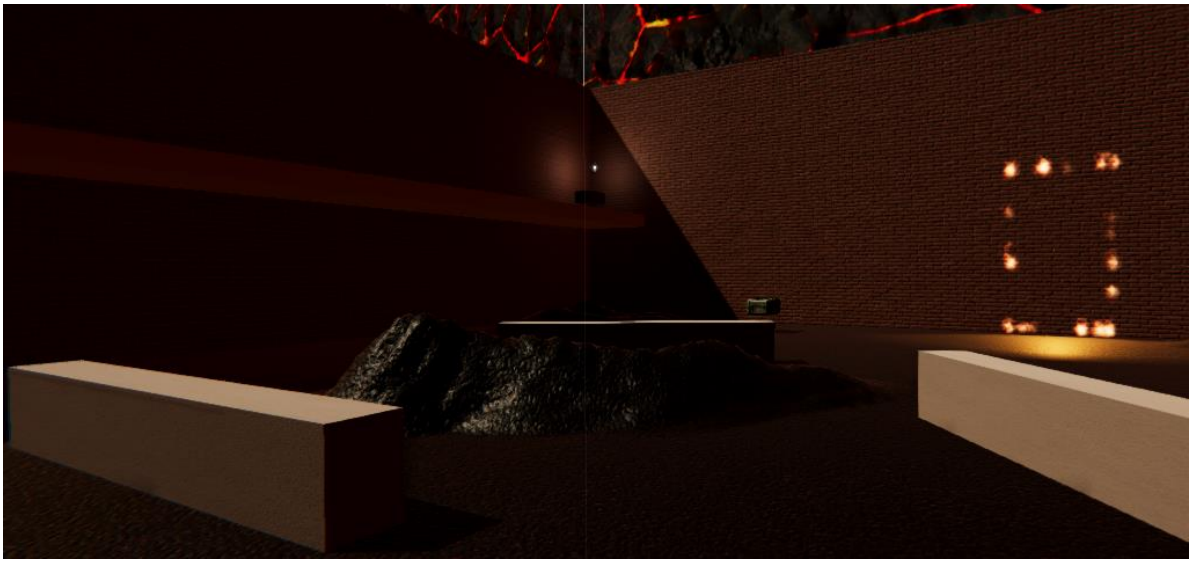
Naravno čak i prije instaliranja bilo kakvog programa za izradu igara potrebno je utvrditi što točno želimo postići tim projektom. U mom slučaju je to akcijska igra sa elementima pucanja. Prateći tu logiku dolazimo do zaključka da će naša igrica imati glavnog lika, neprijatelje te ostale elemente i skripte pucnjave, životne bodove neprijatelja i glavnog lika, streljivo, te razna oružja. Inspiracija za ovaj rad je bila igra DOOM (2016.)

Prvi korak u izradi ovog projekta jest bila izrada funkcionalnog glavnog lika, na sreću Unity u sebi sadrži već predefinirane objekte(u nastavku FPSController). U sebi sadrži skriptu koja je zadužena za pomicanje navedenog objekta (First Person Controller Script). Audio izvor koji svira dok se objekt pokreće. Taj originalni objekt ćemo kasnije nadograđivati s mnoštvo drugih funkcionalnosti.

Kako bi igrač mogao vidjeti što se zapravo događa u igrici potrebna mu je „kamera“. Kamera je također objekt koji nema „collider“. Odnosno ostali fizički objekti neće utjecati na kameru, može prolaziti kroz zidove, itd. Ali kako bi spriječili takva neželjena ponašanja i naravno kako bi mogli kontrolirati našeg glavnog lika „spojit“ ćemo kameru i FPSController zajedno. Kada kažemo spojit mislimo na to da ćemo objekt kamere učiniti djetetom objekta FPSController. Tim postupkom smo dobili glavnog lika koji se može kretati i gledati. Sada dolazimo do drugog problema, a to je izrada mape na kojoj se objekti mogu nalaziti i kretati.

4.1. Izrada mape (Arene)

U ovoj vrsti igrice nemamo potrebe za velikim terenom. Početnu plohu terena smo napravili s dimenzijama 100x100. Preostale postavke terena su ostale nepromijenjene. Kako glavni lik i neprijatelji ne bi pali sa ruba mape postavili smo velike zidove oko terena. Objekt kvadrata je savršeno poslužio za takvo rješenje. Kako bi malo dodali dinamičnosti terenu pojedine kutove smo uredili alatom „Raise/Lower Terrain“.



Slika 22 - Primjer uporabe alata

Ostatak mape smo uredili sa nekoliko prepreka (zidića) i povišenih platformi kao što je vidljivo na slici 21.

Arenu smo podijelili na dva jednaka dijela pomoću velikog zida sa prolazom. Cilj toga je razbiti monotonost jedne velike prostorije te prikazati da Nav Mesh uredno funkcionira i na kompliciranijim terenima. Ispod platforme s dvije kule napravljeno je jezerce s vodom da se prikaže jednostavnost korištenja shadera za vodu.

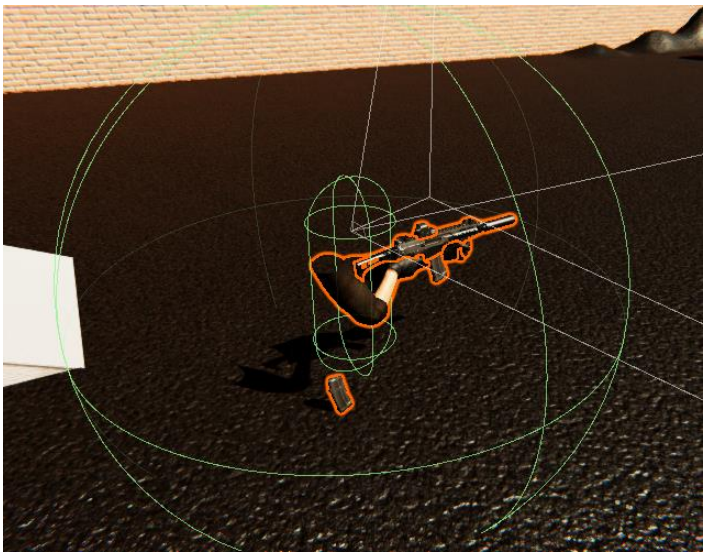
Slika 23 - Jezerce ispod platforme

Nakon izrade terena potrebno je „ispeći“ Nav Mesh. Klikom na objekt Terrain otvara se inspektor na desnoj strani prozora. Pokraj kartice inspektor se nalazi kartica Navigation

koju trebamo pritisnuti. Odabiremo Bake te nailazimo na nekoliko postavki. Određujemo veličinu i visinu „ispečenog“ agenta te koliki je najveći nagib po kojem se agent može penjati. Ako postoje prepreke pred agentom možemo odrediti najveću veličinu koju može preskočiti. Klikom na bake će se pojaviti plavi overlay preko terena koji nam pokazuje gdje se neprijatelji mogu kretati. Nakon pregleda Nav Mesha možemo preciznije definirati nekakve postavke te pokušati ponovno „ispeći“ mrežu ako do sada nismo bili zadovoljni. Sada možemo bilo kojem objektu u igri dodijeliti svojstvo Nav Mesh Agenta. Kasnije ćemo objasniti kako je to sve povezano u jednu cjelinu.

4.2. Glavni lik

Kako smo već naveli da za izradu glavnog lika je potreban prefab FPSController. Na izradu glavnog lika će najviše utjecati tema i zamisao igre. U ovom projektu sam se fokusirao na glavnog lika koji ima sustav zdravlja, pucanja, trzaj pucanja i kretanja. Naš glavni lik ima nekoliko različitih pušaka pa ne možemo skriptu za pucanje postavit na objekt glavnog lika, već na pušku. Svaka puška ima različita svojstva kao što su šteta, brzina pucanja i jačina trzanja puške. Kako bi glavni lik imao pušku dodajemo objekt u FPSController koji ćemo nazvati Weapon Holder. Objekt Weapon Holder sadrži skriptu Weapon Switch koja je vrlo jednostavna. Ako korisnik zavrti kotačić miša prolazimo kroz svu djecu te zavisno o njihovom rednom broju odabire se koja puška će biti aktivna, a ostale postavljamo kao neaktivne. Objekt puške u sebi sadrži animirani 3D model , sustav čestica i objekt DMGObject koji sadrži skriptu za pucanje.



Slika 24 - Glavni lik s colliderima

Pokretanjem skripte poziva se metoda Start() koja inicijalizira objekt kamere te na njega dodaje komponentu trzaja nakon pucnja puške. Metoda Update() se poziva nakon svake osvježene sličice (frame). Nadalje je bitna logika kako smo zamislili pucnjavu u našoj igrici. U ovom projektu je to odrađeno da metoda Update() provjerava je li pritisnuta lijeva tipka miša. Ako je pritisnuta zovemo metodu Shoot() sve dok korisnik ne pusti tipku miša. Metoda Shoot() provjerava ima li dovoljno streljiva za trenutno odabranu pušku. Svaki puta kada pozovemo metodu Shoot() pokrećemo trzanje pucnja, audio zapis pucnja, okidač animacije i sustav čestica. Nakon toga se poziva funkcija Raycast koja „baca“ zraku, od točke izvorišta, u zadanom smjeru te provjera je li došlo do sudara s nekim objektom koji sadrži collider. Ako je pogođeni objekt u dometu pozivamo metodu pogođenog objekta i šaljemo toj metodi iznos štete koji je primio neprijatelj.

```
    void Update () {
        if (Input.GetButtonDown("Fire1"))
        {
            InvokeRepeating("Shoot", 0f, 1f / firerate);
        }
        else if (Input.GetButtonUp("Fire1"))
        {
            CancelInvoke("Shoot");
        }
    }
    void Shoot()
    {
        if (ammo.CurrentRifleAmmo > 0)
        {
            recoilComponent.StartRecoil(0.2f, 15f, 5f);
            AudioSource gun_sound = GetComponent<AudioSource>();
            gun_sound.Play();
            animator.ResetTrigger("pucanjAK");
            animator.SetTrigger("pucanjAK");
            muzzleFlash.Play();

            ammo.CurrentRifleAmmo--;
            RaycastHit shot;
            If (Physics.Raycast(transform.position,
transform.TransformDirection(Vector3.forward), out shot))
            {
                udaljenost = shot.distance;
                if (udaljenost < dopustenaUdaljenost)
                {
                    shot.transform.SendMessage("smanjiZivot",damageAmount,
SendMessageOptions.DontRequireReceiver);
                }
                if (shot.rigidbody != null) {
                    shot.rigidbody.AddForce(-shot.normal*200);
                }
            }


        }
    }
```

Programski kod 1: Primjer logike pucanja

4.3. Klasa GlobalAmmo

Kako što smo i naveli moramo pratit koliko naš glavni lik ima streljiva za koristiti. Svakim klikom lijeve tipke miša moramo smanjiti određeno streljivo za jedan. U prethodno navedenoj skripti pozivamo klasu GlobalAmmo te jedan od njenih atributa provjeravamo je li manji od 0 te ako nije smanjujemo ga za jedan.

4.4. Sustav životnih bodova

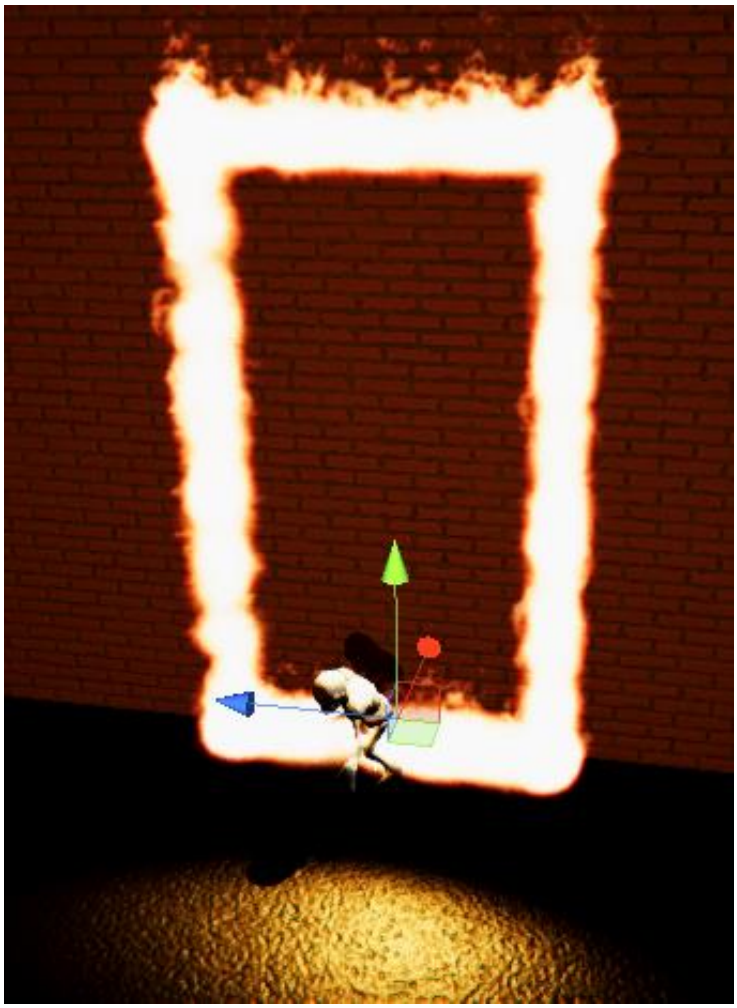
Sustav životnih bodova je jedno vrlo jednostavno rješenje. Sve što nam je potrebno je jedna skripta koja će se nalaziti u FPSControlleru. Tu skriptu smo nazvali Player Health. Skripta je sastavljena u dva dijela, stamina i životni bodovi. Glavni lik se stvara sa 100 životnih bodova, te može primati štetu od protivnika. Svaki frame se provjerava ima li glavni lik manje ili jednako nuli životnih bodova. Ako životni bodovi padnu tako nisko pozivamo metodu Death() gdje zaustavljamo igricu, glazbu i aktiviramo panel na kojem piše Game Over. Na tom istom canvasu se nalazi gumb Exit te on gasi igricu. Stamina je broj koji dozvoljava brzo trčanje glavnog lika. Naravno ako taj broj bude 0 glavni lik više ne može trčati. Stamina se smanjuje ako se korisnik pomiče i ima pritisnutu tipku „shift“ na tipkovnici. Stamina se smanjuje zavisno o proteklom vremenu od zadnjeg framea. Zbog takve implementacije stamina će uvijek opadati istom brzinom bez obzira koliko sličica dobivamo u jednoj sceni.

```
public void TakeDamage(int amount)
    {
        damaged = true;
        currentHealth -= amount;
        healthSlider.value = currentHealth;
        if (currentHealth <= 0 && !isDead)
        {
            Death();
        }
    }
    void Death()
    {
        isDead = true;
        glazba.GetComponent<AudioSource>().Pause();
        panelGameOver.SetActive(true);
        Time.timeScale = 0;
           }
}
```

Programski kod 2: Primanje štete

4.5. Stvaranje neprijatelja

Prije početka izrade igrice bitno je odrediti cilj igre, odnosno što želimo raditi u njoj. U ovom projektu to je čisto ubijanje neprijatelja i preživljavanje. Kako bi se naš heroj mogao boriti protiv neprijatelja potrebno ih je stvoriti. Ako bi ih sve odjednom stvorili igra ne bi funkcionirala. Prvi problem bi bile performanse, rijetko koje računalo bi moglo imati više od 15 frameova po sekundi ako bi stvorili sve neprijatelje od jednom, a drugi problem je što bi nas ti neprijatelji vrlo brzo ubili. Jednostavno rješenje tomu je dinamično stvaranje neprijatelja svakih nekoliko sekundi. Stvorimo nekoliko praznih objekata (u slučaju ovog projekta 4 komada) te njima dodijelimo skriptu SpawnPoint. Skripta SpawnPoint koristi metodu koja se poziva svakih nekoliko sekundi. Pozivom te metode se instancira zadani objekt, u ovom slučaju neprijatelj. Vremena stvaranja smo napravili nasumično kako bi izbjegli stvaranje 4 neprijatelja odjednom što dovodi do zamrzavanja igre na nekoliko sekundi.



Slika 25 - Vatrena vrata stvaraju neprijatelja

4.6. Neprijatelji

Neprijatelji su objekti koji pokušavaju poraziti glavnog lika. U ovom projektu smo koristili asset kostura sa mačem. Srećom dolazi sa kvalitetnim animacija trčanja, udaranja mačem i primanja štete pa se ne treba potrošiti puno vremena na to. Kako bi se naš neprijatelj mogao kretati dodali smo mu Third Person Character scriptu. Većinu metoda i atributa nećemo koristi jer ne kontroliramo neprijatelja mi već računalo. Iz navedene skripte ćemo samo koristiti metodu koja provjerava pomiče li se objekt ili stoji na mjestu. Ako se pomiče u animatoru izazivamo animaciju hodanja. Sljedeći korak je usklađivanje animacije i brzine hodanja kako bi izgledale uvjerljivo.

4.6.1. Nav Mesh Agent

Nav Mesh Agent je komponenta koja je se nalazi unutar 3D modela neprijatelja. U njoj određujemo kako će se ponašati naš neprijatelj dok se nalazi na „ispečenom“ Nav Meshu. Neprijatelj može skretati te postoji nekoliko svojstava koje možemo modificirati kako bi ga prilagodili našoj vrsti igre.

Veličina agenta	
Radius	Radijus agenta, korišten za računanje sudara između prepreka i drugi agenata.
Height	Visina agenta, koju koristimo za izračun može li agent proći ispod prepreke.
Skretanje	
Speed	Maksimalna brzina kretanja po sekundi.
Angular Speed	Najveća brzina okretanja.
Acceleration	Najveće ubrzanje po sekundi.
Stopping distance	Udaljenost na kojoj će se agent zaustavit od cilja.
Izbjegavanje prepreka	
Quality	Kvaliteta izbjegavanja prepreka. Ako imamo velik broj prepreka i agenata ponekad je potrebno smanjiti kvalitetu izbjegavanja zbog velikog tereta na

	CPU. Ako smanjimo kvalitetu agenti će i dalje zaobilaziti prepreke, ali neće u kalkulaciju uzimati ostale agente.

Već smo nekoliko puta spomenuli kako se naš agent kreće do cilja, ali nigdje nismo naveli kako on zna što mu je cilj. To rješavamo implementacijom skripte AICharacterControl.

Pozivom FindGameObjectWithTag("Player").transform saznajemo rotaciju, veličinu i poziciju. Tijekom metode Update() ažuriramo cilj agentu pozivom agent.SetDestination (target.position).

```
  private void Awake()
        {
            target = GameObject.FindGameObjectWithTag("Player").transform;
        }
        private void Start()
        {
                          agent =
GetComponentInChildren<UnityEngine.AI.NavMeshAgent>();
            character = GetComponent<ThirdPersonCharacter>();

               agent.updateRotation = false;
               agent.updatePosition = true;
        }
        private void Update()
        {
            if (target != null)

                agent.SetDestination(target.position);

            if (agent.remainingDistance > agent.stoppingDistance)
            {
                agent.isStopped = false;
                character.Move(agent.desiredVelocity, false, false);
            }
            else
                agent.isStopped = true;
        }
        public void SetTarget(Transform target)
        {
            this.target = target;
        }
```

Programski kod 3: Logika pronalaženja glavnog lika

Zadnja stvar koju neprijatelj mora raditi jest pokušati poraziti glavnog lika. U ovom projektu sam se odlučio za vrlo jednostavan sustav napadanja. Jedan collider se nalazi na neprijatelju, a drugi na glavnom liku. Ako se ta dva collidera dotaknu i budu dotaknuti neko vrijeme glavnom liku će se oduzeti životni bodovi. Nakon toga je potrebno uskladiti animaciju napadanja kako bi sve zajedno izgledalo uvjerljivo.

```
Void Awake()
    {
        player = GameObject.FindGameObjectWithTag(„Player“);
        playerHealth = player.GetComponent<PlayerHealth>();
        anim = GetComponent<Animator>();
    }

    void OnTriggerEnter(Collider other)
    {

        if (other.gameObject == player)
        {
            playerInRange = true;
            anim.SetBool(„Attacking“, true);
            timer = 0.7f;
        }
    }
    void OnTriggerExit(Collider other)
    {
        if (other.gameObject == player)
        {
            playerInRange = false;
            anim.SetBool(„Attacking“, false);
            timer = 0f;
            timeBetweenAttacks = 2f;
        }
    }

    void Update()
    {
        timer += Time.deltaTime;

        if (timer >= timeBetweenAttacks && playerInRange)
        {
            Debug.Log(timer);
            Attack();
            timeBetweenAttacks = 2.7f;
        }
        if (playerHealth.currentHealth <= 0)
        {
            anim.SetTrigger(„PlayerDead“);
        }
    }
    void Attack()
    {
        timer = 0f;

        if (playerHealth.currentHealth > 0)
        {
            playerHealth.TakeDamage(attackDamage);
        }
    }
```

Programski kod 4: Logika napadanja neprijatelja

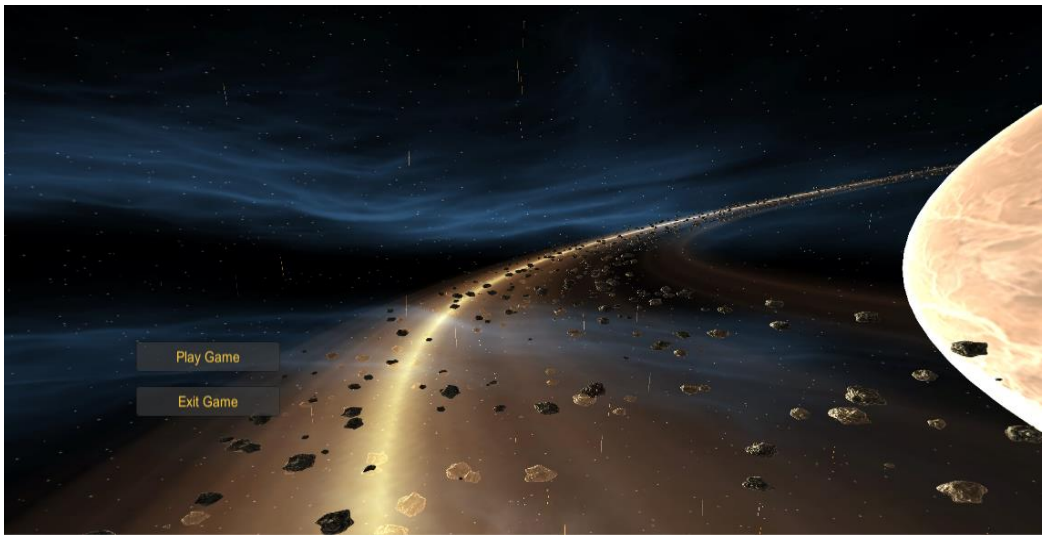
4.7. UI

UI je zapravo samo jedan sloj dinamične slike postavljen preko kamere. UI ovog projekta je poprilično jednostavan. U lijevom kutu se nalazi traka životnih bodova, a desno od nje je traka stamine. U desnom kutu se nalazi pokazivač streljiva.

Obje trake su klizači (Slider) te imaju maksimalnu vrijednost 100. U prije navedenim skriptama gdje smo spominjali životne bodove i staminu se također nalaze metode koje mijenjaju vrijednost ovih klizača. Za streljivo smo napravili poseban objekt na kojem se nalazi skripta GlobalAmmo. Skripta sadrži početne vrijednosti svih vrsta streljiva, te u metodi Update() ažurira vrijednosti na canvasu.

4.8. Main Menu

Prvo što igrač vidi u ovoj igrici jest početna stranica. Kako bi napravili main menu moramo imati novu scenu. Menu je zapravo canvas sa nekoliko gumbova. Dodan je i sustav čestica kako bi simulirali atmosferu vulkanskog planeta. Ovdje je dovoljna samo jedna skripta koja je zadužena za gašenje igre ili mijenjanje scene, odnosno učitavanja „levela“.



Slika 26 - Main Menu

5. Zaključak

Rad je podijeljen na tri bitne cjeline. Povijest izrade video igara bez kojih najvjerojatnije ne bi mogli raditi to što radimo danas. Objašnjenje i opis Unity enginea u kojem je praktični dio napravljen te opis proces izrade igre. Detaljno je opisana logika izrade računalne igre, korištenje svojih, a i javnih asseta te izrada sučelja. Cilj izrade igre jest upoznavanje proces koji moramo pratiti kako bi dobili kvalitetan proizvod, upoznavanje svakog djela programa i njegovih alata.

Većina 3D modela je preuzeta sa Unity Asset Storea ili su napravljeni ručno(jednostavniji modeli).

Izrada računalnih igara je vrlo kompliciran, a u istom vrijeme jednostavan proces. Izradom ove igrice shvatio sam da se svatko može okušati u developmentu računalnih igara. Prije bilo kakve izrade je uvijek potrebno točno utvrditi što želimo napraviti.

6. Literatura

[1] Graetz, J. M. (kolovoz 1981) The origin of Spacewar Creative Computing. Vol. 6

[2] Unity: Unity 2018.2.3f1 (2018.)
Dostupno na: https://unity3d.com/unity/whatsnew/unity-2018.2.3, kolovoz 2018.

[3] Unity: Unity Documentation (2018.)
Dostupno na: https://docs.unity3d.com/Manual/index.html, kolovoz 2018.

[4] YouTube: Brackeys (2017. – 2018.)
Dostupno na: https://www.youtube.com/user/Brackeys, kolovoz 2018.

[5] Unity: Importing (2018.)
Dostupno na: https://docs.unity3d.com/Manual/ImportingAssets.html, kolovoz 2018.

[6] Unity: Scripting (2018.)
Dostupno na: https://docs.unity3d.com/Manual/ScriptingSection.html, kolovoz 2018.

[7] Unity: Animation (2018.)
Dostupno na: https://docs.unity3d.com/Manual/AnimationSection.html, kolovoz 2018.

[8] Unity: Animation Clips (2018.)
Dostupno na: https://docs.unity3d.com/Manual/AnimationClips.html, kolovoz 2018.

[9] Unity: UI (2018.)
Dostupno na: https://docs.unity3d.com/Manual/UISystem.html, kolovoz 2018.

[10] Unity: Canvas (2018.)
Dostupno na: https://docs.unity3d.com/Manual/UICanvas.html, kolovoz 2018.

[11] Unity: Navigation and Pathfinding (2018.)
Dostupno na: https://docs.unity3d.com/Manual/Navigation.html, kolovoz 2018.

[12] Unity: Building a NavMesh (2018.)
Dostupno na: https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html, kolovoz 2018.

[13] Unity: Audio Clip (2018.)
Dostupno na: https://docs.unity3d.com/Manual/class-AudioClip.html, kolovoz 2018.

[14] Unity: Audio Listener (2018.)
Dostupno na: https://docs.unity3d.com/Manual/class-AudioListener.html, kolovoz 2018.

[15] Unity: Audio Source (2018.)
Dostupno na: https://docs.unity3d.com/Manual/class-AudioSource.html, kolovoz 2018.

[16] Unity: Lighting overview (2018.)
Dostupno na: https://docs.unity3d.com/Manual/LightingInUnity.html, kolovoz 2018.
[17] Unity: Console Window (2018.)
Dostupno na: https://docs.unity3d.com/Manual/Console.html, kolovoz 2018.
[18] Unity: Types of light (2018.)
Dostupno na: https://docs.unity3d.com/Manual/Lighting.html, kolovoz 2018.

7. Popis slika

8. Popis programskih kodova