PROJEKTIRANJE 3D MODELA VATRENOG ORUŽJA U BLENDER-U I UNITY-U

Tomašković, Tomislav

Veleučilište u Karlovcu
Strojarski odjel
preddiplomski stručni studij *Mehatronike*

Tomislav Tomašković

Projektiranje 3D modela vatrenog oružja U Blender-u i Unity-u

3D modeling of firearms in Blender and Unity

Završni rad

Karlovac, 2021. godina

Veleučilište u Karlovcu
Strojarski odjel
preddiplomski stručni studij *Mehatronike*

Tomislav Tomašković

Projektiranje 3D modela vatrenog oružja U Blender-u i Unity-u

3D modeling of firearms in Blender and Unity

Završni rad

mentor: mr. sc. Vedran Vyroubal

Karlovac, 2021. godina

S punom odgovornošću izjavljujem da sam završni rad izradio samostalno, služeći se navedenim izvorima podataka. Zahvaljujem se profesoru i mentoru *mr.sc. Vedranu Vyroubalu* na svim savjetima i pružanju stručne pomoći tijekom izrade rada.

Tomislav Tomašković

Sažetak

U okviru rada izradit će se 3D simulacijska igra streljane. Izradit će se modeli vatrenog oružja i prostor u kojem će se modeli koristiti. Cijeli proces podijeljen je u nekoliko dijelova koji će biti opisani kroz sažeti koncept.

Prvi dio bio je napraviti 3D modele u programu naziva *Blender* [1]. *Blender* je jednostavan te je odličan za početnike, ali i napredne korisnike. Koristi se za puno stvari, ali najviše za kreiranje modela, postavljanje tekstura i animaciju. Prilagođen je za mnoge programe za kreiranje video igara (*Game Engine*).

Drugi dio je izrada jednostavne simulacije u *Unity game engine-u* [2]. Korišten je *Unity* jer je dobar game engine opće namjene (”general purpose game engine”) koji ima vrlo dobru podršku za 2D i 3D vizualizaciju, te posjeduje vrlo sposoban programski sklop za simuliranje fizike stvarnog svijeta (“physics engine“). Cilj je bio implementirati napravljene 3D modele da se pokaže kako se modeli koriste za realizaciju s dodanom stvarnom fizikom prostora.

U *Unity-u* se programira logika koristeći njihove biblioteke.

Summary

As part of the work, a 3D shooting simulation game will be created. Models of the firearms and the space in which the models will be used will be made. The whole process is divided into several parts which will be described through a concise concept.

The first part was to make 3D models in program called Blender [1]. The Blender is simple and great for beginners as well as advanced users. It is used for a lot of things, but mostly for modeling, texturing and animation. Iti s adapted for any program for creating video games (Game Engine).

The second part is to create a simple simulation in the Unity game engine [2]. Unity is used because it is a great general purpose engine, which works well with 2D and 3D games, simulations etc. And among other things, has a strong physics engine that is welcome. The goal was to implement the created 3D models to show how the models are used for realization with added real space physics.

In Unity, logic is programmed using their libraries.

Sadržaj

Popis slika:

1.Uvod

Tema ovog završnog rada je izrada 3D računalne igre namijenjena za isprobavanje izrađenih modela vatrenog oružja. Igra je osmišljena tako da igrač može uzeti jedno od ponuđenih oružja i vidjeti njegove specifikacije isprobavši ga na metama u izrađenoj streljani. Cilj igre je pokazati prednosti i nedostatke svakog oružja. Svako oružje treba proučiti prije izrade modela jer se razlikuje po izgledu i specifikaciji. Potrebno je prebacivanje modela u pravi format jer neki formati mogu sadržavati više podataka što sam model čini kvalitetnijim. Nužno se upoznati s objektnim programiranjem i izraditi skripte koje će kontrolirati ponašanje ubačenog modela.

Razlog odabira ove teme je proširivanje znanja iz 3D modeliranja i programiranja stečenog na fakultetu. Želja za stvaranjem i korištenjem nečeg svojeg.

2. Blender

Jedan od mnogih softverskih alata korišten za kreiranje 3D modela, animiranih filmova, vizualnih efekata, 3D aplikacija i slično. Razlog zašto je *Blender* [3] tako popularan među ostalim alatima je upravo zbog toga što je program otvorenog koda (eng. *„open source“*) te je time pristupačan svakome na korištenje, proširivanje i modificiranje.

Besplatan je i jednostavan pa je idealan za početnike, a i za nekoga tko je ozbiljnije u 3D modeliranju. Osim što ima mnogo opcija koje drugi alati nemaju, već se i konstantno nadograđuje od stotinu ljudi diljem svijeta koje zajedno povezuje cilj da program bude što bolji te da je potpuno besplatan. Osim Windows operacijskog sustava podržani su i Linux te MacOS operacijski sustavi.

Blenderove mogućnosti su: 3D modeliranje (eng. *„3D modeling“*), UV *mapiranje* (eng. *„UV unwrapping“*), *postavljanje tekstura* (eng. *„texturing“*), raster grafičko uređivanje (eng. *„raster graphics editing“*), kosturna animacija (eng. *„rigging and skinning“*), simulacije fluida i dima (eng. *„fluid and smoke simulations“*), simulacije čestica (eng. *„particle simulations“*), simulacija mekanog tijela (eng. *„soft body simulation“*), digitalno oblikovanje (eng. *„sculpting“*), animiranje (eng. *„animating“*), referentno pomicanje (eng. *„match moving“*), *renderiranje* (eng. *„rendering“*), pokretna grafika (eng. *„moving graphics“*), video uređivanje i *komponiranje* (eng. *„video editing and compositing“*).

2.1 Povijest

Holandski animacijski studio NeoGeo počeo je razvijati *Blender* kao vlastitu aplikaciju. Prva verzija pojavljuje se u siječnju 1998. godine. NeoGeo je kasnije zatvoren, a klijentske ugovore je preuzela druga kompanija. Nakon zatvaranja NeoGeo, Ton Roosendaal osnovao tvrtku Not a Number Technologies (NaN) koja je nastavila razvijati *Blender*, ali je NaN bankrotirao 2002 godine što je također značilo prestanak razvoja *Blendera*.

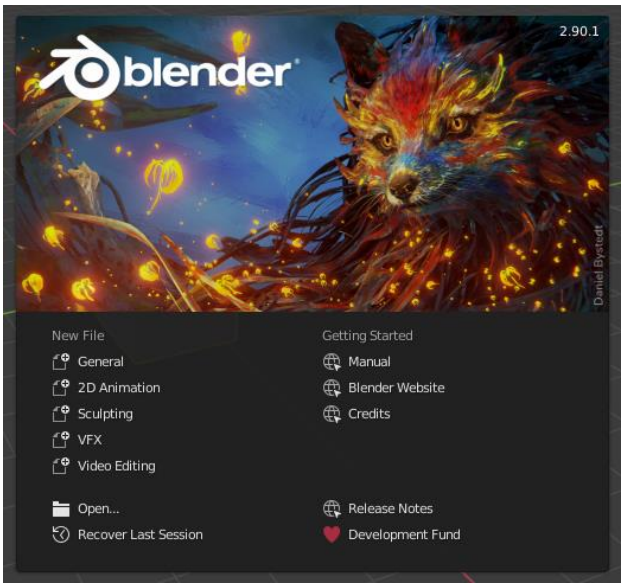
U svibnju 2002, Roosendaal je počeo neprofitnu organizaciju *Blender* Foundation s ciljem da nađe način za njegov razvoj te da ga učini „*open source*“ projektom. Danas *Blender* je besplatan i softver otvorenog koda koji iz dana u dan razvija zajednica. Posljednja aktualna i stabilna verzija *Blendera* (v.92.0) izašla je 25.2. 2021.

Programski jezici koji su pomogli u stvaranju ovoga softverskoga programa su C [5], C++ i Python. Podržan je na nekoliko operativnih sustava koje tvore, Windows, Linux i MacOS, a dostupan je u 32 i 64 bitnim varijantama. Autorstvo ovoga programa pripada nizozemskom animacijskom studiju NeoGeo, glavni autor bio je software developer i suvlasnik tvrtke Ton Roosendaal.

2.2 Pokretanje

Prilikom pokretanja [6] Blender-a, otvara se početni ekran koji sadrži informacije o trenutnoj verziji, novim nadogradnjama, pomoći te nas može uputiti na Internet stranicu *Blender-a*. Početni zaslon prikazan je na slici 1.

Početni zaslon pruža opcije za stvaranje novog projekta ili otvaranje već postojeće nedavno korištene *blend-datoteke*.



Slika 1: Početni zaslon (Splash Screen) [6]

2.3 Elementi Sučelja

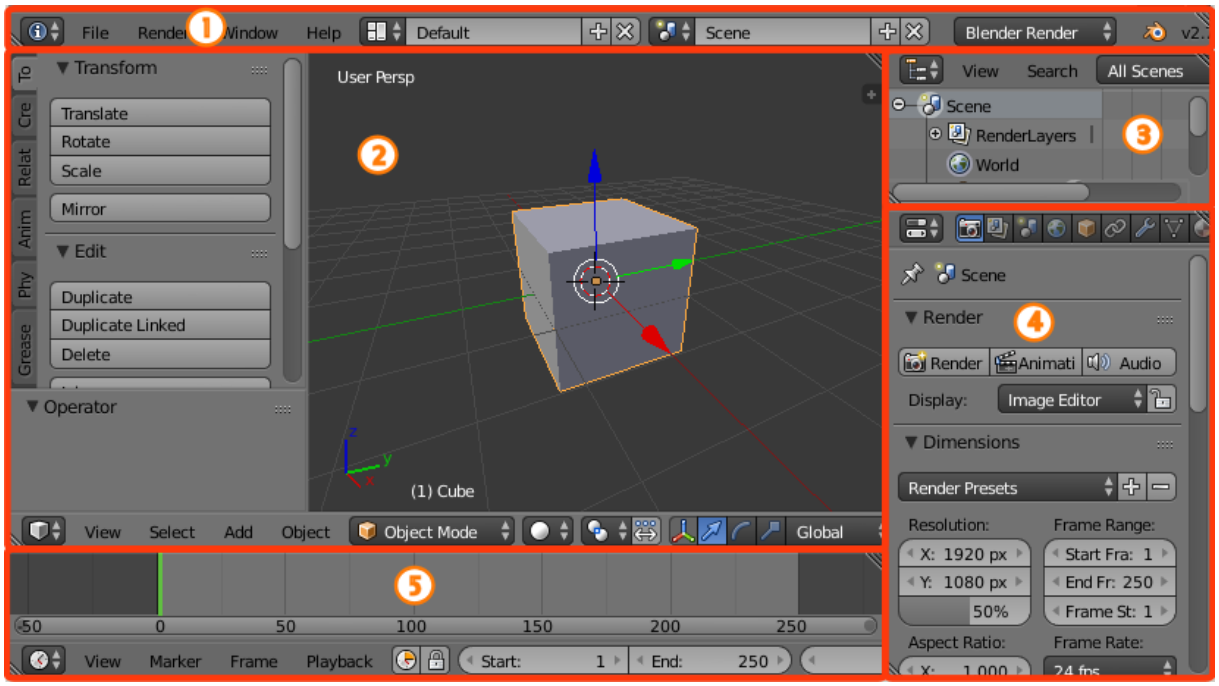
Prozor (*Window*), Zaslon (Screen), Područja (*Areas*), Editori (*Editors*), Regije (*Regions*), Tabovi (*Tabs*), Paneli (*Panels*), Kontrole (*Controls*).

Sučelje se može prilagoditi po potrebi zadatka ili ukusa. (slika 2)

2.4 Standardni zaslon

Po standardu *Blendera*, zaslon je podijeljen na 5 sekcija koji sadržavaju uređivače (eng. *„Editors“*):

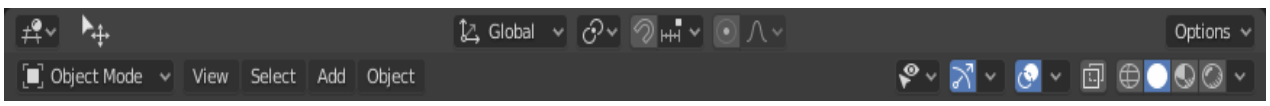
1) Informacijski *uređivač* na vrhu (*Info*)
2) Veliki 3D preglednik (*3D View*)
3) Vremenska linija na donjem dijelu (*Timeline*)
4) Lista objekata korišteni u *Blenderu* na desnome dijelu (*Outliner*)
5) Uređivač svojstava dole desno (*Properties Editor*)



Slika 2: Standardno korisničko sučelje

2.5 Zaglavlje (eng. *„Header“*)

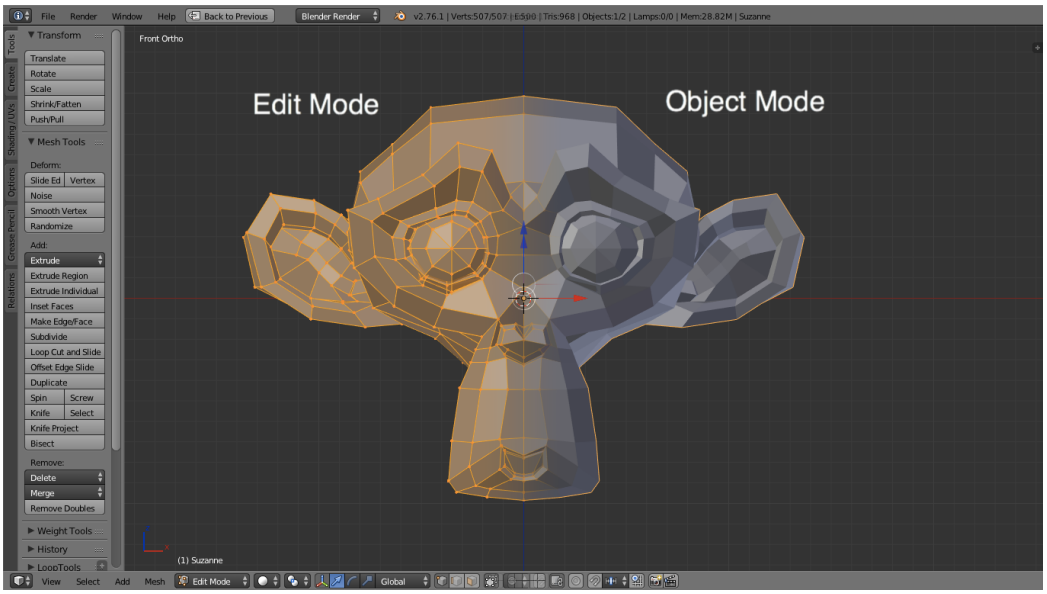
Nalazi se ili na gornjem ili na donjem dijelu područja. Svi uređivači sadrže zaglavlje kao kontejner za izbornik (eng. *„menu“*) i najčešće korištene alate. Izbornik i alati će se promijeniti ako se promijeni tip uređivača.



Slika 3: Zaglavlje (Header)

2.6 Objektni način (eng. *„Object Mode“*)

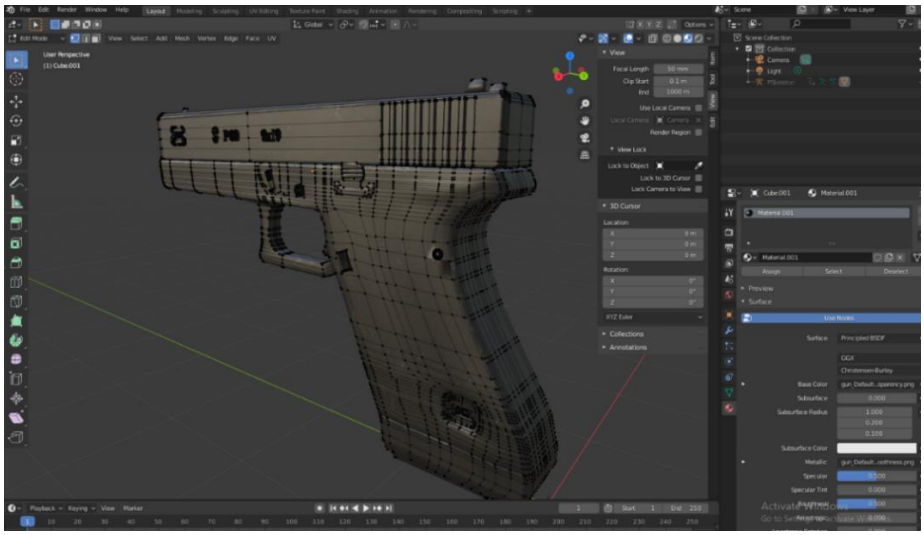
„Objektni“ način [7] utječe na cijeli objekt te je dostupan za sve tipove objekta i namijenjen je za uređivanje bloka podataka objekta (položaj, rotacija, veličina). Razlika Objektnog i Uređivačkog načina je prikazana na slici 4.



Slika 4: Uspoređivanje dvaju modova [7]

2.7 Uređivački način (eng. *„Edit mode“*)

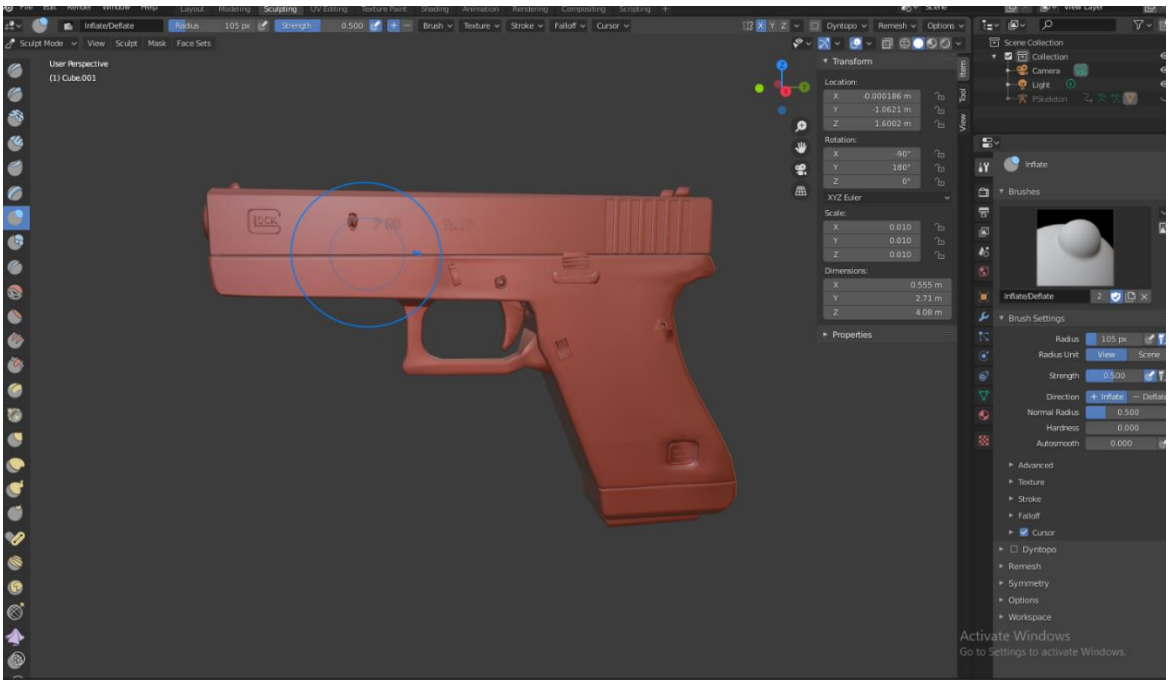
Operacije u „Uređivačkom“ načinu [8] utječu samo na geometriju objekta dok ne utječe na globalna svojstva kao što su veličina i rotacija. Može se uređivati samo mreža (eng. *„mesh“*) tog objekta kojeg smo odabrali. Za uređivanje ostalih objekata, moramo izaći i odabrati ih pa opet ući u Uređivački način. (slika 5)



Slika 5: Modeliranje objekta (Edit Mode) [snimka zaslona]

2.8 Oblikovni način (eng. *„Sculpt mode“*)

„Oblikovni“ način [9] je sličan „Uređivačkom“ načinu, koristi se za promjenu oblika nekog modela, ali „Oblikovni“ način koristi nešto drugačiji pristup. Umjesto da se bavi pojedinim elementima kao što su vertikale, rubovi i slično, ovdje koristimo tzv. kist kojim se manipulira geometrija i tako se dobiva željeni oblik kao na slici 6.



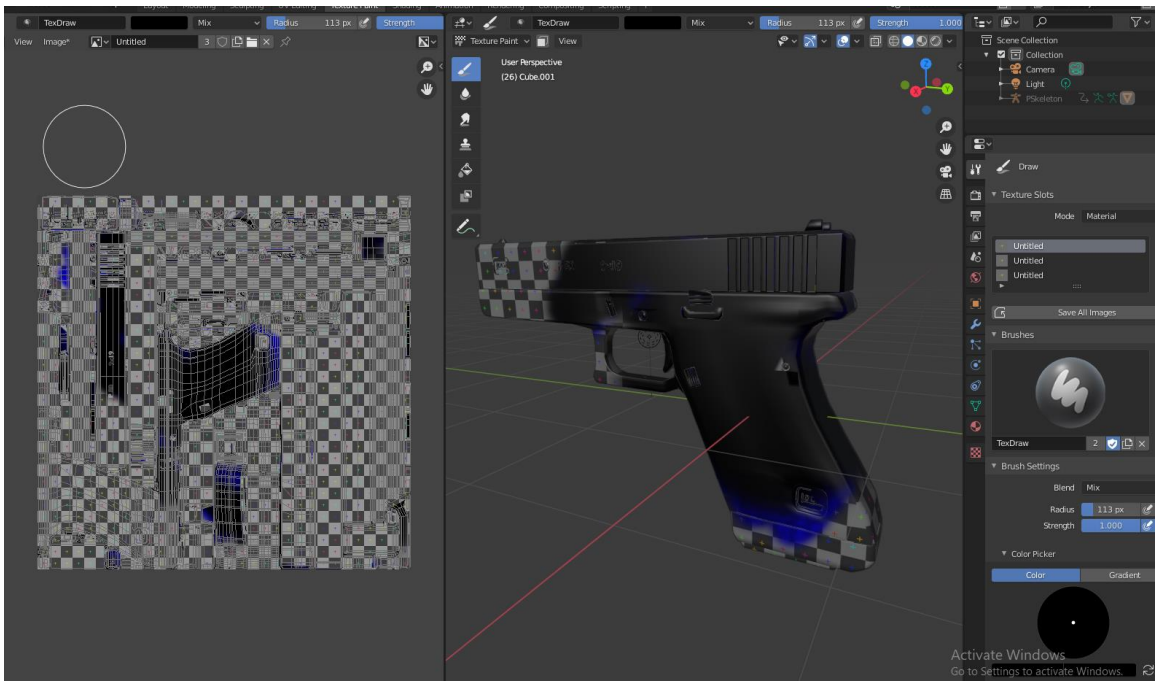
Slika 6: Skuptiranje (Sculpt Mode) [snimka zaslona]

2.9 Bojanje tekstura (eng. *„Texture paint“*)

Bojanje tekstura [10] koristi se za brzo i jednostavno uređivanje UV tekstura i slika. UV tekstura je slika (slika, slijed ili film) koja se koristi za bojanje površine mreže. (slika 7)

Postoje 3 načina za uređivanje slike koju koristi UV tekstura:

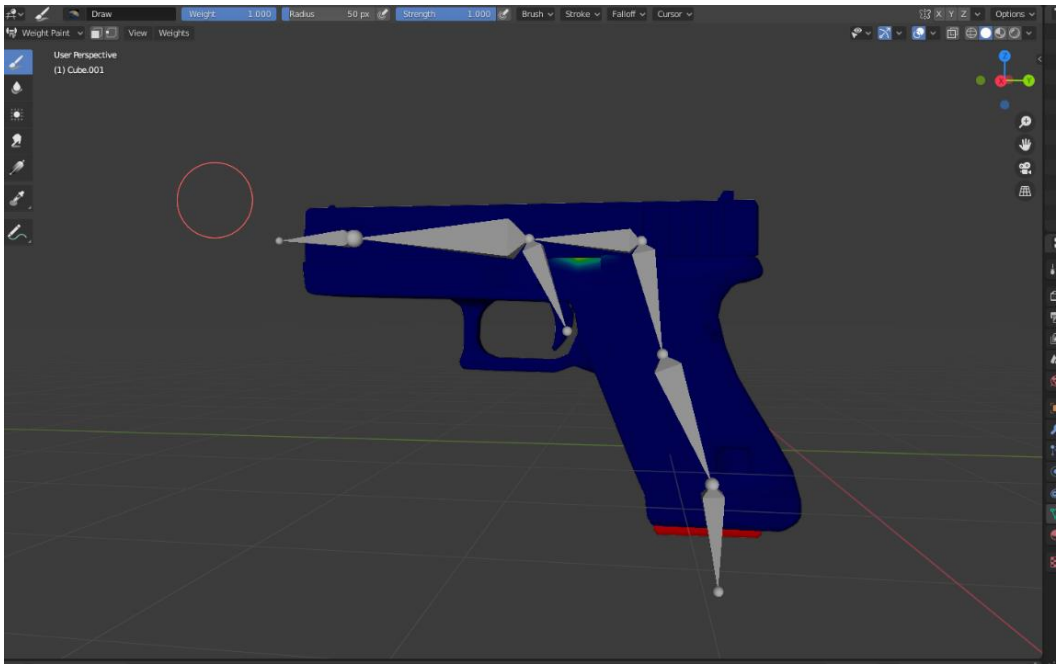
- Obojimo ravnu slike i preko UV karte prenosimo boju na lica mreže.
- Obojimo mrežicu u 3D prozoru i pustimo *Blender* da preko odabrane UV karte ažurira UV teksturu.
- Upotrijebi se bilo koji program za uređivanje slika gdje se odabere UV tekstura i učita slika. *Blender* zatim upotrijebi UV kartu i teksture za prijenos boja na lica mreže.



Slika 7: Bojanje tekstura (Texture paint) [snimka zaslona]

2.10 Određivanje utjecaja pri manipulaciji (eng. *„Weight paint“*)

Određivanje utjecaja pri manipulaciji [11] je važno za mnoge aspekte tijeka modeliranja u *Blenderu* kao što su namještanje mreže, nanošenje modifikatora (eng. *„modifier“*) i širenje čestica. Primarno se koristi za namještanje mreže gdje se grupe vrhova koriste za definiranje relativnih utjecaja kostiju (eng. *„bones“*) na mrežu. Na slici 8 prikazane su tzv. kosti koje treba dodijeliti različitim dijelovima objekta kako bi se mogli pomicati.



Slika 8: Određivanje utjecaja pri manipulaciji (Weight Paint) [snimka zaslona]

3. Modeliranje u Blender-u

3D modeliranje je proces kreiranje matematičke reprezentacije nekog trodimenzionalnog objekta i time se dobiva 3D model. 3D model je neki matematički prikaz oblika u 3D prostoru. Postoji nekoliko osnovnih 3D modela kao što su kocke, kugle, cilindri te mogu biti jednostavni ili složeni.

Postoji puno programa za 3D modeliranje kao što su *Blender*, *Maya*, *ZBrush* itd. Prvi modeli napravljeni su 1960-ih godina i nisu im svi imali pristup dok je danas dostupno svima te im je primjena sve veća. Primjenjuju se u filmskoj industriji, arhitekturi, industriji video igara, automobilskoj industriji i slično.

Razvojem tehničkih znanosti kroz povijest, javlja se potreba za boljim i preciznijim modelima. 3D modeliranje je neko vrijeme bilo dostupno samo profesionalcima, ali danas je to drugačije i dostupno je svima.

4. Postavljanje tekstura u Blenderu

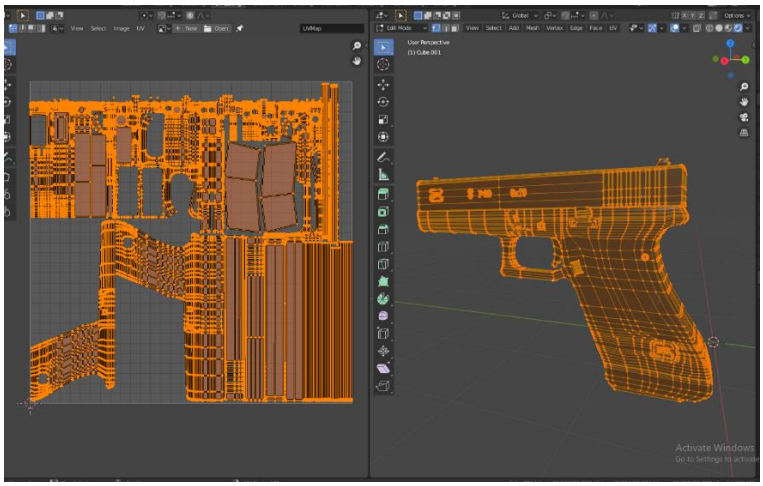
Teksture su neke vidljive površine koje se naprave tako što se objektu dodaju različita svojstva kao što su boja, prosvjetljenje, prozirnost (eng. *„emission“*), hrapavost i slično. Cilj je postići da objekt izgleda što realnije. Postoji više načina za dobivanje tekstura u Blenderu kao što i postoji više programa u kojima se može napraviti tekstura za objekt.

4.1 UV Raspakiravanje (eng. „*UV Unwrap*“)

Koristi se za raspakiravanje mreže [12] da bi se tekstura bolje rasporedila po objektu. Raspakiravanje uvijek dolazi onda kada je cijeli model gotov jer se svaka promjena na modelu mora ponovo raspakiravati. Svaka točka na UV mapi odgovara vrhu na mreži. Linije koje spajaju UV odgovaraju rubovima mreže. Koristi se kako bi se 2D tekstura stavila na 3D objekt kao što je prikazano na slici 9. Postupak raspakiravanja je jednostavan, ali postoji mnogo opcija i svaka znatno utječe na ishod raspakiravanja.

proces raspakiravanja:

1. Označe se točke ili stranice ako je kompliciraniji objekt.
2. Označi se sva mrežu objekta.
3. Odabere se način UV mapiranja s izbornika UV raspakiravanja.
4. Prilagode se postavke odmotavanja.
5. Doda se testna slika da se vidi hoće li doći do izobličenja.
6. Podesi se UV u uređivaču UV / slika.



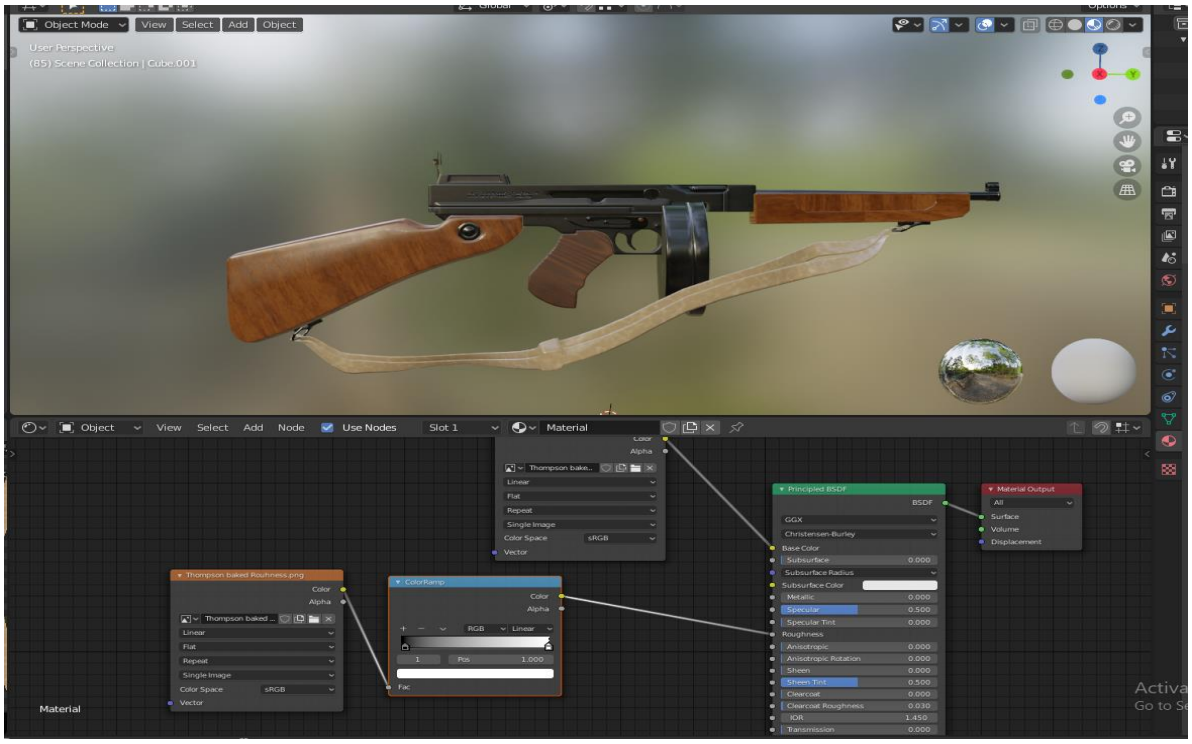
Slika 9: Proces raspakiravanja (Process of Unwrapping) [snimka zaslona]

4.2 Uređivač Čvorova (eng. *„Node Editor“)*

Koristi se za rad na temelju čvorova. Stablo čvorova može se promijeniti pomoću Uređivača Čvorova koji se nalazi u zaglavlju (eng. *„Node Editor-a“*). Uređivač Čvorova je glavni prozor *Uređivača Sjena*.

4.3 Uređivač Sjena (eng. *„Shader Editor“*)

Uređivač Sjena [13] koristi se za uređivanje materijala korišteni za *renderiranje* (eng. *(„rendering“*). Postoji puno čvorova koji se koriste za uređivanje materijala, ali ovdje je korišteno njih pet. *Diffuse Shader* koristio se za boju materijala, dok se za prozirnost koristio *Shader Glass* za staklo. Za hrapavi i metalni materijal kod pištolja ili metaka upotrijebljen je *Shader Roughness* i *Metalness*, a kod rasvjete *Shader Emission* da se dobije osvjetljenje. Također su korišteni *Shaderi* poput *ColorRamp* kako bi neki detalji bili vidljiviji kod hrapavosti te da se boja može učiniti svjetlijom i tamnijom. (slika 10)



Slika 10: Proces uređivanja sjena (Process of Shader Editor) [snimka zaslona]

5. Postavke Rendera

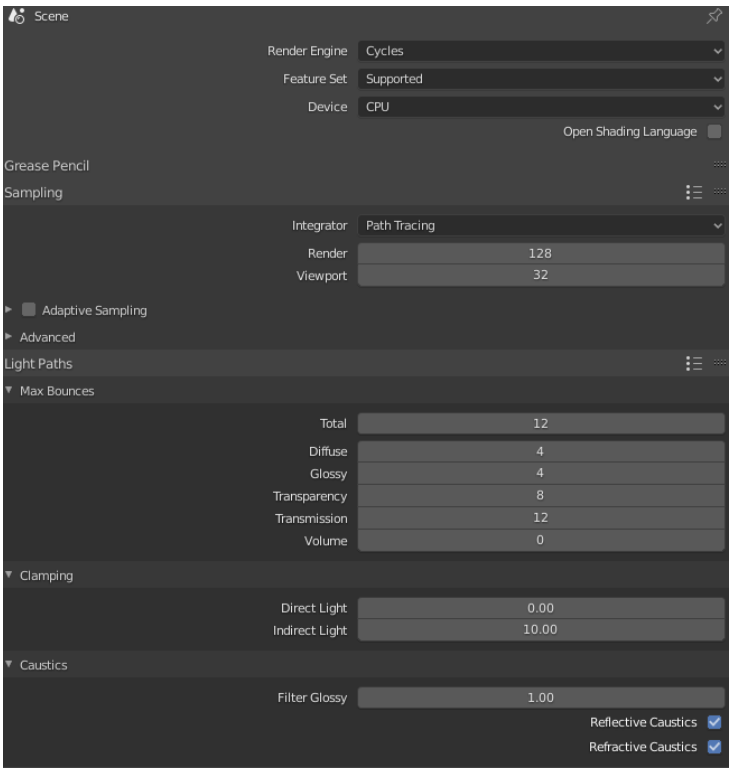
Renderiranje [14] je proces stvaranja 2D slike ili videozapisa neke 3D scene.
Ta slika ovisi o četiri čimbenika na koja korisnik može utjecati:

- Kamera
- Osvjetljenje u sceni
- Materijal svakog predmeta
- Razne postavke *rendera* (kvaliteta slike)

Blender sadrži 3 *rendera* različitih jačina:

- **Eevee** je fizički zasnovan stvarni *render*.
- **Cycles** je fizički utemeljen trag puta.
- **Workbench** je dizajniran za izgled modeliranje i preglede.

Za renderiranje svih scena korišten je *Blenderov Cycles render* jer je precizniji i izgleda realističnije. Postavke *rendera* vidljive su na slici 11.



Slika 11: Postavke Renderanja (Rendering settings) [14]

6. Animacija (eng. *„Animation“*)

Animacija [15] je tjeranje objekta da se pomiče ili mijenja oblik s vremenom. Postoji nekoliko načina na koji se objekti mogu animirati:

- **Kretanje kao cijeli objekt**
  -Mijenjajući svoju poziciju, orijentaciju ili veličinu u vremenu.

- **Deformirajući ih**
  -Animiranje njihovih vrhova ili kontrolnih točaka.

- **Naslijeđena animacija**
  -Uzrokovanje kretanje predmeta na temelju drugog predmeta. Animacija se obično postiže korištenjem ključnih kadrova (eng. *„keyframe“*).

6.1 Namještanje (eng. *„Rigging“)*

„*Rigging*“ je postupak kojim se dodaju kontrole objektima, obično u svrhu animacije. Namještanje često uključuje upotrebu jedne ili više značajki:

- **Armature (eng. *„Armature“*)**
  -Omogućuje mrežnim objektima fleksibilne spojeve i često se koristi za *skeletnu* animaciju.

- **Ograničenja (eng. *„Constraints“*):**
  -Za ograničenje pokreta (granice do kojih se neki dio objekta može kretati)

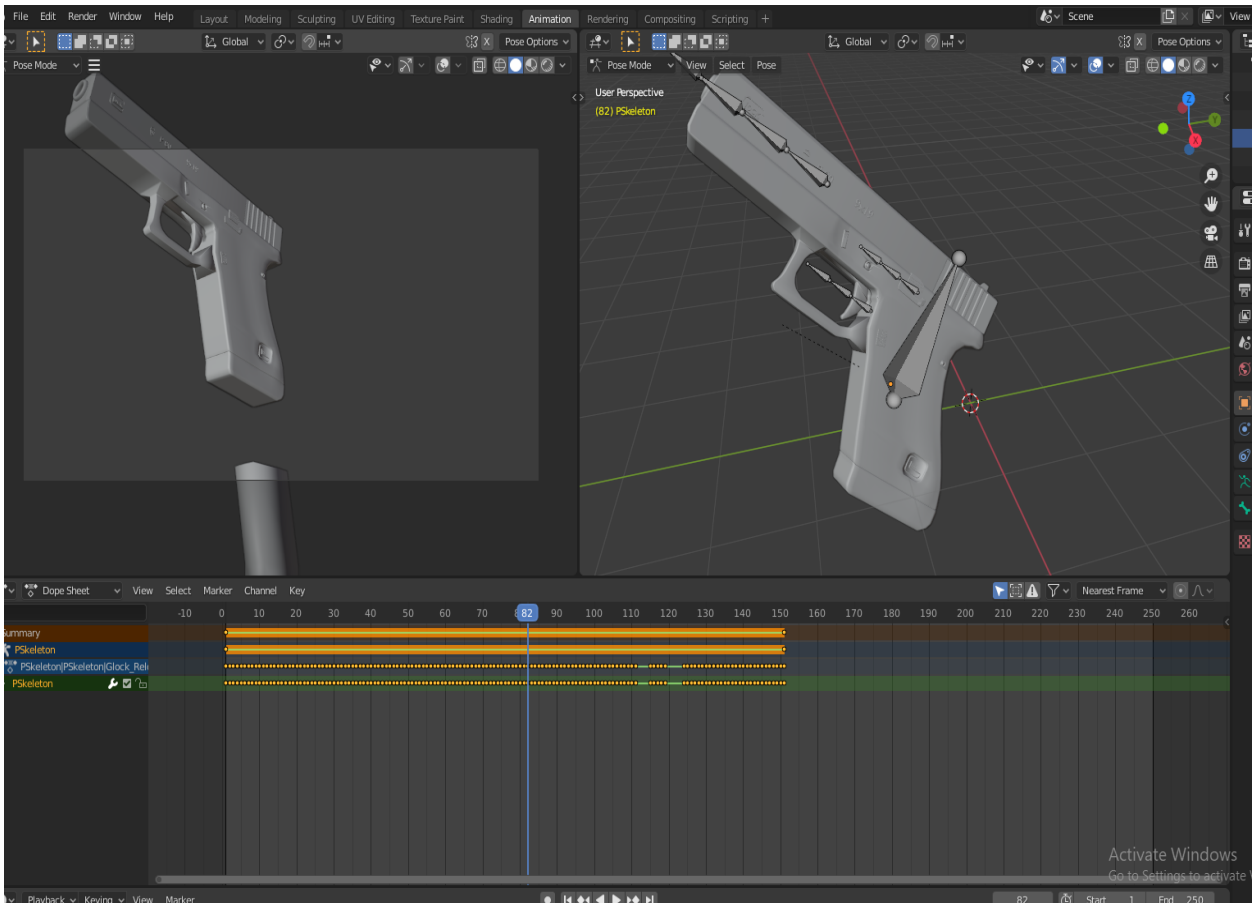
- **Modifikatori objekta (eng. *„Object Modifiers“)*:**
  -Deformacija mreže može biti prilično uključena, postoji više *modifikatora* koji pomažu u kontroli.

- **Ključevi za oblik (eng. *„Shake Keys“*):**
  -Podržava drukčije ciljne oblike (izraz lica) koje treba kontrolirati.

- **Vozači (eng. *„Drivers“*)**

  -Ovako oprema može istovremeno kontrolirati mnogo različitih vrijednosti kao i automatsko ažuriranje nekih svojstava na temelju promjena na drugim mjestima.

U ovom projektu za animaciju korištene su armature da bi se dodale tipke kojima se kontrolira pojedini dio te ograničenja za svaki dio da ima određenu granicu kretanja. Slika 12 prikazuje trajanje animacije i označene ključne kadrove.



Slika 12: Proces Animiranja (Animation Process) [snimka zaslona]

7. Izrada Modela

Kao 1. dio cijelog projekta treba napraviti prostor u kojem će se sve odvijati te modele koji će biti korišteni tijekom igre. Sve korištene modele treba prebaciti u .fbx datoteku jer je to najbolji način za ubacivanje u *Unity*. Svaki model opisat će se u nekoliko koraka te se istaknuti dijelovi koji su zahtijevali veći dio rada.

7.1 Proces izrade modela Glocka

Počinje tako da se nađe par slika *Glocka* koje služe kao referenca tijekom izrade modela. Tako će se lakše izraditi kostur modela kao i detalje koji dolaze nakon kostura.

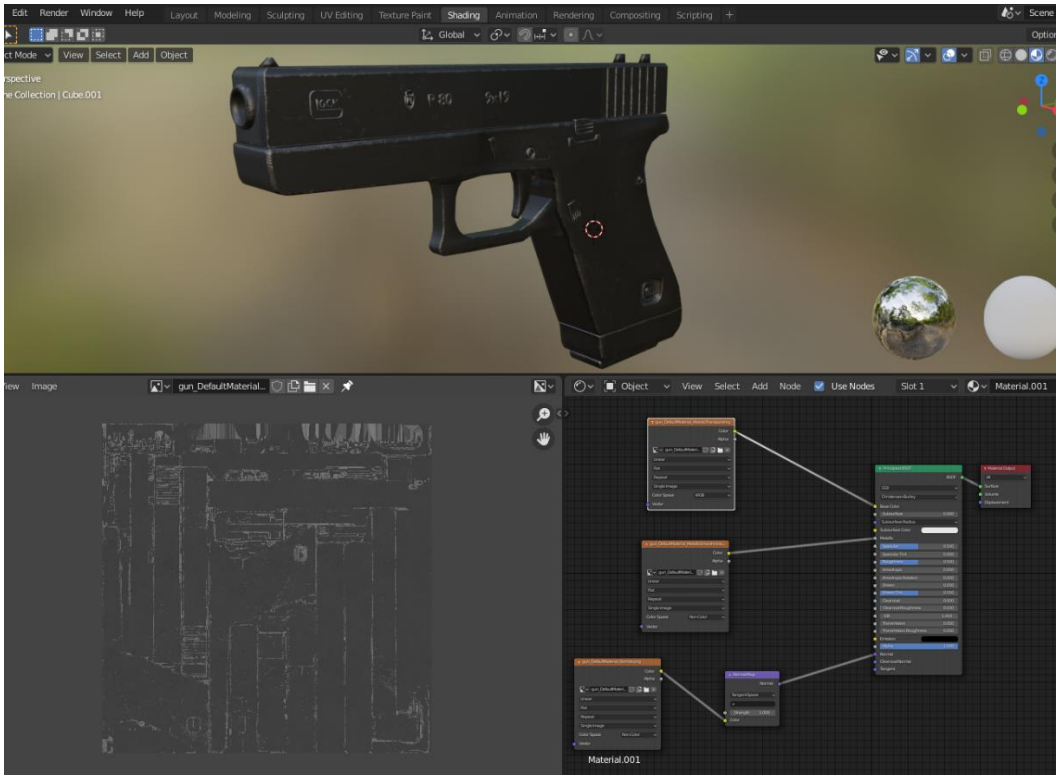
U Objektivnom načinu, dodaje se kocka koja se dijeli na dvije polovice. Jedna polovica se obriše i dodaje se *modifikator* zrcaljena (eng. „*mirror*“) po y-osi. Sada više ne treba uređivati cijelu kocku već samo jednu polovicu, a na drugoj će sve biti zrcaljeno i time je olakšan proces izrade. Prelazi se u Uređivački način i produžuje dio mreže za gornji dio pištolja dok se donji dio izvlači da se dobije rukohvat pištolja. Na sredini modela, pomiču se kutovi koji se poravnavaju, izvlače i okreću dok se ne dobije okidač pištolja.

Kada je kostur gotov, dodaje se cilindar koji se produžuje po y-osi i *skalira* da pristaje kosturu modela. Korišten je *modifikator* zaobljenja (eng. „*bevel*“) i obrisan je prednji dio cilindra da se dobije šupljina. Za postizanje prirodnog izgleda cijevi pištolja, upotrijebljena je opcija *smooth* i *subdivision*. Preko opcije za ubacivanje (eng. „*insert*“) i izvlačenja (eng. „*extrude*“) izrađeno je spremište za streljivo koje je oblikovano pomicanjem točaka za prirodniji izgled.

Nakon izrade kostura modela, trebalo je cijeli model podijeliti na nekoliko dijelova da bi se lakše napravili detalji na modelu. Prvo su napravljeni vijci i udubljenja po modelu koristeći *modifikator Boolean*. Nakon toga po z-osi izvučena su lica i skalirana prema sredini da se napravi ciljnik na pištolju. Završna stavlja uređivanja bila je dodati tekst koji je zatim bilo potrebno pretvoriti u drugačiji oblik da se može uređivati.

Slika 13: Glock Model [snimka zaslona]

Nakon izrađenog modela, trebalo je označiti stranice koje će se rezati da se model može raspakirati i napraviti UV mapu. Nakon što su model i proces raspakiravanja gotovi, modelu se dodaje materijal koji se uređuje u *Uređivaču sjena*. (slika 14)



Slika 14: Proces izrade i uređivanja UV mape za Glock [snimka zaslona]

Završni dio je prebacivanje modela u *.fbx* datoteku za izvoz i lakše ubacivanje u *Unity*.

7.2 Proces izrade modela AK-47

Prilikom izrade modela za AK-47 potrebno je prvo pronaći slike iz različitih kutova koje će pomoći kod izrade modela. Dodavanjem pravokutnika za izradu kostura puške i dijeli se na dva dijela. Idući korak u procesu je brisanje jednog dijela kako bi se mogao dodati modifikator zrcaljenja koji omogućava uređivanje samo jedne strane modela, dok će se sve promjene oslikavati i na "zrcaljenoj" strani.

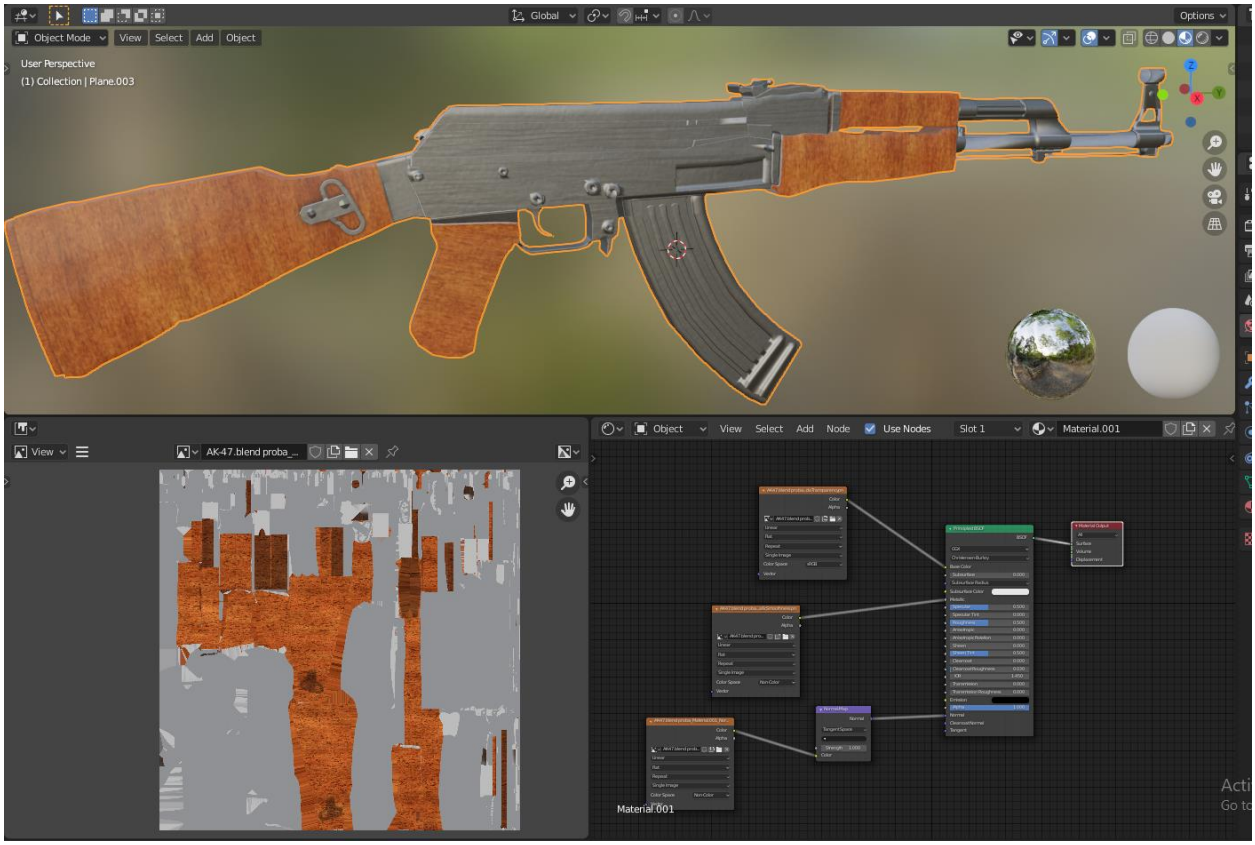
Pravokutnik se produžuje po y-osi i smanjuje na jednoj strani da bi se izvukla mreža i napravio oslonac puške. Oslonac je potrebno zaobliti da ne izgleda previše „oštro" i time se dobije prirodniji oblik. Nakon oslonca, dodaju se vertikale na donjem dijelu puške koje će služiti za izvlačenje držača i spremišta za streljivo po z-osi. Spremište za streljivo je bilo potrebno savinuti i označiti lica za postizanje udubljenja.

Na prednjoj strani puške, dodana su dva cilindra međusobno spojena i *skalirana* da pristaju ostatku modela. Na oba cilindra pobrisan je dio mreže da se napravi šupljina te zaobljena preko *modifikatora* za zaobljenje.

Nakon izrade kostura, treba ući u Uređivački način i napraviti detalje preko izvlačenja i pomicanja kutova. Ciljnik je izrađen tako što se iz prednjeg dijela cijevi, dio mreže izvukao po z-osi te pomoću *skaliranja* i rotiranja postigao prirodni oblik kako *AK-47* ima. Koristeći *modifikator boolean* i opciju za glatkoću (eng. „*smooth*") napravljeni su vijci.

Slika 15: Model AK-47 [snimka zaslona]

Ovdje je UV mapa napravljena pomoću Pametnog Raspakiravanja (eng. „*Smart Unwrapping*“). Dodaje se materijal i uređuje u *Uređivaču sjena* preko čvorova. Na slici 16 prikazana je UV mapa i čvorovi preko kojih se uređuje materijal.



Slika 16: Proces izrade i uređivanja UV mape za AK-47 [snimka zaslona]

Na kraju model se sprema i prebacuje u *.fbx* datoteku da bude spreman za ubacivanje u *Unity*.

7.3 Proces izrade modela AWM-a

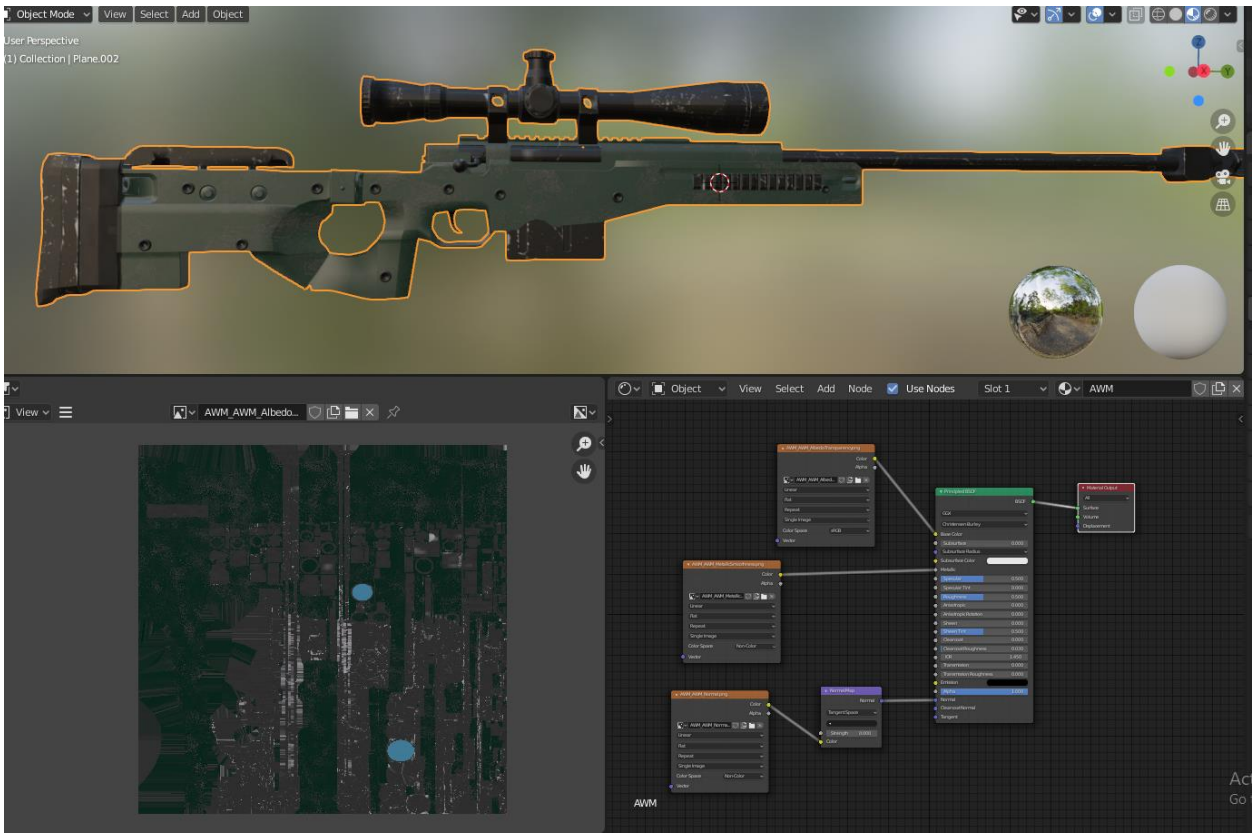
Kao i kod prethodnih modela, potrebno je koristiti par slika iz različitih kutova za referencu. Dodaje se pravokutnik i dijeli na pola tako da se jedna strane može obrisati. Koristi se *modifikator* zrcaljenja i proširuje pravokutnik po y i x-osi. Obrišu se lica mreže na sredini modela da bi se kutovi mogli izvući po z-osi prema dole i zarotirati prema unutra. Sada se može izvući i savinuti prekidač koristeći proporcionalno uređivanje (eng. „*Proportional Editing*“). Nakon što je kostur gotov, neke dijelove je trebalo zaobliti i dodati im glatkoću.

Cijev snajpera je napravljena preko cilindra koji je skaliran po y-osi i postavljen na ostatak modela. Izrađena je šupljina te su krajevi zaobljeni prema unutra. Kraj cijevi je podebljan pomoću *modifikatora Solidify* te su kutovi poravnati da se mogu zarotirati.

Ciljnik je trebalo odvojeno napraviti i poslije ga spojiti s ostatkom kostura snajpera. Dodan je objekt cilindra, označen na nekoliko mjesta te na svakom od tih mjesta različito povećana ili smanjena mreža pomoću skaliranja u svrhu postizanje prirodne veličine i izgleda kakav ciljnik ima. Na modelu su napravljena zadnja uređivanja i dodani su vijci pomoću *Booleana.*

Slika 17: Model AWM-a [snimka zaslona]

Za njegovu UV-mapu nisu označene stranice već je korištena opcija za Pametno Raspakiravanje (eng. „*Smart Unwrapping*“) jer je bilo nekoliko problema s topologijom pa tekstura nije ispadala dobro raspoređena po modelu. Materijal modela je uređen u *Uređivaču sjena* i prebačen u *.fbx* datoteku spremnu za *Unity*. (Slika 18)



Slika 18: Proces izrade i uređivanja UV mape za AWM [snimka zaslona]

7.4 Proces izrade modela Thompsona

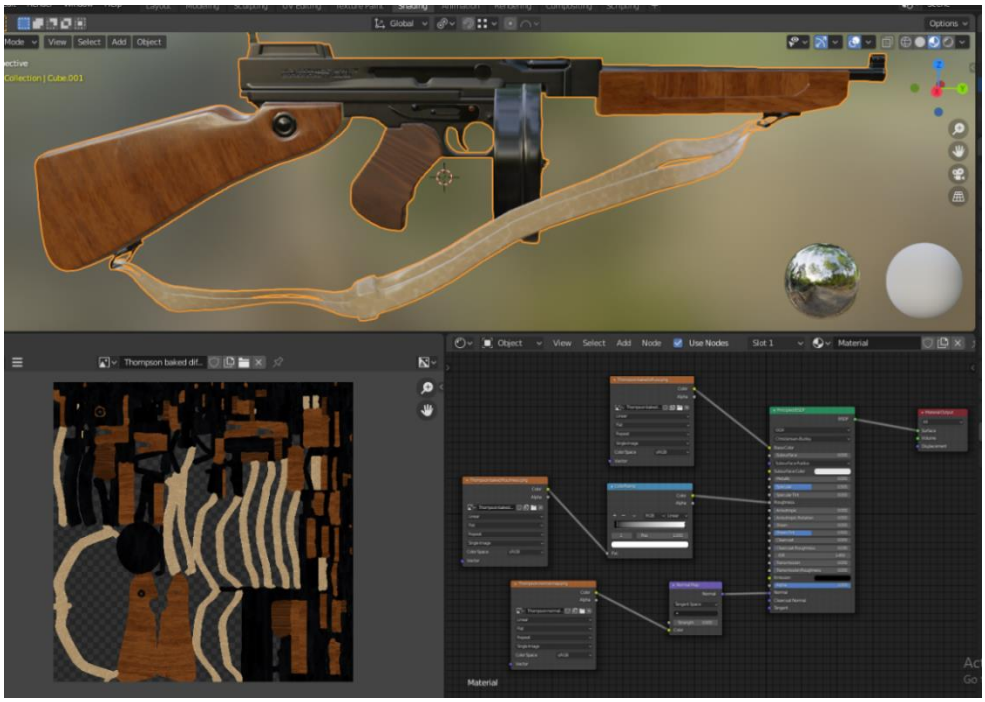
Prvi korak identičan je kao i za prethodne modele. Model je podijeljen i označena su lica koja će se izvlačiti za ostale dijelove oružja. Izvučena je mreža za rukohvat te pomaknute stranice i rotirane pod kutem od 30' za postizanje određenog oblika.

Sljedeće je preko cilindra napravljeno spremište za streljivo skraćivanjem po x-osi jer *Thompson* ima nešto drugačiji spremnik od prethodnih oružja. Cijev je napravljena pomoću cilindra koji je produžen po x-osi i zatim su pomoću naredbe *Scale* skupljeni krajevi cilindra. Ciljnik je napravljen putem označavanja i izvlačenja kutova iz cijevi oružja te je na kraju zaobljen da izgleda prirodnije.

Zadnja stavka bio je napraviti remen, a to je napravljeno preko pravokutnika. Pravokutnik je trebalo produžiti po y-osi te ga skratiti po x-osi. Da bi se postigao savitljivi izgled remena, pravokutnik je podijeljen na nekoliko dijelova i zatim svaki dio zarotiran za nekoliko stupnjeva.

Slika 19: Model Thompsona [snimka zaslona]

Na svakom objektu označeno je nekoliko stranica da se pokaže *Blenderu* kako da raspakira objekt i da se dobije UV-mapa pomoću koje će biti stavljena tekstura. Materijal je dodatno uređivan u *Uređivačkom načinu* te završni model prebačen u *.fbx* datoteku spremnu za *Unity*. (slika 20)



Slika 20: Proces izrade i uređivanja UV mape za Thompson [snimka zaslona]

7.5 Proces izrade modela M60

Početak je identičan kao i za prethodne modele. Dodane su vertikale da bi se kutovi mogli izvući i napraviti rukohvat. Sljedeći korak bio je napraviti zadnji dio oružja koji ide na rame pomoću *Extrude* i *Bevel modifikatora*.

Cijevi su napravljene preko cilindara i ciljnik je dodan naknadno da se smanji ili uveća ovisno o modelu. Napravljen je dvonožac za koji je korišteno *modifikator* zrcaljenja. Nakon toga je napravljena jedna rupa i kopirana nekoliko puta pomoću *modifikatora Boolean* i Liste (eng. „*Array*“).

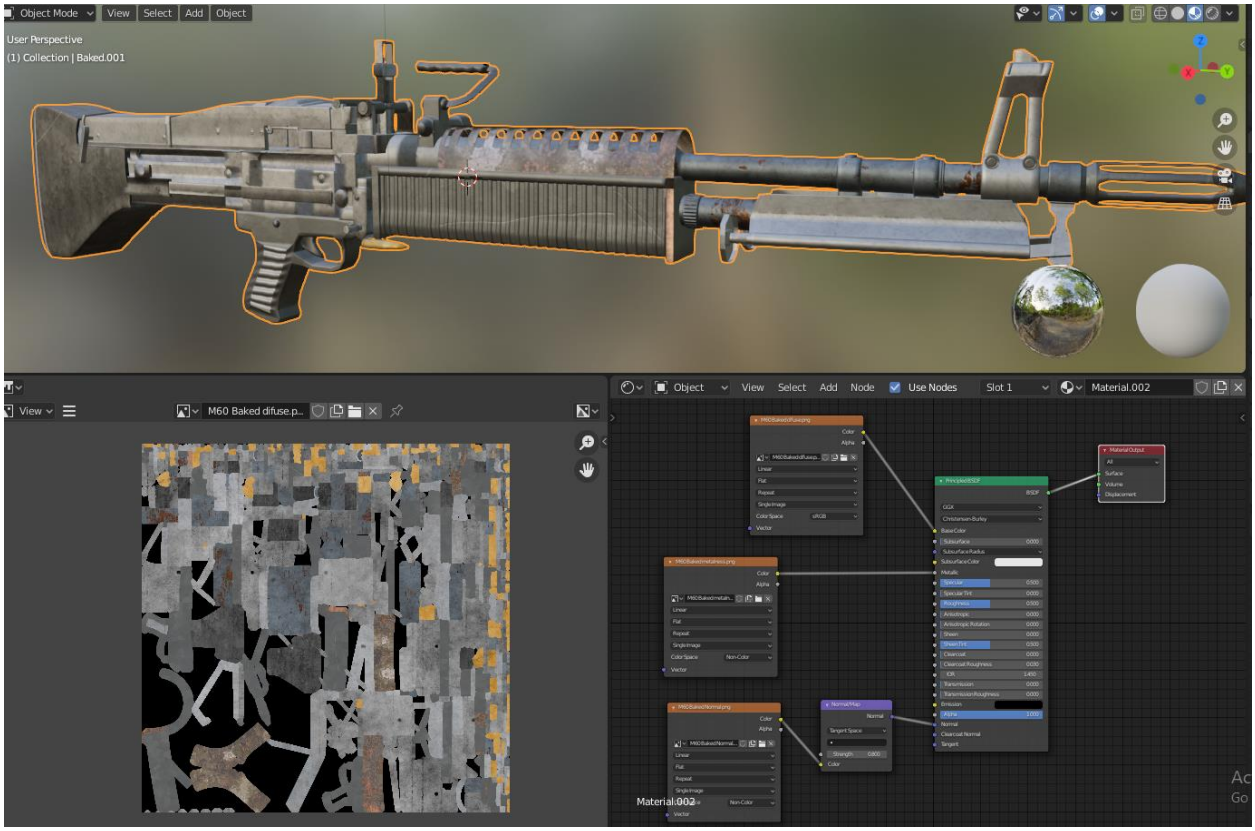
Zadnja stavka i možda najkompliciranija do sada bila je napraviti spremnik odnosno metke koji izlaze i zamjenjuju se iz oružja jer model M60 ima malo drugačiji način punjenja oružja. Prvo je napravljen jedan metak pomoću cilindra, a zatim pomoću *modifikatora* liste (eng. *„Array“*) napravljeno je nekoliko kopija tog istoga metka svaki na istoj udaljenosti. Taj *modifikator* je jako koristan jer koja god promjena se napravi na tom jednome metku, sve se mijenja i na ostalim metcima.

Na kraju su dodana još neka udubljenja, vijci, izbočine te su napravljena oštećenja da se postigne oblik starog oružja. Stalak za oružje nije rađen jer nije bilo potrebe s obzirom na to da ga se neće koristiti u simulacijskoj igri.

Slika 21: Model M60 [snimka zaslona]

Napravljena je UV-mapa kao i kod ostalih modela pomoću označenih nekoliko stranica da bi se model mogao raspakirati. Kada je dodan materijal, trebalo ga je u *Uređivačkom*

načinu dodatno urediti za postizanje izgleda kakvo ima istrošeno i oštećeno oružje. Preko čvorova je povećana grubost i jačina teksture. (slika 22)



Slika 22: Proces izrade i uređivanja UV mape za M60 [snimka zaslona]

7.6 Proces izrade modela Streljane

Iako je ovo najveći model od svih u ovom projektu što se tiče dimenzija, sam model nije bio tako težak. Svi objekti bili su jednostavni i bez previše detalja.

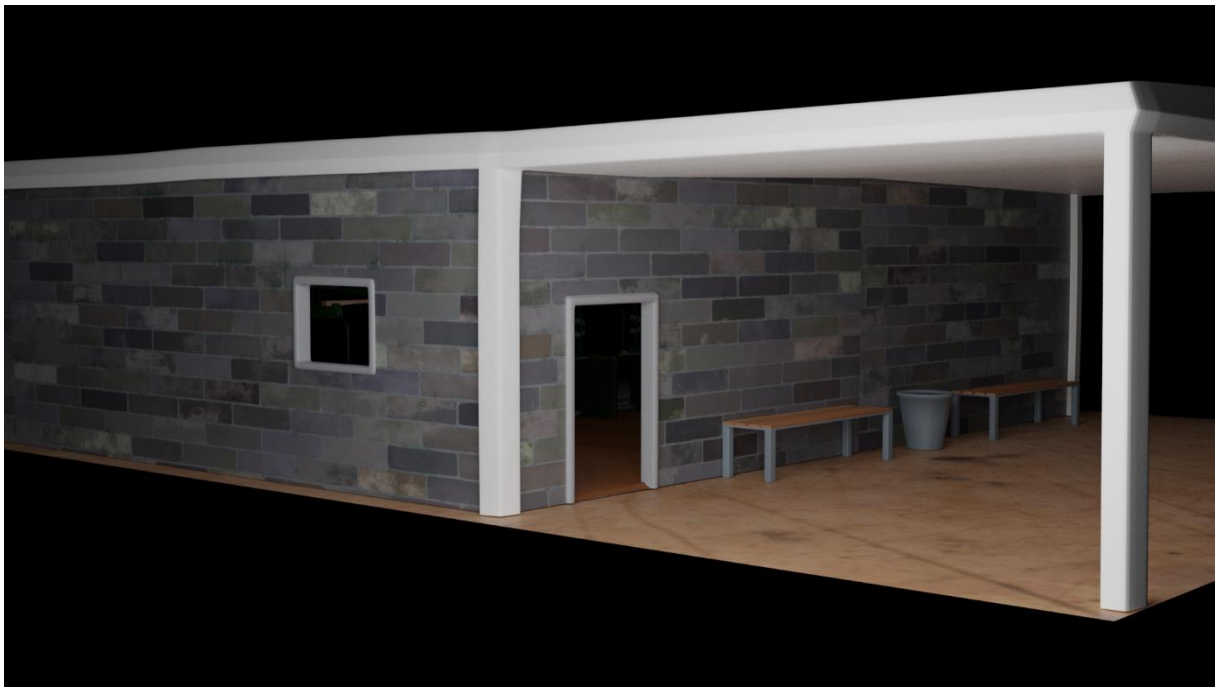
Rad započinje tako što se stvara objekt *plane* koji je korišten kad „pod“ te na njega je stavljen veliki objekt kocke da se napravi okvir streljane. Korišten je *modifikator* da se poveća debljina okvira da se mogu napraviti prozori i vrata tako što bi se označili kutovi i pobrisala sredina.

Zatim su napravljene neke lakše stvari poput klupa, stolova, koš za smeće, slušalica, gumbova i metaka jer se samo trebala pomicati mreža za postizanje željenog oblika.
Na svim objektima dodan je *modifikator* liste za postizanje veće količine te na kraju im je dodana opciju glatkoće i *modifikator Subdivision* da oblik izgleda realnije.

Kada su riješeni lakši dijelovi, počelo je modeliranje mjesta za ciljanje i pucanje. Trebalo je napraviti redne brojeve za koje je korišten tekst i pretvoren u mrežu tako da se mogu uređivati. Staklo je napravljeno povezujući kutove te mu je smanjena grubost (eng. *„roughness“*) kako bi se moglo vidjeti kroz njega.

Kao zadnji dio ostavljena su lampe i pomične mete (eng. *„target“*) jer su imali malo više detalja za napraviti. Sam oblik nije težak, ali je bilo potrebno dodati svjetlo i sjenu.

Kod većine ovih objekata kao što su mete, stalak ,klupe ,stolovi, svjetla itd. korišten je *modifikator* liste da se ne radi svaki objekt posebno već se samo kopira prvi napravljeni model i svaka promjena koju napravi.



Slika 23: Model Streljane izvana [snimka zaslona]

Slika 24: Model Streljane unutra [snimka zaslona]

Na kraju je trebalo precizno označiti stranice svakog objekta da ga se uredno raspakira kako bi svi dijelovi stali na UV mapu. S obzirom na to da se koristi svjetlo i prozirnost kod modela, trebaju se dodati čvorovi kojima se uređuje prozirnost i svjetlo u Uređivačkom načinu.

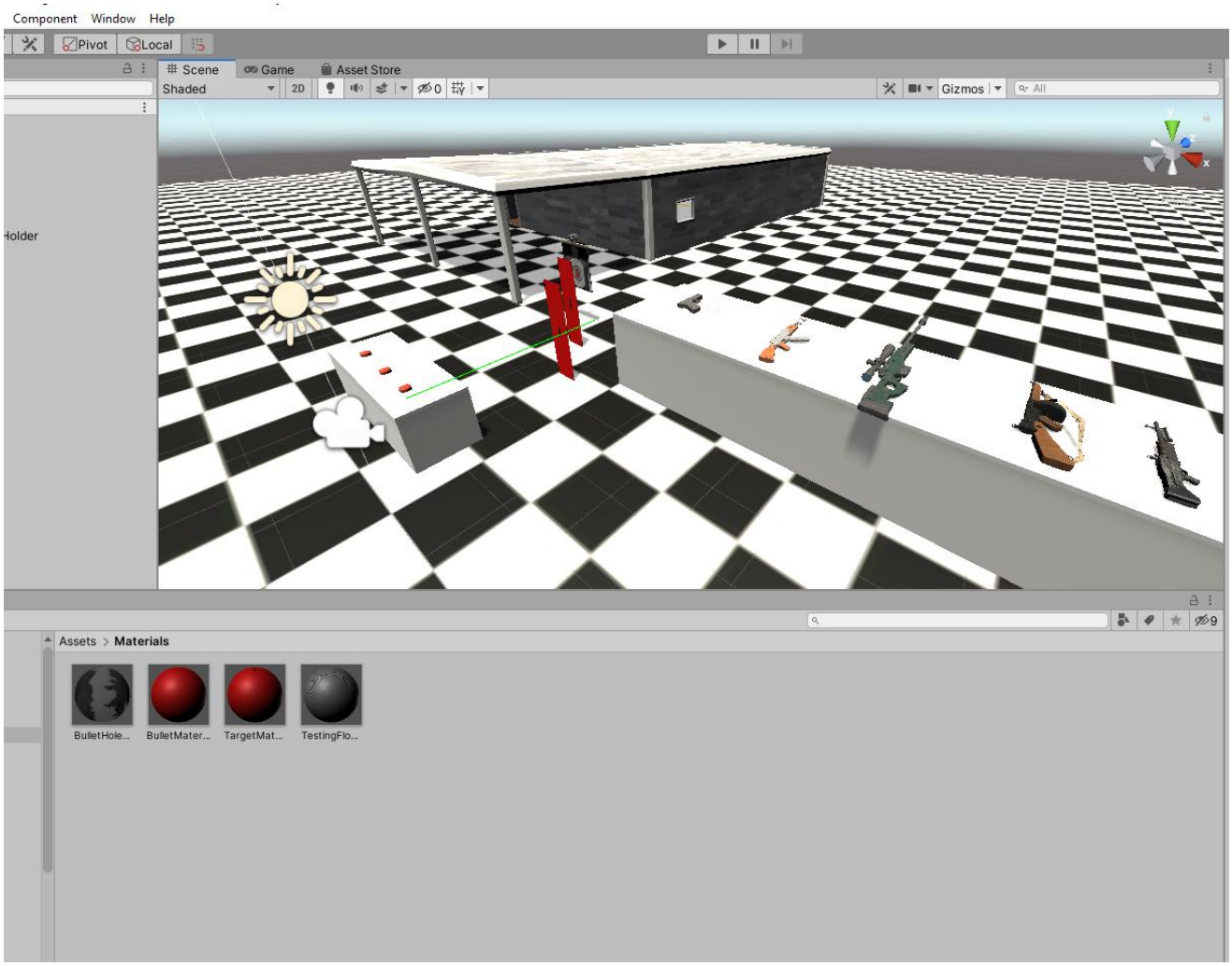
Svakom objektu treba napraviti UV-mapu, teksturu i urediti materijale. Na kraju se modeli prebacuju u *.fbx* datoteku kako bi bili spremni za ubacivanje u *Unity*.

8. Unity

Unity [16] je alat koji se koristi od strane mnogih razvojnih programera kako bi kreirali i naposljetku osposobili svoje kreacije. On je *višeplatformski* softver za razvoj igara, a razvija je kompanija *Unity Technologies*. *Unity* softver je bogat mogućnostima, a ujedno i jako jednostavan za korištenje. Isto tako *Unity* je besplatan za osobnu upotrebu (do određenog profita stečenog njime).

Može se koristiti za razvoj 2D i 3D igara, kao i simulacija za sve podržane platforme, arhitektonske prezentacije i slično. Softver nudi kodiranje primarno u C# [17], a za *Unity* uređivač u obliku dodataka. Unity nije samo softver za izgradnju igrice, riječ je o profesionalnom alatu koji koriste neka od najvećih imena u čitavoj industriji. Kao primjer može se uzeti igra Angry Birds koja je prije par godina postala popularna uahvaljujući Unityu.

Unity uređivač podržan je na Windowsu i MacOS-u, s verzijom uređivača dostupnim za Linux platformu. Softver podržava kreiranje igara na 27 platforma, a neke od njih su iOS, Android, Tizen, Windows, Universal Windows Platform, MacOS, Linux, WebGL, PlayStation 4, PlayStation Vita, Xbox one, Wii U, 3DS, Oculus Rift, Steam VR, PlayStation VR, Windows Mixed Reality, Daydream, Android TV, Samsung Smart TV, TVOS, Nintendo Switch, Fire OS, Facebook, Gameroom, Google ARCore, Vuforia.



Slika 25: Svi modeli korišteni u projektu [snimka zaslona]

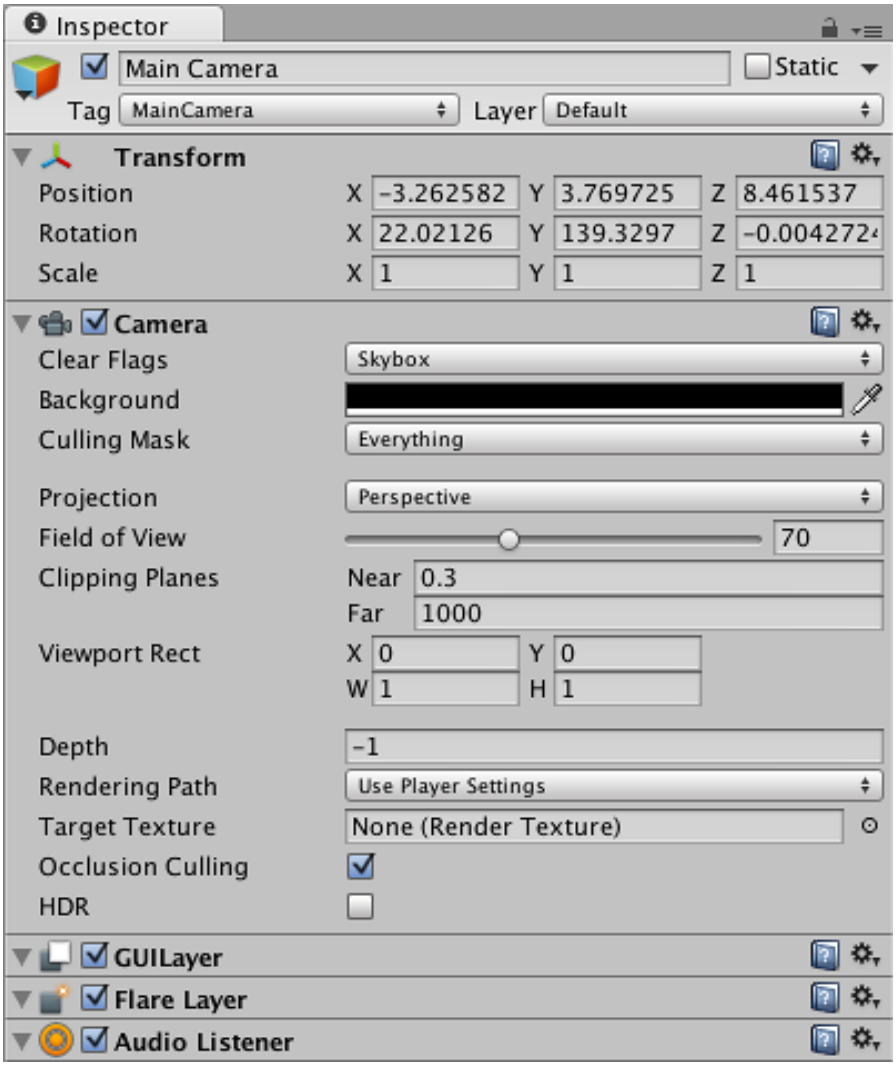
8.1 Povijest

Game engine Unity lansiran je 2005. godine, a jedan od ciljeva bio je demokratizirati razvoj igrica i učiniti programiranje istih pristupačnim većem broju developera. Prvenstveno je bio osmišljen za *Mac OS X*, ali kasnije mu je nadodana podrška i za *Microsoft Windows*.

Kroz povijest [18] lansirano je nekoliko *Unity* verzija: *Unity* 2.0 2007. godine, *Unity* 3.0 2010.godine, *Unity* 4.0 2012. godine (ova verzija je bila važna jer je uključivala *DirectX* i *Adobe Flash* podršku), *Unity* 5.0 u 2015. godine, *Unity* 2017-2020. što je zadnja verzija *Unity-a*. Nakon verzije 5, *Unity* je prebacio sustav imenovanja verzija gdje prvi broj vise ne predstavlja veliku promjenu verzije *engine*-a već godinu proizvodnje. 2009 godine, *Unity personal* postaje besplatan.

8.2 Prozor Inspektora (eng. *„Inspector Window“*)

Igre u *Unity-u* sastoje se od više *GameObjekata* (eng. *„GameObject“*) koji sadrže mrežice, skripte, zvukove ili druge grafičke elemente poput svjetla. Prozor Inspektora [19] prikazuje detaljne informacije o trenutno odabranom *GameObjektu*, uključujući sve priključene komponente i njihova svojstva. Ovdje se mijenja rad *GameObjekta* na sceni. Bilo koje svojstvo prikazano u inspektoru može se izmijeniti. Isto tako može se imati više inspektora gdje svaki prikazuje određenu stavku. Slika 26 prikazuje prozor inspektora

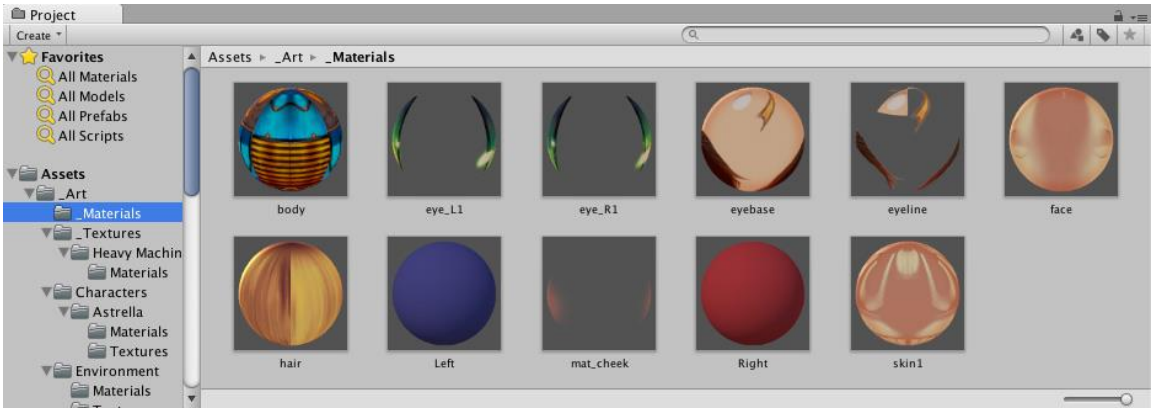


Slika 26: Prozor Inspektora (Inspector window) [19]

8.3 Prozor Projekta (eng. *„Project Window“*)

Prozor Projekta [20] je mjesto gdje se dodaju materijali, teksture, skripte i ostale stvari korištene u projektu prikazano je na slici 27. Prikazuje sve datoteke povezane s projektom i glavni je način za pretraživanje i pomicanje datoteka, skripta, objekata i slično u glavnom direktoriju (eng. *„Assets“*). Kada se pokrene novi projekt, ovaj prozor je otvoren.

Prozor projekta može se pomicati i uređivati po volji. Lijeva strana pokazuje strukturu mapa projekta kao hijerarhijski popis.



Slika 27: Prozor projekta (Project window) [20]

8.4 Scenski Pogled (eng. *„Scene View“*)

Interaktivni pogled u *scenu*. Koristimo *Scenski* Pogled [21] za selektiranje i pozicioniranje scenarija, likova, kamera, svijetla, i svih ostalih objekta. U *Scenskom* Pogledu [21] se uređuje *scena*, dodaju elementi *scene* koji mogu biti vizualni, audio, osvjetljenja i sl.

Scenski Pogled ima set navigacijskih kontrola koje se koriste za brzo i jednostavno kretanje.

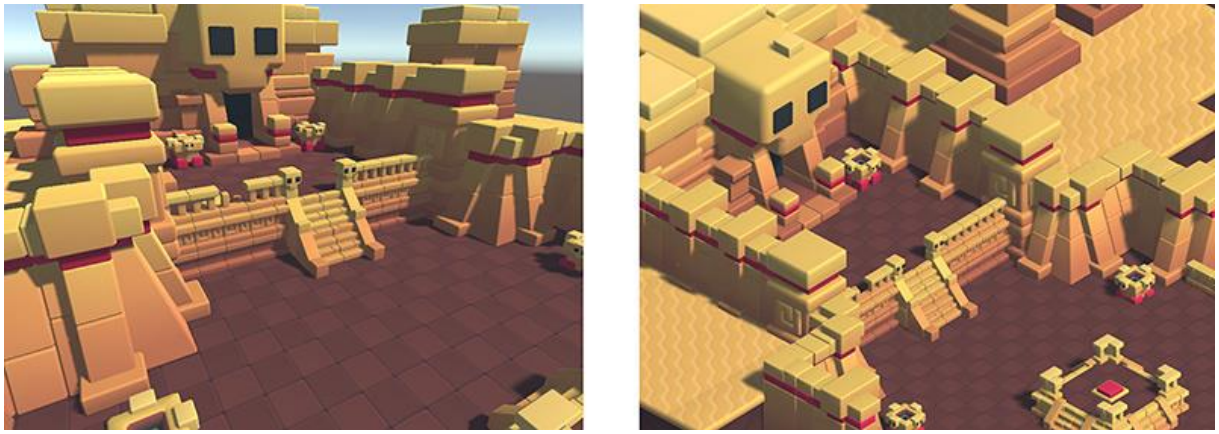
- *Scenski* kompas (eng. *„Scene Gizmo“*)
- Alati za pomicanje, orijentiranje i zumiranje.
- Alat za centar

Slika 28: Scene Gizmo

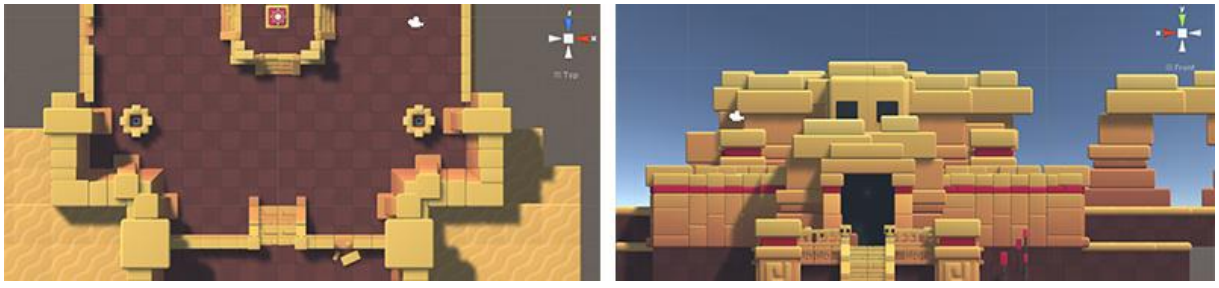
Scenski Kompas je u desnom gornjem kutu prozora *Scenskog* Pogleda. Koristi se za promjenu kuta gledanja i projekcije. Sadrži tri glavne osi orijentacije u prostoru X, Y, Z.

Isto tako koriste se neki osnovni prečaci za brže kretanje i uređivanje scene.

W – kretanje (eng. *„move“*)
E – rotiranje (eng. *„rotate“*)
R – skaliranje (eng. *„scale“*)
T – pravokutni alat (eng. *„rect tool“*)
Postoji više načina projekcija koje možemo koristiti. Jedan je ortogonalni dok je drugi perspektivni.



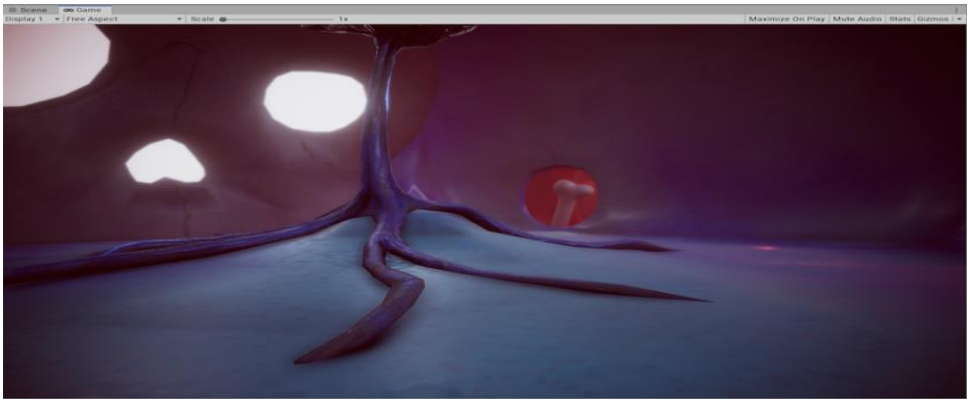
Slika 29: Prikaz Perspektivnog pogleda (Lijevo) i Ortogonalnog (Desno) [21]



Slika 30: Ista scena s pogledom od gore i sprijeda [21]

8.5 Simulacijski Pogled (eng. *„Game View“*)

Renderirani pogled kamera u simulaciji prikazan je na slici 31. Pokazuje finalni izgled simulacije [22] .Isto tako može, i ne mora pokazivati: *Gizmos* i slično.



Slika 31: Renderirani finalni izgled simulacije [22]

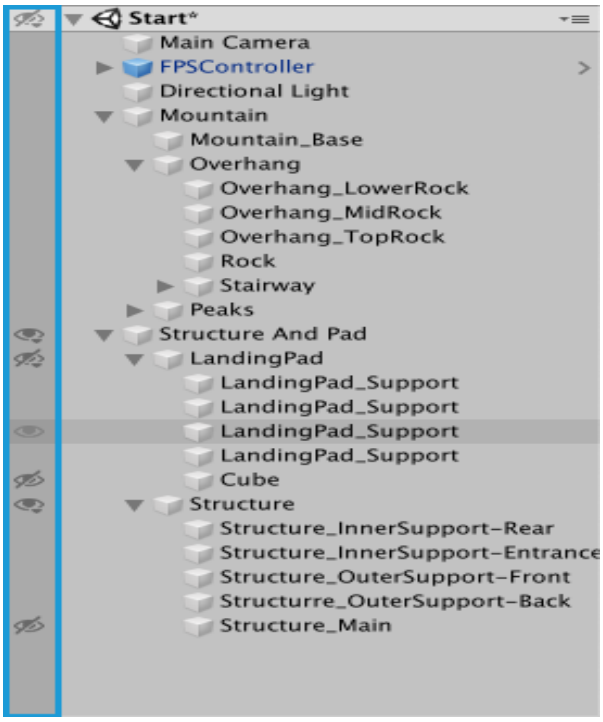
Play Mode

Slika 9: Play/Pause/Step

Gumbima na alatnoj traci kontrolira se simulacijski scenarij te sve promjene koje se naprave dok je simulacija pokrenuta se resetiraju kada se izađe iz reprodukcije.

8.6 Hijerarhijski Prozor (eng. *„Hierarchy Window“*)

Sadrži svaki *GameObjekt* u sceni kao što su modeli ili kamere. Može se koristiti za sortiranje i grupiranje *GameObjekata* koji se koriste u sceni. Kada se dodaju ili uklanjaju *GameObjekti* u *sceni*, također se dodaju ili uklanjaju u hijerarhijskom prozoru. Hijerarhijski prozor [23] također može sadržavati i druge scene gdje svaka scena ima svoje *GameObjekte*.



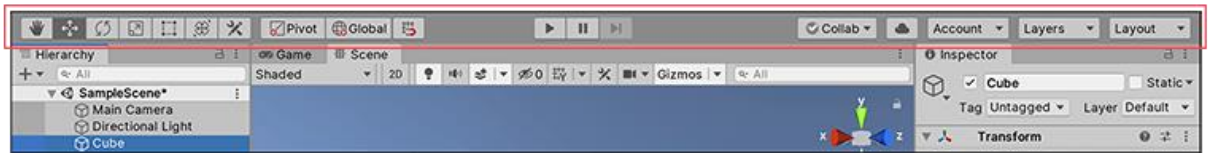
Slika 32: Hijerarhijski Prozor (Hierarchy Window)

8.7 Roditeljstvo (eng. *„parenting“*)

Unity koristi koncept Roditelj-Dijete (eng. *„Parent-Child“*) za grupiranje *GameObjekata*. *GameObjekt* može imati druge *GameObjekte* koji nasljeđuju njegova svojstva. *GameObjekti* se isto tako mogu povezati zajedno kako bi pomogli u premještanju, *skaliranju* ili *transformiraju* kolekcije *GameObjekata*.

8.8 Alatna trak (eng. *„Toolbar“*)

Nalazi se na vrhu Unity Uređivača. To nije prozor i jedini je dio sučelja koji se ne može uređivati. Prikazuje često korištene prečace kao što su pomicanje, rotiranje, skaliranje, pokretanje, pauziranje i slično, ali se rijetko koristi praksi jer su prečaci (eng. *„shortcuts“*) brži. (slika 33)

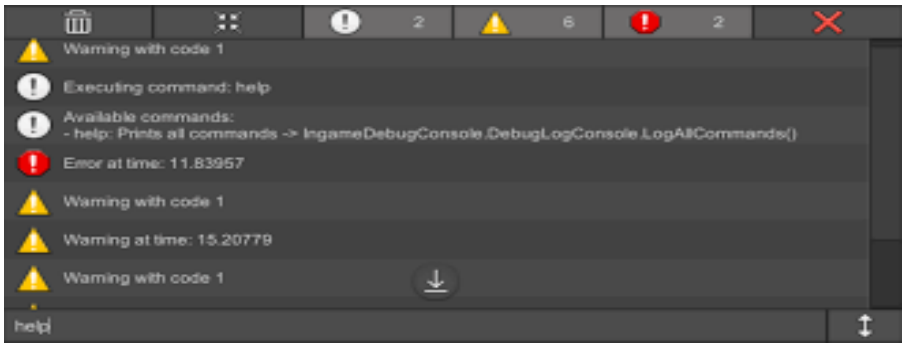


Slika 33: Alatna Traka (Toolbar)

8.9 Prozor Konzole (eng. *„Console Window“*)

Prozor Konzole [24] (slika 34) pokazuje greške, upozorenja i slične poruke koje je napravio *Unity*. *Developer* ih koristi za praćenje i otklanjanje grešaka (eng. *„debugging“*) igre. Poruke mogu dolaziti od *engine-a*, kompajlera (eng. *„compiler“*) ili *developer* može sam pisati u konzolu. Poruke u konzoli dijele se na:

- Informacije (Debug.Log())
- Upozorenja (Debug.LogWarning)
- Greške (Debug.LogError())



Slika 34: Prozor Konzole (Console Window) [24]

9. Strukturiranje u Unity-u

U ovom projektu napravljena je simulacijska igra gdje će se koristiti modeli napravljeni u *Blenderu*.

Prije nego se počnu pisati skripte, treba biti upoznat s orijentiranjem u *Unity-u*. Kreće se tako da se dodaje nekoliko objekata te se na njih stavljaju materijali. Uređuje se *Transform* (pozicija, rotacija i veličina nekog objekta). Nekim *GameObjektima* dodano je Kruto tijelo (eng. „*Rigidbody")* kako bi tijekom sudaranja s drugim *GameObjektom* došlo do promjene u ponašanju (npr. metak pao kada udari u nešto). Nakon upoznavanja interface *Unity-a* i nekih osnovnih, a nužnih funkcije koje nam *Unity* omogućuje, kreće se s upoznavanjem jezika.

Kada se upozna s objektno orijentiranim dijelom jezika i susretne s nekim osnovama programiranja kao što su varijable, metode, petlje i slično, kreće se s namjerom da se doda neka funkcija objektu.

Prvo je napravljena kapsula (igrač) i *scena* (streljana) koje su korištene kao prototip za testiranje funkcionalnosti dok se ne završi stvaranje modela koji će se koristiti. Nakon toga napravljeno je nekoliko manjih stvari poput pomicanja igrača. Kreirani su razni objekti koji će služiti kao zamjenski dok se ne ubace modeli kako bi se moglo isprobavati radi li sve kako treba.

Kamera je dodana igraću da se pomiče s njim i dodana joj je skriptu za rotaciju na miš. Kako bi metak pao na sudar, meta ili neki drugi pogođeni dio moraju imati *Collider* (fizika za interakciju dvaju objekata u dodiru jedan s drugim) na sebi.

Napravljeno je ispaljivanje objekta (pucanje) pomoću sile (eng, *„force"*), ali je kasnije zamijenjeno na bacanje zraka (eng. „*raycast*") kako bi se smanjila opterećenost na procesor (*Fixed update* testiranja te interpolaciju metka u slučaju preskakanja mete) te je bio plan da metak *odmah* pogodi metu jer je vrijeme putanje metka zanemarivo.

Raycast funkcija radi tako da „zraka" očita prvu metu. *Raycast* kao metoda vraća *boolean* da li je pogodio nešto po zadanim parametrima funkcije. U strukturi *Raycastinfo*

(*Raycastinfo* je struktura s 12 članova) prosljeđuje informaciju o pogođenome objektu i lokaciji te još dodatne informacije.

Jedna od težih stvari, a nužna za ovaj projekt bila je kreiranje funkcije za izvlačenje (eng. *„pooling“*) nekog objekta. *Pooling* je uzorak (eng. „*pattern*“) koji se često koristi u izradi igara kako bi neke „teške“ funkcije (*Instantiate()*) prebacio iz *update-a* (svaka sličicu) u *Awake* (prije prve slike) da ne bi došlo do *fps* pada (manji performans) svaki puta kada se ispaljuje metak ili generira rupu metka).

Ovaj bazen (eng. *„pool“*) svaki puta kada treba ispaliti metak, da već pripremljeni metak i ispali ga, a metak na sebi ima skriptu da se isključi (eng. *„disable“*) čime se automatski vrača u *pool*. Ako se potroše svi metci (ili bilo koji povučeno objekt) onda *pool* koristi stvaranje (eng. *„instantiate“*), te tako način proširuje *pool*, ali potencijalno dolazi do pada fps-a. Na samome kraju napravljeno je da kada igrač ispali određenu količinu metaka u metu, ona se zamjeni novom.

9.1 Unity Components (Unity komponente)

Transform – 4X4 Matrica koja sadrži: poziciju, rotaciju i veličinu *GameObjekta* te također *parenting* informacije. Svaki *GameObjekt* ima *Transform*.

Rigidbody (2D) – dodaje fiziku na *GameObjekt* te samim time otkrivanje *Collisiona*/*Triggera*. 2D varijanta ne može surađivati s običnom (3D) varijantom. Ima *AddForce()* (dodaj silu) koji će utjecati na brzinu (eng. „*velocity*“), te *AddForceAtPoint()* koji može utjecati i na brzinu pod kutom (eng. „*angular velocity*“).

CharacterController – Dodaje kapsuli *Collider* i pojednostavljene mogućnosti fizike, namijenjeno za bazno kretanje u *world space sceni*.

MeshRenderer – Prikazuje 3D model iz *Mesh* Filtera i sadrži sve postavke kako se model prikazuje (materijali, osvjetljenje i sjene…)

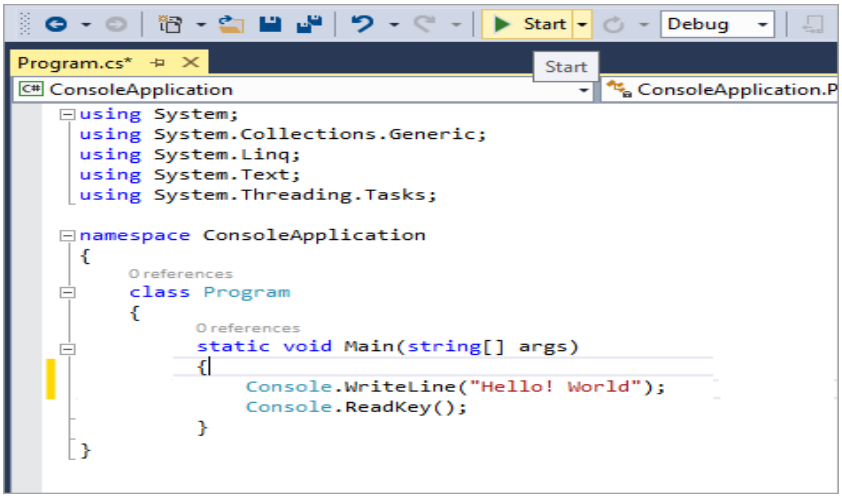
Collider (*box*, *sphere, capsule*, *mesh*, *convex mesh*) – Može biti *collider* ili *trigger* (*trigger* je *collider* koji očitava *Enter/Stay/Exit*, ali ne blokira). Otkriva sudare te ograničava kretanja *Rigidbody-a* i *CharacterController-a* u slučaju *collidera*.

10. Osnovno o C#-u

To je objektno orijentirani programski jezik koji je stvorio *Microsoft* i koji radi na *.NET Framework*. C# [17] ima korijene iz obitelji C, a jezik je blizak ostalim popularnim jezicima kao što su C++ i Java. Prva verzija objavljena je 2002. Najnovija verzija, C# 9, objavljena je u rujnu 2020. Sintaksa C# jezika prikazana je na slici 35.

C# koristi se za:

- Mobilne aplikacije
- Desktop aplikacije
- Web aplikacije
- Web usluge
- Web stranice
- Igre
- VR
- Aplikacije baza podataka
- I puno više.



Slika 35: Sintaksa C# jezika [17]

Vrste programiranja:

Strong typing (stroga pravila kod kompajliranja), ***Lexically scoped*** (asocijativan jezik koji pokazuje postojeće varijable i slično), ***imperative*** (imperativno programiranje gdje se mijenja stanje programa),
declarative (strukturirano programiranje elemenata bez opisa toka), ***functional*** (matematičko funkcijska orijentacija), ***generic*** (tip algoritma koji se piše u terminu- stavke koje se opisuju kasnije te se onda pojavljuju kao parametri koje treba definirati), ***object-oriented*** (najvažnija stavka ovog jezika, bazira se na konceptu objekta koji sadrži razna svojstva koje je potrebno definirati), ***component-oriented*** (orijentiran na primjeni kroz razne komponente).

U ovom projektu korišten je *Visual Studio Community* [25] za C#.

10.1 Skripte (eng. *„Scripts“*)

Skriptiranje je bitan dio svih igara. Čak i one najjednostavnije igre trebaju skripte za svoje funkcioniranje. Skripta se koriste kako bi dobili željeno ponašanje *GameObjekta* i njegovih komponenti. Primjer skripte prikazan je na slici 36.

Skriptiranje u *Unity-u* se razlikuje od običnoga programiranja u osnovnome dijelu gdje se ne treba kreirati „Kod“ da se pokrene aplikacija već to *Unity* napravi za nas te se mi samo usredotočimo na razradu simulacije (eng. *„simulation“*) kroz skripte.

Unity radi tako da prvo očita sve podatke koji se nalaze u Sceni kao što su objekti, svjetla, načine ponašanja te sve *procesuira*. Pokreče sličicu po sličicu (eng. *„frame“*), a mi odlučujemo preko napisanih „*kodova*“ gdje i kako izvesti simulaciju.

Kada projekt ima 1000 ako ne i više linija koda u skripti onda kod mora biti pregledan jer često će ljudi raditi na bazi *koda* (eng. *„code base“*) te mora biti čitljiv i samome developeru kada se nakon duljeg vremena vrati na njega. Optimizacija i čitljivost su različite stvari pa je ponekad bolje žrtvovati pokoju milisekundu u korist čitljivosti. Isto tako *Unity* ima svoj skupljač smeća (eng. *„garbage collector“*) te se za to ne treba brinuti.

```
void FixedUpdate()
{
    grounded = Physics2D.OverlapCircle(groundCheck.position, groundRadius, whatIsGround);

    anim.SetBool("Ground", grounded);

    if (grounded)
        doubleJump = false;

    anim.SetFloat("Speed", GetComponent<Rigidbody2D>().velocity.y);

    float move = Input.GetAxis("Horizontal");

    GetComponent<Rigidbody2D>().velocity = new Vector2(move * maxSpeed, GetComponent<Rigidbody2D>().velocity.y);

    anim.SetFloat("Speed", Mathf.Abs(move));

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

}
```

Slika 36: Primjer skripte

10.2 Varijable (eng. *„Variables“*)

Kada se stvara skripta, u osnovi stvara se vlastiti tip komponente koja se može pridružiti objektima igre kao i svaka druga komponenta. Baš kao što i druge komponente imaju svojstva koja se mogu uređivati preko inspektora, tako se može dopustiti da se vrijednosti u našoj skripti uređuju preko inspektora.

Svaka *varijabla* može u sebi sadržavati neku vrijednost. Uvijek počinju s malim slovom te se po standardu deklariraju na vrhu. O imenima *varijabli* odlučujemo mi, ali pratimo neka pravila. Tip, veličina i opseg varijable prikazani su na slici 37.

Imamo nekoliko vrsti vidljivosti *varijabli*, ali većinom se koriste privatne (eng. *„private“*) i javne (eng. *„public“*). Kod privatnih i javnih *varijabli* razlika je u tome što se javne mogu vidjeti i mijenjati u inspektoru, ali bolje je voditi propisnu *enkapsulaciju* dok vidljivost u funkcijama definiramo sa [*SerializeField*] da bude vidljiva te [*HideInInspector*] za javne koju ne želimo da bude vidljiva.

C# programski jezik podržava dva osnovna tipa: vrijednosni i referentni tip. Razlika između njih leži u načinu na koji se njima upravlja.

Vrste tipova varijabli:

Vrijednosni:

- Int (cjelobrojna vrijednost)
- Char (znak)
- String (niz znakova)
- Float (decimalna vrijednost)
- Double (broj s pomičnim zarezom)
- Struct (kopija u memoriji)

Type	Size in bits	Minimum	Maximum
char	8	-128	127
unsigned char	8	0	255
short	8	-128	127
unsigned short	8	0	255
int	16	-32768	32767
unsigned int	16	0	65535
long	32	-2,147,483,648	2,147,483,647
unsigned long	32	0	4,294,967,295
float	32	-3.40282 X 10 ^ 38	3.40282 X 10 ^ 38
double	64	-1.79769 X 10 ^ 308	1.79769 X 10 ^ 308
pointer or void	system dependent	?	?

Table 1 - basic C types

Slika 37: Tipovi Varijabla

Referentni:

- **Class**
- **Array**
- **Delegate**
- **Interface**
- **Few more**

10.3 Funkcije (eng. *„Functions“*)

Skripte manipuliraju *varijablama* pomoću funkcija. Postoji niz funkcija koje se automatski pokreću unutar *Unity-a.*

```
/*
    Awake()
    Start()
    Update ()
    FixedUpdate()
    LateUpdate

*/
```

Slika 38: Osnovne Funkcije

Lista (slika 38) koja pokazuje redoslijed izvršenja funkcija u *Unity-u.*

Awake():

Awake se poziva samo jednom kada se *GameObjekt* s tom komponentom *instancira*. Ako je *GameObject* neaktivan, tada se neće pozivati dok se ne aktivira. Međutim, *Awake* se poziva čak i ako je *GameObjekt* aktivan, ali komponenta nije omogućena (s malim potvrdnim okvirom pored imena). *Awake* se može koristiti za inicijalizaciju svih varijabli kojima trebate dodijeliti vrijednost.

Start():

Start je sličan *Awake*, *Start* će se pozivati ako je *GameObjekt* aktivan (eng. *„active“*), ali samo ako je komponenta omogućena.

Update();

Update se poziva jednom po slici (once per frame). Ovdje se stavlja „kod“ da bi se definirala logika koja se kontinuirano izvodi, poput animacija, ali i drugih dijelova igre koji se moraju stalno ažurirati.

FixedUpdate();

Služi onda kada treba da bilo koji dio *engine-fizike* obavi neki posao.

LateUpdate():

Funkcija slična funkciji *Update*, ali će *LateUpdate* biti pozvan na kraju sličice. *Unity* će pregledati sve Objekte, pronaći sve *Update-e*, te pozvati sve *LateUpdate-e*. Vrlo korisno u slučaju kamera u *Sceni*. Kao primjer, recimo da se želi pomaknuti igrača u simulaciji. Slučajno se sudara s drugim igračem te završava na drugoj lokaciji. Ako se pomakne kamera u isto vrijeme kada i igrač, kamera će se „zatresti“ i ne bi bila na mjestu gdje treba biti. Stoga je potrebna druga petlja.

OnDrawGizmos():

Poziva se svaki *frame* (slike). *Gizmos* se koriste za pomoć u vizualnom uklanjanju pogrešaka ili postavljanju u prikazu scene. O ovome projektu je korišten za *interakciju GameObjekata*.

OnEnable():

Ova se funkcija poziva kada objekt postane omogućen i aktivan.

OnDisable():

Funkcija koja se poziva kada objekt postane onemogućen i neaktivan.

OnTrigger():

Kada se neki *GameObjekt* sudari s drugim *GameObjektom* onda *Unity* koristi ovu funkciju. Oba *GameObjekta* moraju sadržavati komponentu *Collider* te jedan mora imati *Rigidbody* i *Collider.isTrigger* aktivan. Ako oba *GameObjekta* imaju *Collider.isTrigger* aktivan onda se sudaranje neće dogoditi kao i ako oba *GameObjekta* nemaju *Rigidbody*.

OnCollisionEnter():

Poziva se onda kada neki *Collider*/*Rigidbody* počne dodirivati drugi *Rigidbody*/*Collider*.

10.4 Klase (eng. *„Classes“*)

Klase su kao neki nacrti naših objekata. U osnovi sve skripte započet će deklaracijom klase. *Unity* to automatski stavlja u skriptu čim se stvara nova C# skripta. Ova klasa dijeli ime kao datoteka u kojoj je skripta. To je jako važno jer ako se promijeni ime jednog, mora se promijeniti ima i drugog da bi normalno radilo.

Klasa se isto tako može reći da je neki kontejner u koji se spremaju *varijable* i funkcije. Klasa je dobar način grupiranja stvari koje rade zajedno.

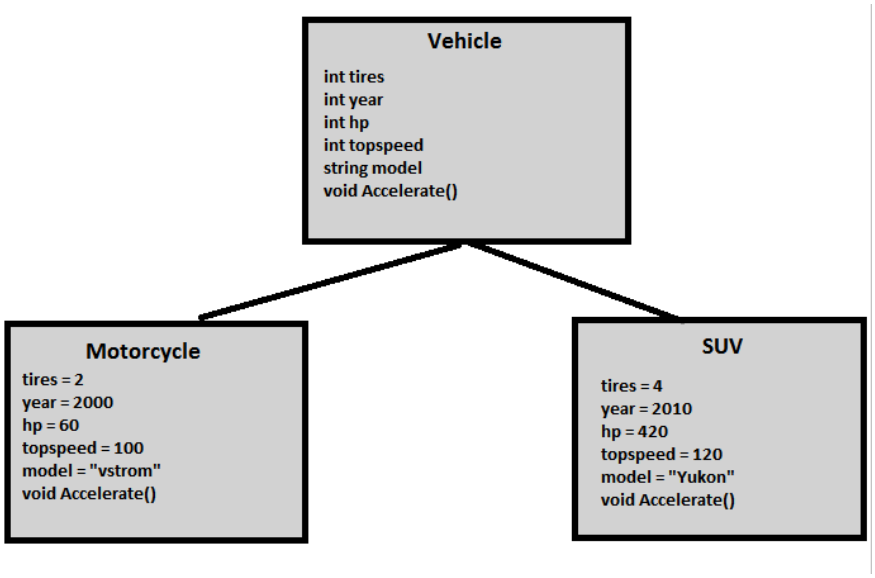
```
1 using UnityEngine;
2 using System.Collections;
3
4 [System.Serializable]
5 public class DataClass {
6     public int myInt;
7     public float myFloat;
8
9 }
10
11 public class DemoScript : MonoBehaviour {
12
13     public Light myLight;
14     public DataClass myClass;
15
16     void Awake () {
17         int myVar = AddTwo(9,2);
18         Debug.Log(myVar);
19
20     }
21
22     void Update () {
23         if (Input.GetKeyDown ("space")) {
24             MyFunction ();
25
26         }
27
28     }
29
30     void MyFunction () {
31         myLight.enabled = !myLight.enabled;
32     }
```

Slika 39: Primjer korištenja klase

Ako se u *Unity-u* radi neka svoja klasa, mora se staviti upit *serialize it* kako bi bila prebačena (eng. *„convert“*) u tip podataka koji *Unity* prepoznaje. Primjer klase prikazan je na slici 40.

Korištene custom klase u ovom projektu:

- **PoolManager : MonoBehaviour**
- **QueuePool : MonoBehaviour**
- **BulletBehaviour : MonoBehaviour**
- **CameraController : MonoBehaviour**
- **PlayerController : MonoBehaviour**
- **PlayerRaycaster : MonoBehaviour**
- **ShootingController : MonoBehaviour**
- **SOWeaponData: ScriptableObject**
- **TargetBehaviour : MonoBehaviour**
- **TargetButtonBehaviour : MonoBehaviour**
- **WeaponController : MonoBehaviour**

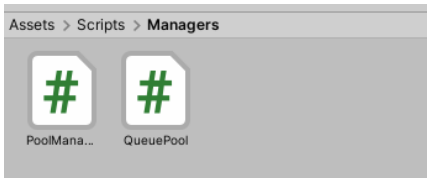


Slika 40: Primjer klase

11. Programiranje u C#

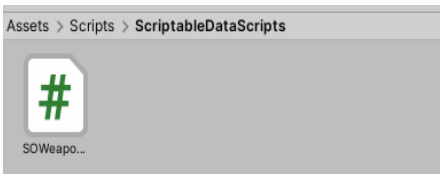
U *Unity-u* sve skripte stavljene su u određeni folder ovisno o tome što rade, kako bi projekt bio organiziraniji. Slike 41-45 prikazuju skripte korištene u projektu.

Managers:



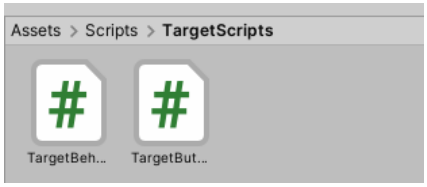
Slika 41: Manager Skripte

ScriptableDataScripts:



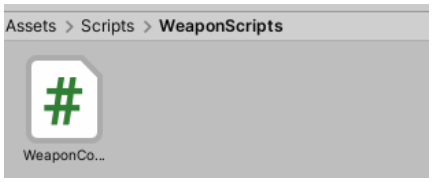
Slika 42: Skriptable objekt skripte

TargetScripts:



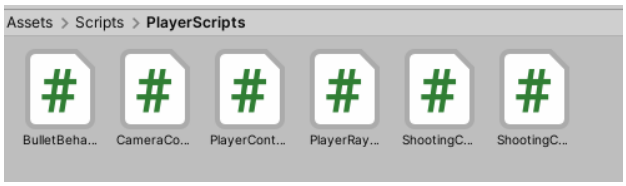
Slika 43: Skripte za mete

WeaponScripts:



Slika 44: Skripte za oružja

PlayerScripts:



Slika 45: Skripte za igrača

PlayerController : MonoBehaviour

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

[RequireComponent(typeof(CharacterController))]
Unity Script | 0 references
public class PlayerController : MonoBehaviour
{

    [SerializeField]
    private float _movementSpeed = 10.0f;

    [SerializeField]
    private Vector3 _gravity = Physics.gravity;

    [SerializeField]
    private Transform _cameraTransform;

    private Vector3 _movementVector;

    private CharacterController _characterController;

    Unity Message | 0 references
    private void Awake()
    {
        _characterController = GetComponent<CharacterController>();
        if(_characterController == null)
        {
            Debug.LogError("Players Character Controller not found");
        }
    }

    Unity Message | 0 references
    private void Update()
    {
        _movementVector = new Vector3(
            Input.GetAxis("Horizontal"),
            0.0f,
            Input.GetAxis("Vertical"));

        _movementVector.Normalize();

        _movementVector *= _movementSpeed;

        _movementVector = _cameraTransform.TransformDirection(_movementVector);

        _movementVector += _gravity;

        _characterController.Move(_movementVector * Time.deltaTime);
    }
```

Slika 46: Skripta za kontroliranje igrača

Sadrži varijablu za brzinu pomicanja igrača kao i za gravitaciju. Svaka *sličica (eng. „frame)* provjerava *movement input* (unos) od igrača (eng. *„player“*), normalizira vektor rezultata te preko *CharacterController-a* pomiče ovisno o rotaciji kamere odnosno smjera u kojem igrač gleda.

Input.GetButton() vraća *bool* pa tako preko *if-a* očitavamo unos i ako je očitan, pokreče se igrač.

CameraController : MonoBehaviour

Sadrži *varijable* za brzinu rotacije kao i za maksimalnu i minimum dopuštenu rotaciju kamere.

Čita input za rotaciju kamere (kretanje miša u ovom primjeru) te okreće kameru za učitanu vrijednost, drži rotaciju u granicama od -180 do 180, te postavlja rotaciju na zadanu vrijednost, zadanom brzinom i unutar željenih granica.

```
Unity Message | 0 references
private void Update()
{
    float mouseInputX = Input.GetAxis("Mouse Y") * _rotationSpeed;
    float mouseInputY = Input.GetAxis("Mouse X") * - _rotationSpeed;

    Vector3 finalCameraRotation = transform.localEulerAngles;

    finalCameraRotation.x += mouseInputX;
    finalCameraRotation.y += mouseInputY;

    if(finalCameraRotation.x > 180)
    {
        finalCameraRotation.x -= 360;
    }

    if(finalCameraRotation.x < -180)
    {
        finalCameraRotation.x += 360;
    }

    if(finalCameraRotation.y > 180)
    {
        finalCameraRotation.y -= 360;
    }

    if(finalCameraRotation.y < -180)
    {
        finalCameraRotation.y += 360;
    }

    finalCameraRotation.x = Mathf.Clamp(finalCameraRotation.x, -_maxRotationX, _maxRotationX);
    finalCameraRotation.y = Mathf.Clamp(finalCameraRotation.y, -_maxRotationY, _maxRotationY);
    finalCameraRotation.z = 0.0f;

    transform.localRotation = Quaternion.Euler(finalCameraRotation);
}
```

Slika 47: Skripta za pomicanje kamere

Skripta PoolManager: MonoBehaviour

PoolManager je ovdje klasa u skripti *PoolManager* na osnovnu klasu *MonoBehaviour*. Definirana je lista gdje će se spremati svi objekti koji se nalaze u tzv. „bazenu“ (*pool*) koji

će se izvlačiti po potrebi. Isto tako napravljene su *varijable* za objekte koji će se izvlačiti i za njihovo pozicioniranje te u slučaju da nema objekta za izvući, on će napraviti novi objekt i dodati ga bazenu.

Skripta je napravljena za objekte koji će se ponavljati (metci) te da ne bi previše opteretili *performansu* (eng. *„performance“*) računala, ova skripta micat će nekorištene objekte za kasniju upotrebu te tako izbjeći opterećivanje računala.

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

Unity Script | 8 references
public class PoolManager : MonoBehaviour
{
    11 references
    private class PoolInstance
    {
        //List containing all of objects in pool
        private List<GameObject> _pooledObjects = new List<GameObject>();

        //Key object which is pooled, reference for creeating new ones
        private GameObject _objectToPool;

        //Transform which serve as a parent for created objects
        private Transform _parentTransform;

        /// <summary>
        /// Constructor for Pool instance, creates pool of selected object in selected amount
        /// </summary>
        /// <param name="objectToPool">Prefab of the object to pool</param>
        /// <param name="amountToPool">How many instances should be created in time of initial pooling</param>
        /// <param name="parentTransform">Transform of a parent under which objects will be in hierarchy</param>
        4 references
        public PoolInstance(GameObject objectToPool, int amountToPool, Transform parentTransform)
        {
            _parentTransform = parentTransform;
            _objectToPool = objectToPool;
            for(int i = 0; i < amountToPool; i++)
            {
                GameObject instantiatedObject = Instantiate(_objectToPool, parentTransform);
                instantiatedObject.SetActive(false);
                _pooledObjects.Add(instantiatedObject);
            }
        }


        /// <summary>
        /// Finds the first inactive object and pool and returns it. If none is found, creates new one, returns it and adds to pool.
        /// </summary>
        /// <returns>First available inactive GameObject</returns>
        1 reference
        public GameObject GetNextAvailableObject()
        {
            for(int i = 0; i < _pooledObjects.Count; i++)
            {
                if(!_pooledObjects[i].activeInHierarchy)
                {
                    return _pooledObjects[i];
                }
```

Slika 48: Skripta za izvlačenje različitih objekata

Singleton (Postoji samo jedna u sceni te mu se može pristupiti preko *public static* reference od bilo kuda. *Object pool pattern* koji unaprijed očitava *GameObjekte* za kasniju upotrebu.

Ova varijanta nakon iskorištenog zadnjeg objekta iz tzv. bazena (*pool*) stvara (*Instantiate*) novi *GameObjekt* te ga predaje.

Ovoj skripti može pristupiti bilo koja skripta dok god je *PoolManager* u sceni. Nalazi se u ovom folderu kako ne bi svatko imao pristup za stvaranje i ponavljanje svojih objekata.

QueuePool : Monobehaviour

Objekt pool pattern koji već unaprijed učitava *GameObjekt-e* za kasniju upotrebu. Deklarirane su *varijable* koje se nalaze u ovoj klasi kao što je količina. Ova varijanta nakon iskorištenoga zadnjeg Objekta iz bazena (*pool*) predaje prvi objekt bez obzira da li je taj aktivan ili ne.

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

Unity Script | 0 references
public class QueuePool : MonoBehaviour
{

    [System.Serializable]
    2 references
    public class Pool
    {
        public string tag;
        public GameObject prefab;
        public int size;
    }

    public List<Pool> pools;

    public Dictionary<string, Queue<GameObject>> poolDictionary;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        poolDictionary = new Dictionary<string, Queue<GameObject>>();

        foreach(Pool pool in pools)
        {
            Queue<GameObject> objectPool = new Queue<GameObject>();

            for(int i = 0; i < pool.size; i++)
            {
                GameObject obj = Instantiate(pool.prefab);
                obj.SetActive(false);
                objectPool.Enqueue(obj);
            }

            poolDictionary.Add(pool.tag, objectPool);
        }

    }


}
```

Slika 49: Skripta za ponovnu upotrebu iskorištenih objekata

TargetBehaviour : MonoBehaviour

Koristi se prilagođeni tip *enum* s 4 stanja (*Ready*, *IsMovingCloser*, *IsClose*, *IsReseting*) koje u sebi imaju vrijednost koja kreće od 0 (*Ready* ima 0 dok *IsReseting* već ima vrijednost 3.

```
[SerializeField]
private Transform _farPosition;

[SerializeField]
private Transform _closePosition;

[SerializeField]
private float _transitionDuration = 2.0f;

[SerializeField]
private iTween.EaseType _movingEaseType = iTween.EaseType.easeInOutCubic;

private TargetState _currentTargetState = TargetState.Ready;
private List<GameObject> _bulletHoles = new List<GameObject>();

1 reference
public void ButtonPressed()
{
    switch (_currentTargetState)
    {
        case TargetState.Ready:
            iTween.MoveTo(gameObject, iTween.Hash(
                    "position", _closePosition.position,
                    "easetype", _movingEaseType,
                    "time", _transitionDuration,
                    "oncomplete", "OnMovingCloserComplete"
                ));
            _currentTargetState = TargetState.IsMovingCloser;
            break;
        case TargetState.IsMovingCloser:

            break;
        case TargetState.IsClose:
            iTween.MoveTo(gameObject, iTween.Hash(
                    "position", _farPosition.position,
                    "easetype", _movingEaseType,
                    "time", _transitionDuration,
                    "oncomplete", "IsResetComplete"
                ));
            _currentTargetState = TargetState.IsReseting;
            break;
        case TargetState.IsReseting:

            break;
    }
```

Slika 50: Skripta za ponašanje mete:

Sadrži podatke o „meti“ (udaljenu i približnu poziciju, trenutni status mete, rupe od metaka) te funkcionalnost koja se poziva iz *TargetButtonBehavioura*, granano po *switch statementu*.

Preko *Switch-a* provjerava se nekoliko slučajeva u kojem se ispituje položaj „mete“ i ovisno o kojem slučaju se radi, izvršit će se drukčiji set naredbi te preko break će izaći iz *switcha*.

Logika za pomicanje mete:

- **Spremno (eng. *„Ready“*)** – da li je meta spremna te u slučaju da je, pritiskom gumba ona se približava.

- **Da li se približava (eng. *„IsMovingCloser“*)**- tranzicijsko stanje u kojem se na gumb ništa ne događa. Animacija (*iTween*) će se pobrinuti da kada je animacija gotova da se stanje promjeni.

- **Blizu je (eng. *„IsClose“*)** – Da li je meta blizu i ako je, onda prelazi u položaj *IsResetting*.

- **Resetira se (eng. *“IsResetting“*)** – Isto kao i kod stanja *IsMovingCloser*. Tranzicijsko stanje u kojem se ništa ne događa pritiskom gumba već pri završetku animacije.

TargetButtonBehaviour : MonoBehaviour

Sadrži referencu na svoj *TargetBehaviour*, na igrača *interakciju* (*PlayerRaycaster*) pokreče metu i animaciju pritiska gumba.

```
[SerializeField]
private TargetBehaviour _buttonTarget;

[SerializeField]
private Vector3 _pushedButtonOffset = new Vector3(0.0f, 0.0f, 0.0f);

[SerializeField]
private float _pushAnimationDuration = 0.5f;

[SerializeField]
private iTween.EaseType _pushEaseType = iTween.EaseType.easeInOutCubic;

private bool _isAnimationRunning = false;

1 reference
public void PushButton()
{
    if(_buttonTarget != null)
    {
        _buttonTarget.ButtonPressed();
    }

    if (!_isAnimationRunning)
    {
        iTween.MoveTo(gameObject, iTween.Hash(
                "position", transform.position + _pushedButtonOffset,
                "time", _pushAnimationDuration,
                "easetype", _pushEaseType,
                "oncomplete", "ReturnButton"
            ));
        _isAnimationRunning = true;
    }
}

0 references
private void ReturnButton()
{

    iTween.MoveTo(gameObject, iTween.Hash(
        "position", transform.position - _pushedButtonOffset,
        "time", _pushAnimationDuration,
        "easetype", _pushEaseType,
        "oncomplete", "ResetButton"
    ));
}
```

Slika 51: Skripta za ponašanje gumba

Koristi se *iTween.easetype* za odabir krivulje (npr. linearna) te tako kreče „meta“. Sadržava *varijable* za dugme (*_buttonTarget*), pomake gumba (*_pushedButtonOffset*), trajanje animacije (*_pushAnimationDuration*) te *bool* koji kontrolira da li je animacija gumba aktivna da se ne bi zablokirala, tj. pokrenula nova dok traje stara (*iTween* nema takvu zaštitu). Preko *null* reference *checkera* koji provjerava da li referenca na *TargetBehaviour* postoji i drugi if koji provjerava da li se animacija već vrti kako se ne bi pokrenula 2 puta.

SOWeaponData : ScriptableObject

Instance klase sadrže sve potrebne informacije o oružju (Ime, čekanje prije svakog pucnja, punjenja, količina metaka, trenutna količina metaka, ciljanje i provjera da li je oružje snajper ili ne),

postavljaju se u *Unity Editor* inspektoru te svaka stavka ima svoj *Get property*. Informacije se prilikom uzimanja oružja preko *Weapon Controllera* prebacuju u *ShootingController*.

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

[CreateAssetMenu(fileName = "WeaponData00", menuName = "ScriptableData/New Weapon Data", order = 51)]
Unity Script | 4 references
public class SOWeaponData : ScriptableObject
{

    [SerializeField]
    private string _weaponName;
    2 references
    public string WeaponName { get { return _weaponName; } }

    [SerializeField]
    private float _secondsBetweenShots;
    1 reference
    public float SecondsBetweenShots { get { return _secondsBetweenShots; } }

    [SerializeField]
    private float _reloadTimeInSeconds;
    2 references
    public float ReloadTimeInSeconds { get { return _reloadTimeInSeconds; } }

    [SerializeField]
    private int _ammoPerClip;
    3 references
    public int AmmoPerClip { get { return _ammoPerClip; } }

    [SerializeField]
    private int _currentAmmoInClip;
    5 references
    public int CurrentAmmoInClip { get { return _currentAmmoInClip; } set { _currentAmmoInClip = value; } }

    [SerializeField]
    private float _aimingFOV;
    0 references
    public float AimingFOV { get { return _aimingFOV; } }

    [SerializeField]
    private bool _isSniper = false;
    1 reference
    public bool IsSniper { get { return _isSniper; } }

}
```

Slika 52: Skripta za držanje podataka svakog oružja

WeaponController : MonoBehaviour

U ovoj skripti napravljena je *varijabla* za podatke o svakom oružju te *varijabla* za spremanje podataka. Koristi se nekoliko upita koji provjeravaju da li ima oružje u ruci, da li su uneseni podaci o određenom oružju (Ime i slično).

```
[SerializeField]
private Transform _weaponStoreTransform;

Unity Message | 0 references
private void Start()
{
    if(_weaponData != null)
    {
        _weaponData.CurrentAmmoInClip = _weaponData.AmmoPerClip;
    }
}

1 reference
public SOWeaponData PickUpWeapon(Transform newWeaponTransform)
{
    if(_weaponData != null)
    {
        transform.SetParent(newWeaponTransform);
        transform.localPosition = Vector3.zero;
        transform.localRotation = Quaternion.identity;

        return _weaponData;
    }
    else
    {
        Debug.LogError("Selected weapon is without data");
        return null;
    }
}

1 reference
public string GetWeaponName()
{
    if (_weaponData != null)
    {
        return _weaponData.WeaponName;
    }
    else
    {
        Debug.LogError("Selected weapon is without data");
        return null;
    }
}

1 reference
public void ReturnWeapon()
{
    transform.SetParent(_weaponStoreTransform);
    transform.localPosition = Vector3.zero;
    transform.localRotation = Quaternion.identity;
}
```

Slika 53: Skripta za ponašanje oružja

Drži referencu na svoj *SO instancu*, prilikom uzimanja oružja stavi ga kao roditelj na igrača te prilikom vraćanja *shooting controller* pozove odavde skriptu da se oružje vrati na mjesto i da pamti to mjesto.

PickUpWeapon ():

- Funkcija koja se poziva za skupljanje postojećeg oružja pritiskom tipke E. Igrač neće podići oružje ako već posjeduje oružje u ruci ili ako je previše udaljen od oružja. U ovom slučaju svaki puta kada igrač želi uzeti oružje, mora se približiti stolu na kojem su oružja.

ReturnWeapon ():

- Funkcija koja se poziva kako bi se oružje vratilo na početno mjesto pritiskom tipke Q. Igrač ne može vratiti oružje ako nema ni jedno oružje u ruci te ako je pokrenuta funkcija za punjenje oružja. Igrač u ovom slučaju ne treba doći blizu stola kako bi vratio oružje već je dovoljno da se pritisne tipka Q.

GetWeaponName ():

- Funkcija koja se poziva za podatke o oružju. Daje podatke o imenu oružja ako je igrač dovoljno blizu i ako je okrenut prema jednom od oružja.

ShootingController : Monobehaviour

Prilikom uzimanja oružja, povlači sve informacije potrebne za pucanje tog oružja. Sadrži *korutinu* koja kontrolira brzinu pucanja (eng. *„fire rate“*) i punjenje (eng. *„reloading“*). Upravlja *user interfaceom* ovisno o imenu oružja i broju metaka te pali/gasi potrebne UI panele.

Sadrži *Shoot* funkciju koja *raycastom* gađa metu, uzima metak iz bazena (*pool-a*) te ako pogodi metak stvara rupu od metka na meti samo.

```
private bool _isReloading = false;
private bool _isWeaponInHand = false;
private bool _isShootingCoroutineRunning = false;
private PlayerRaycaster _playerRaycaster;

private SOWeaponData _currentWeaponData = null;

Unity Message | 0 references
private void Awake()
{
    _playerRaycaster = GetComponent<PlayerRaycaster>();
    if(_playerRaycaster == null)
    {
        Debug.LogError("ShootingController couldn't find PlayerRaycaster");
    }
    _weaponInfoPanel.SetActive(false);
    _raycastInfoPanel.SetActive(true);
    _raycastInfoText.text = "";
}

Unity Message | 0 references
private void Start()
{
    PoolManager.Instance.AddNewPool(_bulletPrefab, _numberOfbulletsPooled);
    PoolManager.Instance.AddNewPool(_bulletHolePrefab, 50);
}

Unity Message | 0 references
private void Update()
{
    if ((Input.GetButtonDown("Shoot") || Input.GetButtonDown("Reload")) && _isWeaponInHand)
    {
        StartCoroutine(ShootingAndReloading());
    }

    else if(Input.GetButtonDown("DropWeapon") && !_isReloading && _isWeaponInHand)
    {
        _currentWeaponData = null;
        _weaponInfoPanel.SetActive(false);
        _raycastInfoPanel.SetActive(true);
        _playerRaycaster.ReturnCurrentWeapon();
        _isWeaponInHand = false;
    }
}

1 reference
public void InitNewWeapon(SOWeaponData weaponData)
{
    _currentWeaponData = weaponData;
    _weaponInfoPanel.SetActive(true);
    _raycastInfoPanel.SetActive(false);
```

Slika 54: Skripta za kontrolu pucanja

Unesene su *varijable* za razne podatke o oružju (brzina pucanja, broj metaka i slično). Koristi se upit if kako bi se provjerilo da li je tipka za pucanje ili punjenja stisnuta te ako je onda se pokreće. *Korotina* (eng. *„Courotine“*) *ShootingAndReloading*. Ovdje se koristi *while loop* koji će izvršiti neki posao ako je uvjet/izjava točna. Ako je kriva onda *loop* prestaje. U ovom slučaju dok god je stisnuta tipka za pucanje ili punjenja, provjerit će se nekoliko upita:

- Prvo upit provjerava da li je stisnuta tipka za pucanje ili punjenja te da je oružje u ruci. Ako je onda pokreće *korutinu ShootingAndReloading*.
- Dugi upit provjerava da li je stisnuta tipka za bacanje oružja, punjenja ili da je oružje u ruci. Ako se oružje puni nećemo moći baciti oružje kao ni ako ga nemamo u ruci.

Shoot():

Privatna funkcija koja služi za ispaljivanje metaka koje dohvaća preko *PoolManagera*. Isto tako *raycasta* (baca „zraku“) da vidi što je pogođeno te na mjestu udara stavlja dohvaćeni metak. Ako je meta pogođena, onda na mjestu udara stvara rupu od metka.

PlayerRaycaster : Monobehaviour

Kada oružje nije u ruci, provjerava da li se gleda prema nečemu s čime se može interaktirati (meta, gumb, oružje) te vrši potrebnu funkcionalnost zavisno o tipu koji prepoznaje preko *LayerMask-e*. S metom se ne može interaktirati direktno. Njega pomičemo gumbom.

```
[SerializeField]
private string _pickupWeaponPrefix = "Press [E] to pick up: ";

[SerializeField]
private string _pushButtonText = "Press [E] To Push the Button";


private bool _isWeaponInHand = false;
private ShootingController _shootingController;
private WeaponController _currentWeapon = null;


private void Awake()
{
    _shootingController = GetComponent<ShootingController>();
    if(_shootingController == null)
    {
        Debug.LogError("PlayerRaycaster couldn't find Weapon Controller");
    }
}

private void Update()
{
    if(!_isWeaponInHand)
    {
        RaycastHit hitInfo;
        if (Physics.Raycast(_raycastStartingpoint.position, _raycastStartingpoint.forward, out hitInfo, _interactionRange,
            _interactableObjectsLayer + _interactableWeaponLayer))
        {
            if((Mathf.Pow(2, (hitInfo.transform.gameObject.layer)) == _interactableWeaponLayer) && (_shootingController != null))
            {
                _currentWeapon = hitInfo.transform.GetComponent<WeaponController>();
                if(_currentWeapon != null)
                {
                    _interactionInfoText.text = _pickupWeaponPrefix + _currentWeapon.GetWeaponName();
                    if(Input.GetButtonDown("Interact"))
                    {
                        _shootingController.InitNewWeapon( _currentWeapon.PickUpWeapon(_weaponInHandTransform));
                        _isWeaponInHand = true;
                    }
                }
            }
            else if(Mathf.Pow(2, (hitInfo.transform.gameObject.layer)) == _interactableObjectsLayer)
            {
                _interactionInfoText.text = _pushButtonText;
                if(Input.GetButtonDown("Interact"))
                {
                    TargetButtonBehaviour button = hitInfo.transform.GetComponent<TargetButtonBehaviour>();
                    if(button != null)
                    {
                        button.PushButton();
                    }
```

Slika 55: Skripta za interakciju putem zrake (raycast)

Sadrži *OnDrawGizmos* koji se ne prikazuje u igri, ali u pogledu scene crta liniju zelenom bojom (boja se bira proizvoljno) koja očitava *interaktivne* objekte.

12. Zaključak

Proces stvaranja vlastitih modela je kompleksan, dok izrada simulacije korištenjem tih modela je još zahtjevniji posao. Inače se ovakvi projekti podijele po granama u timu i svatko radi na jednom dijelu. Cilj ovog završnog rada bio je objasniti osnovne stvari koje nude *Blender* i *Unity* i to pokazati na svojoj izradi modela i simulacije.

U današnje vrijeme tehnologija jako napreduje iz godine u godinu i softveri se mijenjaju pa uvijek treba pratiti promjene pogotovo kod programiranja.

U daljnjem razvoju igre moglo bi se napraviti da igrač prolazi razine i skuplja bodove. Mogao bi se napraviti model igrača da se ne koristi kapsula kao u ovoj igri. Zvuk i animacija bi se mogli bolje uskladiti da izgleda prirodnije.

13. Literatura

[1] »Blender,« Blender, Prikupljeno: 2021. Dostupno: https://www.blender.org/.

[2] »Unity,« Unity Technologies , Prikupljeno: 2021. Dostupno: https://unity.com/.

[3] »Wikipedia,« Blender (software), Pikupljeno: 2021. Dostupno: https://en.wikipedia.org/wiki/Blender_(software).

[4] »Blender Manual,« History, Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/latest/getting_started/about/history.html.

[5] »Wikipedia,« C programski jezik, Prikupljeno: 2021. Dostupno: https://sh.wikipedia.org/wiki/C_(programski_jezik).

[6] »Blender Manual_User_Interface,« Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/latest/interface/window_system/splash.html.

[7] »Blender Manual_Object_Mode,« Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/latest/editors/3dview/modes.html.

[8] »Wikibooks Blender3D_Edit Mode,« Wikimedia, Prikupljeno: 2021. Dostupno: https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Mesh_Edit_Mode.

[9] »Blender Manual_Sculpting,« Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/latest/sculpt_paint/sculpting/introduction.html.

[10] »Blender Manual_Texture_Paint,« Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/latest/sculpt_paint/texture_paint/introduction.html.

[11] »Blender Manual_Weight_Paint,« Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/latest/sculpt_paint/weight_paint/index.html.

[12] »Blender Manual_Unwrapping,« Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/2.79/editors/uv_image/uv/editing/unwrapping/index.html.

[13] »Blender_Manual_Shader_Editor,« Prikupljeno 2021. Dostupno: https://docs.blender.org/manual/en/latest/editors/shader_editor.html.

[14] »Blender Manual_Rendering,« Prikupljeno: 2021. Dostupno: https://docs.blender.org/manual/en/latest/render/index.html.

[15] »Blender_Animation&Rigging,« Prikupljeno: 2021. Dostupno: https://www.blender.org/features/animation/.

[16] »Wikipedia Unity_game_engine,« Prikupljeno: 2021. Dostupno: https://en.wikipedia.org/wiki/Unity_(game_engine).

[17] »Wikipedia_C_Sharp,« Prikupljeno: 2021. Dostupno: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

[18] »Unity.hr_Povijest_Unityja,« Prikupljeno: 2021. Dostupno: https://unity.hr/povijest-unityja/.

[19] »Unity_Inspector_window,« Prikupljeno: 2021. Dostupno: https://docs.unity3d.com/Manual/UsingTheInspector.html.

[20] »Unity_Project_window,« Prikupljeno: 2021. Dostupno: https://docs.unity3d.com/Manual/ProjectView.html.

[21] »Unity_Scene_view,« Prikupljeno: 2021. Dostupno: https://docs.unity3d.com/Manual/UsingTheSceneView.html.

[22] »Unity_Game_view,« Prikupljeno: 2021. Dostupno: https://docs.unity3d.com/Manual/UsingTheSceneView.html.

[23] »Unity_Hierarchy_window,« Prikupljeno: 2021. Dostupno: https://docs.unity3d.com/Manual/Hierarchy.html.

[24] »Unity_Console_window,« Prikupljeno: 2021. Dostupno: https://docs.unity3d.com/Manual/Console.html.

[25] »Visual_Studio,« Microsoft, Prikupljeno: 2021. Dostupno: https://visualstudio.microsoft.com/.