Razvoj igre unutar Unity okruženja

Gotal, Darko

Sveučilište
Sjever

Završni rad br. 551/MM/2016

Razvoj igre unutar Unity okruženja

Darko Gotal, 2910/601

Varaždin, veljača 2017. godine

Sveučilište
Sjever

Multimedija, oblikovanje i primjena

Završni rad br. 551/MM/2017

Razvoj igre unutar Unity okruženja

Student

Darko Gotal, 2910/601

Mentor

pred. Andrija Bernik, dipl. inf

Varaždin, veljača 2017. godine

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

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za multimediju, oblikovanje i primjenu

PRISTUPNIK: Darko Gotal | MATIČNI BROJ: 2910/601

DATUM: 23.09.2016. | KOLEGIJ: Video animacija

NASLOV RADA: Razvoj igre unutar Unity okruženja

NASLOV RADA NA ENGL. IZIKU: Game Development Inside Unity Environment

MENTOR: Andrija Bernik, dipl.inf | ZVANJE: Predavač

ČLANOVI POVJERENSTVA:

1. mr.sc. Vladimir Stanisavljević, v. predavač - predsjednik
2. doc.dr.sc. Dean Valdec - član
3. Andrija Bernik, pred., dipl.inf. - mentor
4. mr.sc. Dragan Matković, v. predavač - zamjenski član
5.

Zadatak završnog rada

BROJ: 511/MM/2016

OPIS:

Tema rada je izrada Tower Defence igre unutar programa Unity. Objavama programa za razvoj igara poput Unity, Unreal Enginea, GameMaker i drugih, široj je javnosti omogućeno kreativno izražavanje putem izrade vlastitih igara. Sa sve većim napretkom tih alata, tehničko znanje potrebno za razvoj samih igara je sve manje. Navedeni alati omogućuju korisnicima da sa relativnom lakoćom svoje ideje pretvaraju u zabavne i kreativne igre. Cilj izrade ovog rada je napraviti jednostavnu, ne komercijalnu igru. Razvoj igre će se realizirati unutar Unity engina dok će se strana modeliranja i animacije odvija unutar Maye. Kodiranje mehanike igre će se odraditi unutar MonoDevelopa koji je dio Unity okruženja.

U radu je potrebno:

- Definirati tok rada (eng. Workflow)
- Skicirati princip igre (pomoću papira i Photoshopa)
- Modelirati potrebne dodatke (eng. assete kroz Mayu)
- Animirati potrebne dodatke (eng. assete kroz Mayu)
- Teksturirati modele i kreirane dodatke (Maya, Photoshop)
- Implementirati mehaniku igre (Unity, MonoDevelop)
- Prikazati finalnu kompoziciju kroz dva nivoa

ZADATAK UKUČEN: 07.03.2017. | POTPIS MENTORA: Bernik

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER
KOPRIVNICA

Sažetak

U ovom radu pod nazivom Razvoj igre unutar Unity okruženja objasnit će se proces izrade igre žanra obrane tornja, od prototipiranja mehanike igre do završnih implementacija modela. U radu je objašnjen tok rada, razvoj igrivog prototipa, implementacija logike i mehanike, modeliranje tornjeva i animacija korisničkog sučelja. Početak razvoja igre počinje od smišljanja željene mehanike po kojoj će igra funkcionirati. Potrebno je modelirati potrebnu imovinu koja će biti ključna u igri, poput tornjeva. Same animacije potrebne za ovaj rad bit će odrađene unutar Unity programskog okruženja. Teksture i materijali potrebni za ovaj projekt također će biti stvoreni unutar Unity programskog okruženja pomoću Unityjevih ugrađenih materijala. Prototipiranje mehanike i funkcionalnosti te stvaranje programskog koga stvoreno je unutar Microsoft Visual Studia jezikom C#.

Glavna ideja završnog rada je stvoriti funkcionalnu igru jednostavne mehanike koja ima potencijal u budućnosti biti nadograđena lakim implementacijama novih nivoa, modela i raznolikosti neprijatelja. Raznolikost neprijatelja najjednostavnije je moguće postići postavljanjem drukčijih boja tekstura samih modela. Razvojem prototipa definiraju se stvari poput umjetne inteligencije neprijatelja koja je zadužena za kretanje neprijatelja od točke A do točke B definiranim putem, stvaranje mehanike kojom igrač dobiva mogućnost uništiti neprijatelja i za to biti nagrađen, mogućnost navigacije po mapi i uvjeti koji se postavljaju kako bi igrač mogao pobijediti trenutni nivo i prijeći na sljedeći ili izgubiti i ponoviti trenutni nivo. U završnim fazama razvoja igre implementirat će se imovina dostupna na Unity Asset Store i animirati korisničko sučelje kako igra ne bi bila statična, monotona i dosadna.

Ključne riječi: Maya, mehanika, prototip, Unity

Popis korištenih kratica

3D Trodimenzionalno
AI Umjetna inteligencija (eng. Artificial Intelligence)
HP Količina života (eng. hit points)
TD Obrana tornja (eng. tower defense)

Sadržaj

1. Uvod

Razvojem igrače industrije otvaraju se vrata umjetnicima, programerima i dizajnerima za stvaranje vlastitog sadržaja zabavnog i raznolikog karaktera. Razvoj programskih sučelja za stvaranje zabavnog sadržaja doveo je igraču industriju na nove granice. Programi koji su lako pristupni svima, daju korisniku mogućnost implementacije svoje ideje u realnost. Za otvoreno tržište ili osobnu zabavu i učenje, ta programska okruženja korisniku daju veliku slobodu pri stvaranju svog okoliša, mehanike i dizajna. U ovom radu prikazan je princip korištenja više alata kako bi se stvorila zabavna aplikacija s relativnom lakoćom, te koliko je lagano izmijeniti vizualne stvari poput modela i okoliša, pri tome stvarajući nove slojeve aplikacije, šireći ju.

Unity, kojeg posjeduje Unity Technologies, popularan je izbor pri odabiru programskog okružja za stvaranje igre. Konkurencija mu je Unreal Engine kojim je stvorena igra Crysis. Razlog popularnosti Unityja je mogućnost stvaranja sadržaja za više platformi s minimalnim doradama, u slučaju da su dorade potrebne. Ovaj završni rad je odrađen u Unity programskom okružju jer je relativno jednostavan za koristiti i praktičan sa svojim povuci i spusti (eng. drag&drop) principom rada.

Ključne stvari koje su potrebne u ovom završnom radu su, kao u svakoj igri žanra obrane tornja, neprijatelji koji imaju raznoliku brzinu i količinu života, koji prolaze put i na kraju puta oduzimaju život igraču, tornjevi koji gađaju i nanose štetu neprijateljima, polja koja omogućuju gradnju tornjeva, korisničko sučelje (eng. user interface) koje govori igraču podatke o trenutnoj količini novaca kojim raspolaže, količinom preostalog života i vremenskom prikazu stvaranja sljedećeg vala neprijatelja, početna scena igre kao i scene nivoa, mogućnost pauziranja, resetiranja i nastavka igre te mogučnost navigacije mape nivoa.

Sva mehanika igre programirana je programskim jezikom C# koji je objektivno orijentirani jezik. Alternativni jezik bi bio JavaScript kojeg Unity podržava, ali je C# odabran zato što ga više platformi podržava. Unutar Microsoft Visual Studio 2015 korisniku je pružena opcija automatskog dopunjavanja koda i sintakse. Odlično surađuje s Unity programskim sučeljem te već ima predodređene funkcije koje su nužne za mehaniku, poput *void Update(), void*

OnTriggerEnter() i sličnih. U završnom radu objašnjen je proces ulaska u pojedine komponente unutar skripte pomoću skripti. Objašnjeno je po kojem principu skripte funkcioniraju i pokazuje se snaga objektivno orijentiranog programiranja. Govorit će se o sudaračima (eng. colliders) i krutim tijelima (eng. riggid body) koji su nužni za izradu neprijatelja te o samoj fizici koja je implementirana unutar Unity programskog okružja.

U ovom završnom radu objasnit će se proces izrade imovine potrebne za igru. Objasnit će se proces modeliranja tornjeva pomoću Autodesk Maya programskog okružja, od prazne scene s početnom kockom do završnih modela sva tri tornja te po kojem principu Maya funkcionira i kako se manipuliraju određeni elementi unutar scene. Govorit će se o nekim od alata kona Maya nudi poput alata stvaranja zaobljenja i alata izvlačenja površina.

Govorit će se o animaciji, načelima animacije, zašto su tu načela bitna i zašto ih treba shvatiti te kako usvojeno znanje implementirati u igru. Kroz dvanaest načela naučit će se kako se stvari gibaju po principima koje nam je ostavio Walt Disney, kako stvari učiniti vizualno privlačnijim i ugodnijim. Govorit će se o načinu animiranja objekata unutar Unity programskog okružja, parametrima i komponentama koje je moguće animirati kroz definirano vrijeme te kako sve te animacije implementirati unutar igre.

U završnom djelu objašnjen je proces implementacije imovine dostupne na Unity Asset Store, zašto je Asset Store cijenjen u svijetu igrače industrije te koje su koristi trgovine imovina. Objašnjen je proces dorade imovine u smislu spajanja sa skriptama i svim komponentama potrebnih kako bi igra funkcionirala. Cilj ovog završnog rada je čitatelja provesti kroz cijeli postupak stvaranja igre, od praznog kanvasa do igrivog prototipa kojega je moguće nadograđivati, usavršavati i širiti.

2. Povijest TD-a

Igre žanra obrana tornja (eng. tower defense, skraćeno TD) prvi početak bilježe u ranim 2000-im kao prilagođene mape za strategija koje su igrane u stvarnom vremenu (eng. real-time), kao što su StarCraft i Warcraft III. Temeljna premisa TD igre je zaštititi zgradu ili resurs od valova neprijatelja koji s vremenom postaju sve snažniji. Igrač brani bazu izgradnjom raznih tornjeva duž putanje neprijatelja, svaki sa napadačkim prednostima i manama. Svaki val neprijatelja obično imaju različite vrste oklopa, slabih samo na određenu vrstu napada, što stvara potrebu za dobro strateški postavljenim tornjevima obrane. Neprijatelji općenito slijede predefinirani put, ali neke TD igre prisiljavaju igrača da gradnjom tornjeva usporava putanju neprijatelja. Jedna od prvih računalnih igara stila obrane tornja je *Master of Defence* koju je izdala indie razvijača firma Voodoo Dimention u 2005. godini. Unatoč popularnosti žanra, od 2006. godine nitko se još nije potrudio razviti verzije za sveprisutne Flash platforme. [12]

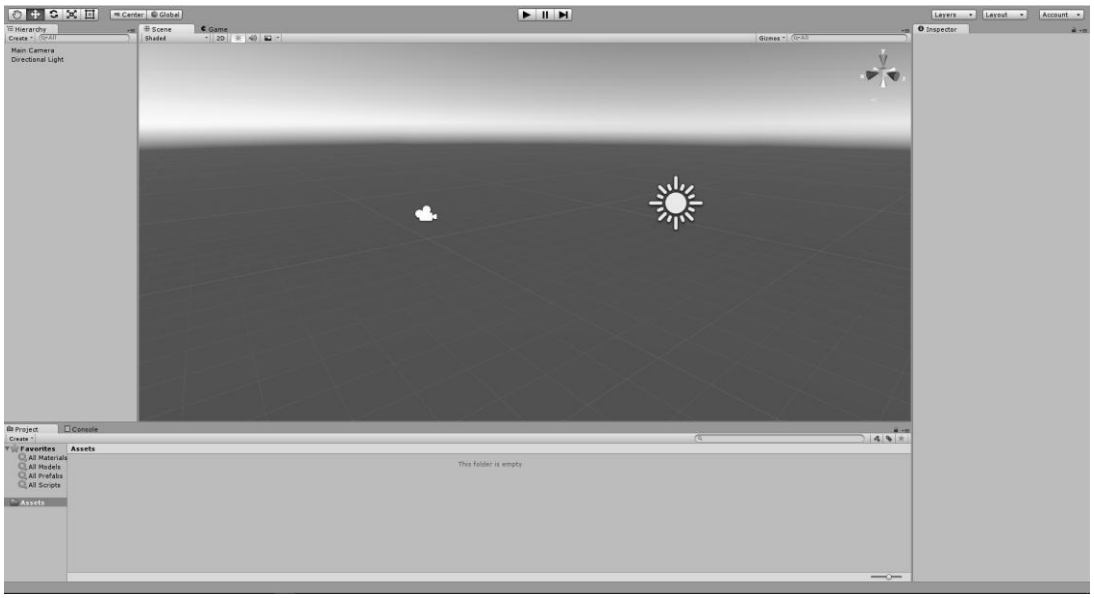
David Scott, inspiriran Warcraft III Element TD, krenuo u stvaranje Flash TD igra, uglavnom kako bi usavršio svoje vještine razvijanja. U siječnju 2007. godine Flash Element Tower Defense je puštena. Igra je bila hit i prisutna na više od 12 000 igračkih portala. Poznanik Scotta, Paul Preece, inspiriran Scottovim uspjehom, kreče u razvijanje sljedećeg Flash TD hita Desktop Tower Defense. Preece je htio stvoriti TD igru u kojoj je igrač morao gradnjom tornjeva usporavati kretnju neprijatelja. Igra je izdana u ožujku 2007. godine. Desktop Tower Defense 2008. godine osvaja Gleemax nagradu za strateški gameplay, *The Glammy* na *Independent Games* festivalu. Tijekom 2008. i 2009. godine, žanr TD igara se počinje pojavljivati na brojnim platformama. Za ručne igrače uređaje objavljuju se igre poput Ninja Town za Nitendo DS, PixelJunk Monster za PSP i AppleApp Store se puni sa TD nazivima. Savage Moon izlazi za PS3 konzolu. Defense Grid: The Awakening, hvaljena TD igra izlazi na PC i Xbox 360 konzoli. [12]

3. Alati i programska rješenja

U izradi završnog rada koriste se tri alata: Unity 5.5 (Personal Edition), Microsoft Visual Studio 2015, Autodesk Maya LT.

3.1. Unity 5.5 (Personal Edition)

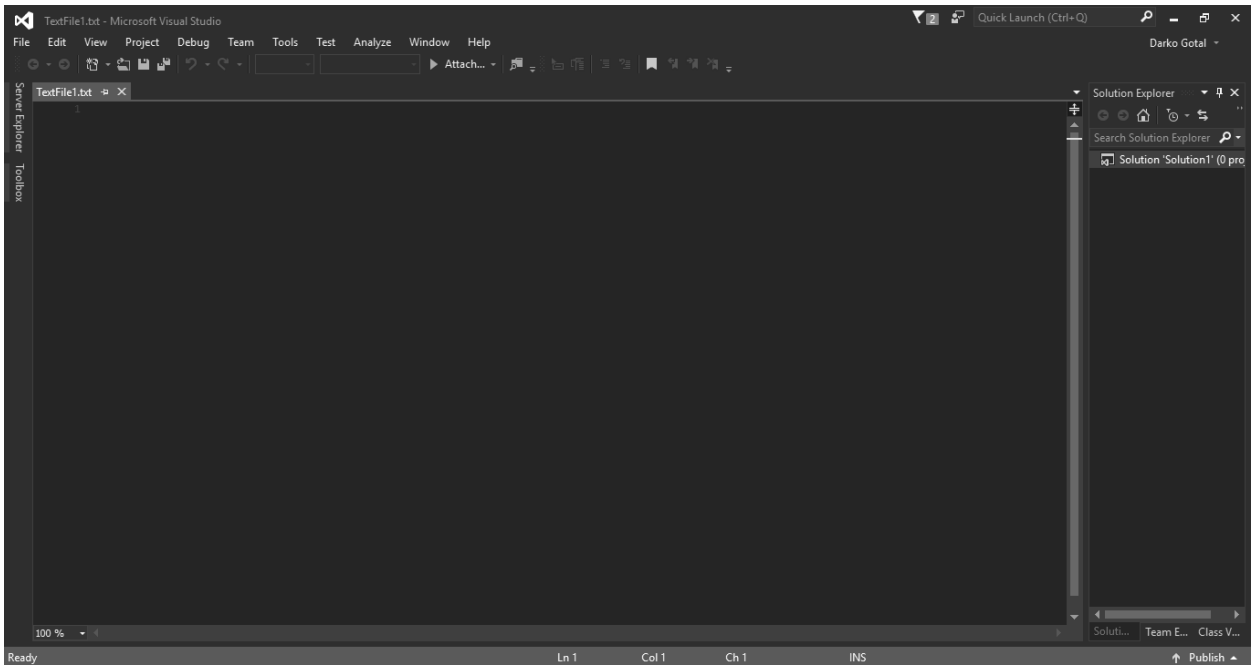
Odabran je Unity u izradi završnog rada iz razloga što se tokom godina pokazao kao pouzdana platforma za izradu računalnih igara (eng. game engine), aplikacija i drugih vrsta kreativnih izražavanja. Unity posjeduje Unity Technologies i filozofija poslovanja im je pružiti korisnicima stabilno programsko rješenje s kojim korisnik može lako razvijat željeni proizvod. Besplatan je za osobno korištenje u slučaju da se s njim zarađuje manje od 500.000 $ godišnje. Pro, isto kao i Enterprise verzija se naplaćuju i imaju neka dodatna svojstva (opcija mekane sjene i sl.). Razvijen je 2005. godine i cijenjen je zbog svoje implementirane fizike i drugih svojstva. Uz svoju pouzdanost, veliki plus Unity programskog okružja je velika baza podataka i korisnika koji se služe Unityjem. Zbog velikog broja foruma, pitanja i odgovora uvijek se može doći do rješenja problema na koji razvijač aplikacije često nailazi. [1]



Slika 3.1. Unity korisničko sučelje

3.2. Microsoft Visual Studio 2015

Microsoft Visual Studio je integrirano razvojno okruženje (integrated development environment - IDE) od Microsofta. Koristi se za razvoj računalnih programa za Microsoft Windows, kao i web stranica, web aplikacija i web usluge. Visual Studio koristi za razvoj Microsoft softverske platforme kao što su Windows API, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight. Odlično surađuje sa Unityjem kod razvoja C# vrste koda dopuštajući korisniku automatsku dopuno koda, funkcija, varijabli i uvjeta. Prilikom automatske dopune stvara se spuštajući izbornik koji nudi opcije automatske dopune sintakse. Alat je odličan za stvaranje objektivno orijentiranog koga. Vizualno je privlačan i ima svoje korisne kratice (automatsko popunjavanje "for" petlje, i sl.). Alternativa je MonoDevelop koji dolazi u sklopu Unityja. [2]

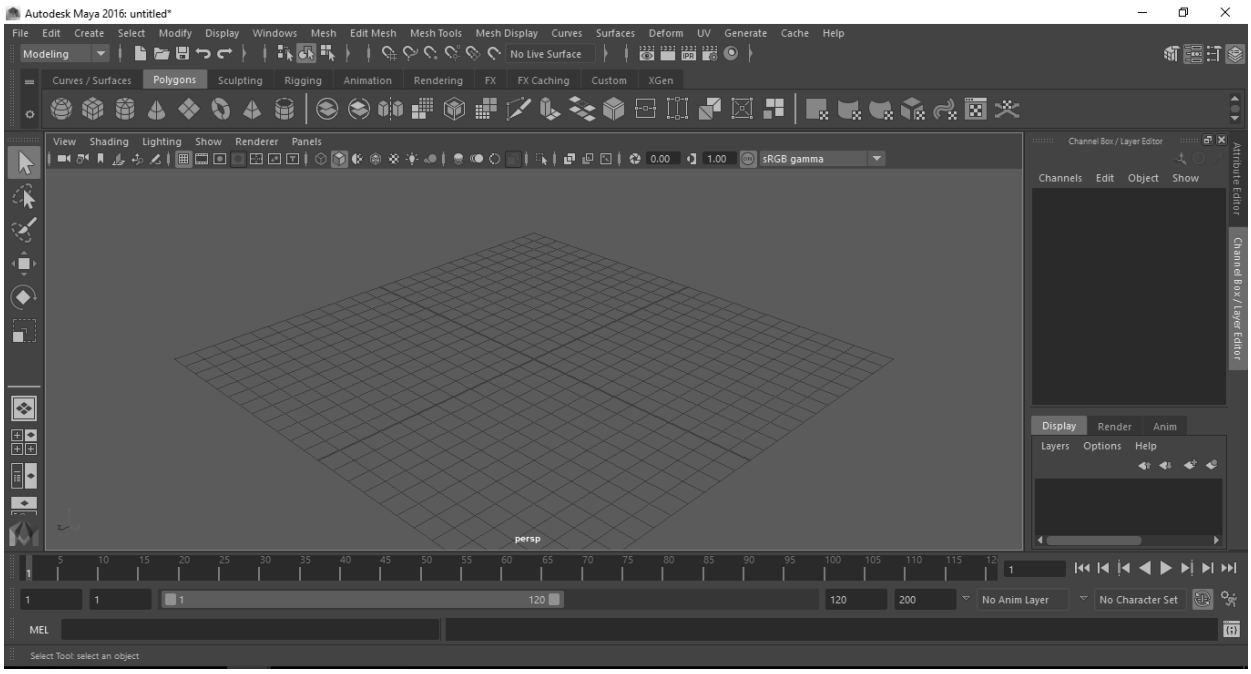


Slika 3.2. Visual Studio korisničko sučelje

3.3. Autodesk Maya LT

Maya, jedan od Autodeskovih proizvoda, je alat za razvijanje 3D modela, kostura (eng. rig), animacija, ukratko svega što bi trebalo developeru za razvoj imovine igre (eng. asset). Izvorno je razvijen od strane Alias Systems Corporation (bivši Alias i Wavefront), a trenutno se nalazi u vlasništvu i razvijen je od strane Autodeska. Maya se koristi za razvoj svih potrebnih imovina (eng. asseta) zbog lakoće modeliranja, teksturiranja i animacije. [3]

Prilikom modeliranja korisnik ima opciju navigacije u trodimenzionalnom prostoru gledajući model i njegove komponente sa svih strana. Osim standardne perspektive, jednim brzim pritiskom tiple razmaka (eng. space) ekran se dijeli na četiri polja, tri koja pokazuju izometričku perspektivu i jedan prozor s klasičnom perspektivom. Selekcija komponente korisnika ostaje u sva četiri prozora te se mijenja u stvarnom vremenu, omogućavajući korisniku lakše praćenje i modeliranje.



Slika 3.3. Maya korisničko sučelje

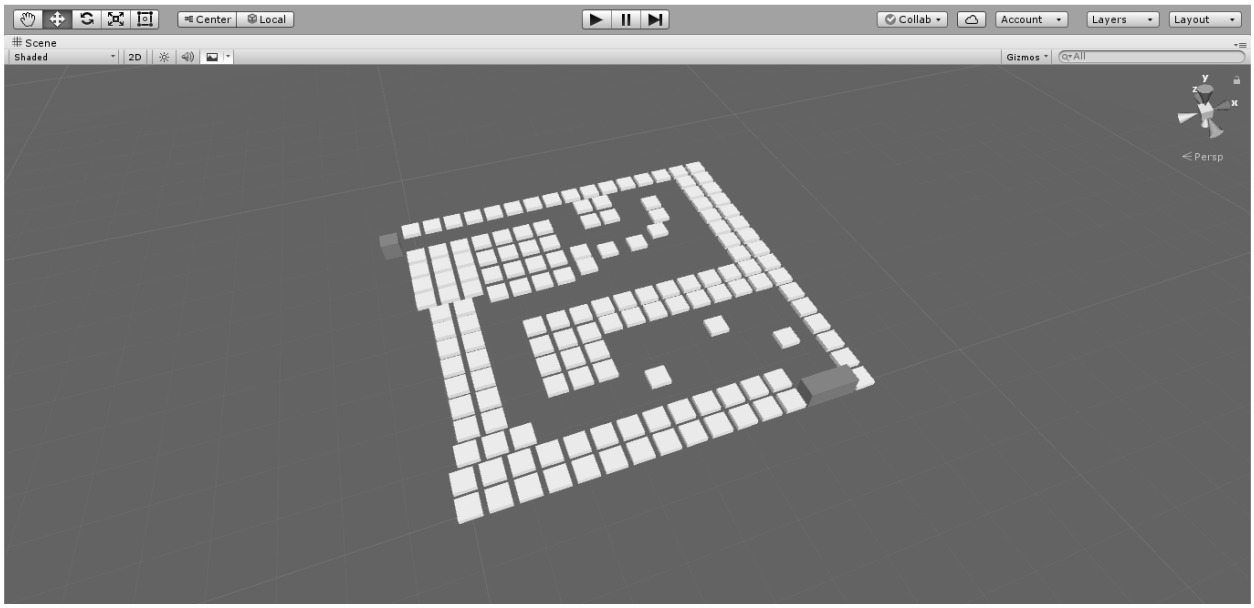
4. Razvoj mehanike

Početak izrade igre je jedan od najvažnijih koraka, jer kao razvijač (eng. developer) potrebno je dobro razmisliti kako se želi što napraviti, kako bi izbjegli probleme i komplikacije u kasnijim fazama razvoja. Fazu prototipiranja je odlično napraviti u startu, jer po principu na kojem Unity radi, može se odraditi kompletnu mehaniku igre bez zamaranja sa grafikom, to jest vizualnim dijelom igre, jer se na kraju sve pametne kopije (eng. prefab) mogu zamijenit gotovim modelima i grafikama. [4]

4.1. Dizajn nivoa

Kreče se smišljanjem kako izgraditi nivo (eng. level). Nakon nekog vremena odlučen je sljedeći tok rada (eng. workflow): izraditi jednu jedinicu polja (ono na čemu će se gradit toranj), pretvoriti to polje u pametnu kopiju (eng. prefab) koji će imati potrebne skripte na sebi, te to polje dupliciratи po potrebi veličine mape. Velika prednost pamenih kopija unutar Unityja je mogućnost mijenjanja jednog komponenta na svim objektima odjednom, za razliku od selektiranja svakog objekta zasebno i svakom objektu zasebno mijenjati jednu komponentu. Polja se dupliciraju na način da se polja pomiču za određeni ofset. S lakoćom se stvara 20x20 mapa i zatim se bez problema mogu obrisati sva polja koja nisu potrebna, mjesta gdje će ići određena imovina (eng. asset), mjesta gdje će se neprijatelji kretati i slično.

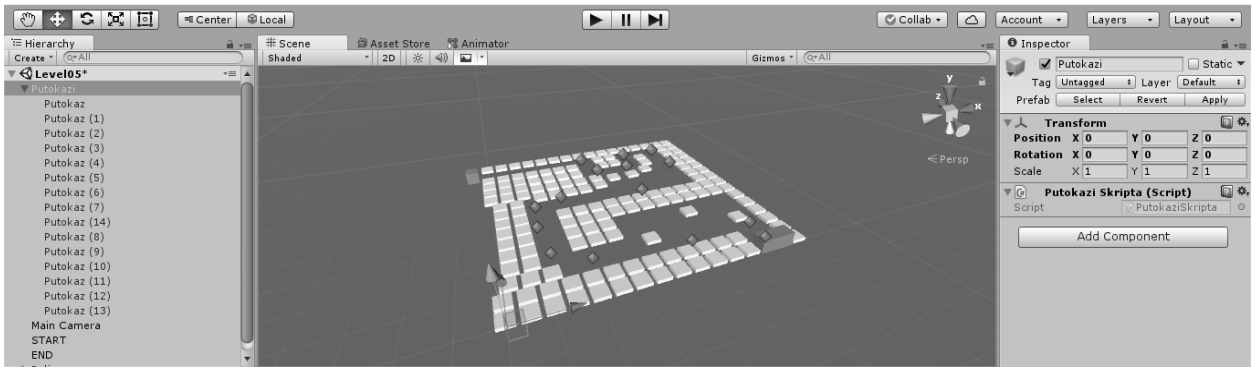
Nakon osmišljavanja polja rade se dvije kocke koje će simbolizirati početak i kraj puta koji neprijatelji prolaze. Za "Start" i "End" koriste se dvije obične kocke unutar Unityja koje se kasnije po želji mogu promijeniti.



Slika 4.1. Polja, početak i kraj zamišljene putanje

Stvara se prazan objekt naziva "Putokazi" koji služi kao grupa i unutar tog objekta objekte "Putokaz" koje će neprijatelji pratiti. Ta tehnika razvoja neprijateljskog ponašanja je često korištena metoda kretnje neprijatelja od određene točke do određene točke, koristi se u većini jednostavnih igara.

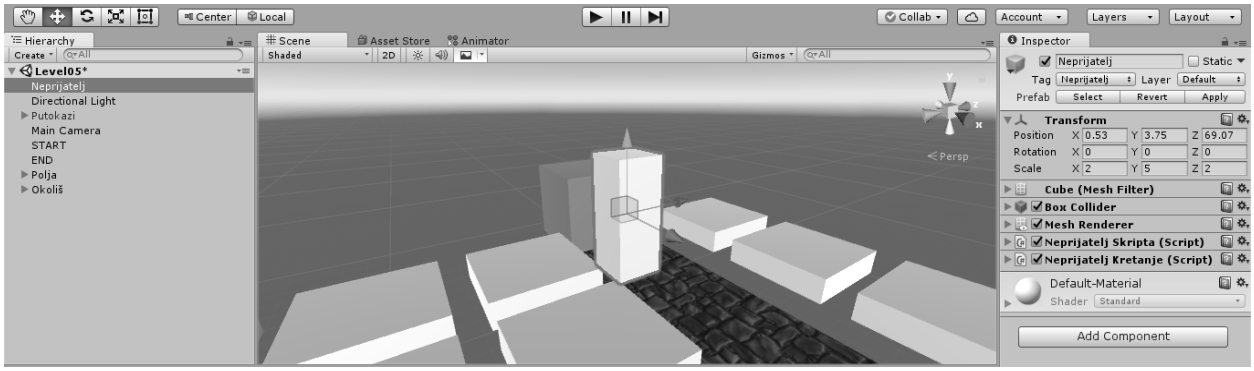
Na objektu "Putokazi" stvara se C# skriptu *PutokaziSkripta* koja na početku nivoa stvara listu (eng. array) svoje djece (objekti koji su unutar tog objekta) i tu listu sprema pod javno deklariranu varijablu tipa "Transform" naziva "tocke". Razlog tome je što će neprijatelji pomoću svoje skripte hvatati pozicije točaka i kretati se prema njima.



Slika 4.2. Putokaz

4.2. Inteligencija neprijatelja

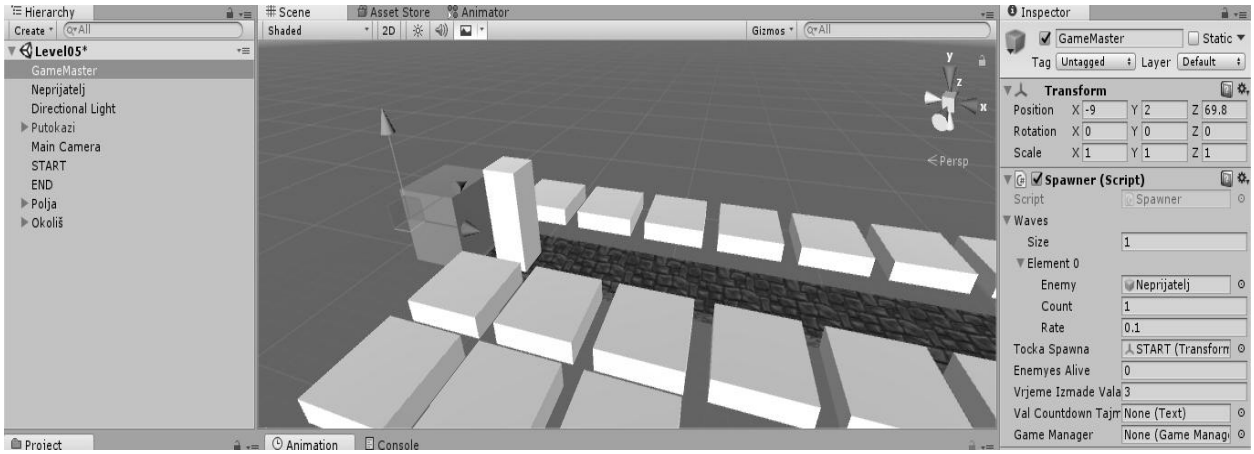
Poslije plana gradnje nivoa i polja, kreče se na prototipiranje neprijatelja. Stvara se C# skriptu koja će morati imati neke komponente kako bi se neprijatelj mogao kretati, rotirati, prikazivati HP (eng. hit points). [5] Stvara se jednostavna izdužena kocka koja će simbolizirati neprijatelja, dodaje se C# skripta *NeprijateljSkripta* koja se kasnije razdvaja na dvije skripte, *NeprijateljSkripta* i *NeprijateljKretanje* kako bi se olakšala mehanika igre. Jedna skripta je zadužena striktno za kretanje neprijatelja dok druga govori koliko neprijatelj ima HP-a, daje neprijatelju mogućnost umiranja, definira koja mu je početna brzina i sama vizualizacija štete koju prima na HP. Rotacija je definirana *Quaternion* varijablom dok se samo kretanje neprijatelja postiže množenjem *Translate* varijable s nekakvom *float* vrijednosti kroz neko vrijeme.



Slika 4.3. Prvi neprijatelj

4.3. Mehanika stvaranja

Poslije izrade neprijatelja počinje se smišljati nekakva mehanika stvaranja (eng. spawn) koja će regulirati koliko je neprijatelja potrebno stvoriti, kojom brzinom i kojim razmakom između valova. Stvara se C# skripta *Spawner* te se stvara prazan objekt unutar scene zvan "GameMaster" na koji se dodajem *Spawner* skripta. Objekt "GameMaster" će biti taj koji će paziti na stvaranje neprijatelja, uvjet pobjede ili gubitka nivoa, prikazivanje menija za pauzu, informacije o igraču (novci, život).



Slika 4.4. Game Master objekt sa Spawner skriptom

Spawner se sastoji od komponenti koje opisuju odakle se neprijatelji stvaraju, koliko je vremena ostalo do stvaranja sljedećeg vala, postavljanje uvjeta pobjede ili gubitka (eng. win/lose) i sam vizualni prikaz vremena između valova. Skripta *Spawner* koristi ugrađenu funkciju unutar Unityja zvanu *Instantiate*. [6]

4.4. Prototipiranje tornja

Nakon prototipiranja stvaranja neprijatelja, sljedeći korak je stvoriti toranj koji će ciljati i pucati na neprijatelje. Unutar Maye izrađuje se jednostavan model tornja. Kreće se od kocke s kojom se igra dok se ne dobije željen izgled. Cijev početnog tornja stvara se od jednostavnog cilindra. Model se ubacuje u Unity te započinje prototipiranjem mehanike pucanja. Stvara se C# skriptu *Turret* i zadaje se na toranj naziva "Canon-T". Kako bi toranj funkcionirao (ciljao metu i pucao) mora imati par komponenti, to su: varijabla TowerHead tipa Transform koja će biti zadužena za rotaciju glave tornja prema neprijatelju, varijabla metaNeprijatelj tipa NeprijateljSkripta koja će očitavati objekte koji imaju skriptu neprijatelja na sebi, varijabla metaTornja tipa Transform koja će slati povratnu informaciju o trenutnoj poziciji objekta koji ima komponentu NeprijateljSkripta na sebi, varijabla neprijateljTag tipa string koji hvata objekte koji su označeni kao "Neprijatelj" a u suprotnom ignorirat objekt, varijablu domet tipa float koja određuje domet tornja, varijablu brzinaOkreta tipa float koja određuje kojom se brzinom glava tornja okreče prema meti, varijablu brzinaPucanja tipa float koja određuje kojom brzinom toranj puca, varijablu odbrojavanjePucanja tipa float koja određuje vremenski razmak između svakog otpucaja, varijablu metakPrefab tipa GameObject kojom određujem koje metke toranj koristi i varijablu tockaPucanja tipa Transform koja govori iz koje pozicije će se metak tornja stvarati.

Unutar *Turret* skripte koja je postavljena na tornju stvara se *void UpdateMete()* koji će se brinuti o postavljanju mete tornja. Ta se funkcija pomoću *Start()* metode zove svakih 0.5 sekundi pomoću *InvokeRepeating* funkcije. Unutar *UpdateMete()* funkcije stvara se lista *GameObject*-a zvana *"neprijatelj"* i toj se listi govori da spremi sve objekte koji na sebi imaju *"neprijateljTag"* oznaku (eng. tag). Poslije toga se stvara *float* varijabla *najkracaDistanca* i zadaje se vrijednost *Mathf.Infinity* što je u prijevodu beskonačnost. Zatim se zadaje nova *GameObject* varijabla zvana *"najbliziNeprijatelj"* i zadaje joj se vrijednost *null* (prazna vrijednost). Ispod se stvara *foreach* petlja koja sadrži *float* varijablu *"udaljenostOdNeprijatelja"* koja računa udaljenost neprijatelja od tornja i unutar petlje *if* uvjet koji govori ako je *udaljenostOdNeprijatelja* manja od *najkracaDistanca* tada je *najkracaDistanca* jednaka udaljenosti od neprijatelja. Ispod *foreach* petlje zadaje se *if* uvjet koji govori da ako *najbliziNeprijatenj* nije jednak *null*, to jest ako postoji i ako je *najkracaDistanca* manja ili jednaka dometu tada je *metaTornja* jednaka *najbliziNeprijatelj.transform* komponenti, to jest dohvaća koordinate neprijatelja unutar radijusa te je *metaNeprijatelj* jednaka *najbliziNeprijatenj.GetComponent<NeprijateljSkripta>(),* to jest

dohvaća komponentu *NeprijateljSkripta* sa objekta. Taj dio će kasnije služiti za implementaciju mehanike kojom se oštećuje neprijatelje. Ispod *if* uvjeta je *else* uvjet koji govori da je *metaTornja null* ako mu ništa nije u dometu.

Ispod *UpdateMete()* funkcije stvara se *LockOnTarget()* funkcija koje će biti zadužen za rotaciju glave tornja prema meti tornja. Unity radi sa vektorima matematičkim putem. Iz tog se razloga stvara varijabla smjer tipa Vector3 i oduzimam poziciju mete tornja od pozicije objekta na kojem se nalazi skripta *Turret*. Stvara se varijabla tipa *Quaternion* naziva *smjerMete* koja se izjednačava sa *Quaternion.LookRotation* (smjer). *LookRotation* je ugrađena funkcija unutar Unity programskog okružja koja dopušta korisniku da unese *Vector3* vrijednost i Unity računa koji je vektor usmjeren prema gore a koji je vektor usmjeren prema meti (naprijed). Zatim se piše sljedeća linija koda:

```
Vector3 rotacijaTornja = Quaternion.Lerp(TowerHead.rotation, smjerMete, Time.deltaTime * brzinaOkreta).eulerAngles;
```

Quaternion.Lerp je također funkcija unutar Unityja koja dopušta laganu tranziciju između vrijednosti A prema vrijednosti B nekim vremenom t. u ovom slučaju vrijednost A je rotacija glave tornja, vrijednost B je smjer mete a vrijeme se određuje brzinom okreta koja je definirana na početku float varijablom i množim se vrijednosti Time.deltaTime.

Time.deltaTime je vrijeme u sekundama koje je bilo potrebno za očitavanje prijašnjeg okvira (eng. frame). [7] Dio koda na kraju (.eulerAngles) pretvara Quaternion vrijednost u Euler kutove. Nakon te linije piše se: TowerHead.rotation = Quaternion.Euler(0,rotacijaTornja.y,0); što govori da je rotacija TowerHead objekta koja je definirana u startu koda jednaka rotacijaTornja vrijednosti u liniji koda iznad. Dio u zagradi govori da rotira objekt samo po Y osi te da su X i Z osi 0. Poslje LockOnTarget() funkcije stvara se funkcija:

```
OnDrawGizmosSelected() {
Gizmos.color = Color.red;
 Gizmos.DrawWireSphere(transform.position, domet); }
```

Ta funkcija govori da pri selekciji objekta Unity crta sferu oko objekta na poziciji objekta i radijusa domet.

Zatim se stvara funkcija *Pucaj()* koja je zadužen za stvaranje metka koji putuje prema neprijatelju. Stvara metak funkcijom *Instantiate*, također funkcija unutar Unityja koja uzima varijablu tipa *GameObject* koji će stvorit, startnu poziciju i rotaciju objekta.

```
   void Pucaj() {
    GameObject    metakGO    =    (GameObject)Instantiate(metakPrefab,
tockaPucanja.position, tockaPucanja.rotation);
    MetakSkripta metakSkripta = metakGO.GetComponent<MetakSkripta>();
    if (metakSkripta != null) {
    metakSkripta.Hvataljka(metaTornja); } }
```

Funkcija stvara novi *GameObject metakGO* koji stvara pametnu kopiju (eng. prefab) metka i stvara taj metak na lokaciji točke pucanja sa rotacijom objekta točke pucanja. Zatim stvara varijablu *metakSkripta* tipa *MetakSkripta* i dohvaća komponentu *MetakSkripta* sa metka koji se stvorio. Zadaje se uvjet ako postoji skripta *MetakSkripta* na stvorenom metku poziva se funkcija *Hvataljka* sa skripte *MetakSkripta*.

Zatim se sve ove stvari moraju pozvati unutar *Update()* metode koja je dio Unityja kako bi skripta funkcionirala. *Update* metoda čita kod unutar sebe svaki okvir (eng. frame). [8]

```
   void Update () {
    if(metaTornja == null) {
    return; }
    LockOnTarget();
    if (odbrojavanjePucanja <= 0f) {
    Pucaj();
    odbrojavanjePucanja = 1f / brzinaPucanja; }
    odbrojavanjePucanja -= Time.deltaTime;}
```

Update metoda Turret skripte govori u startu da ako je metaTornja prazna, to jest ako toranj nema metu da ne čita kod dalje. U slučaju da toranj ima metu zove funkciju LockOnTarget. Poslije toga je postavljen uvjet koji govori da ako je odbrojavanjePucanja (float varijabla zadana na početku koda) manja ili jednaka nuli pozove funkciju Pucaj(). Odbrojavanje vremena pucanja se računa oduzimanjem vrijednosti odbrojavanjePucanja od Time.deltaTime.

4.5. Prototipiranje metka tornja

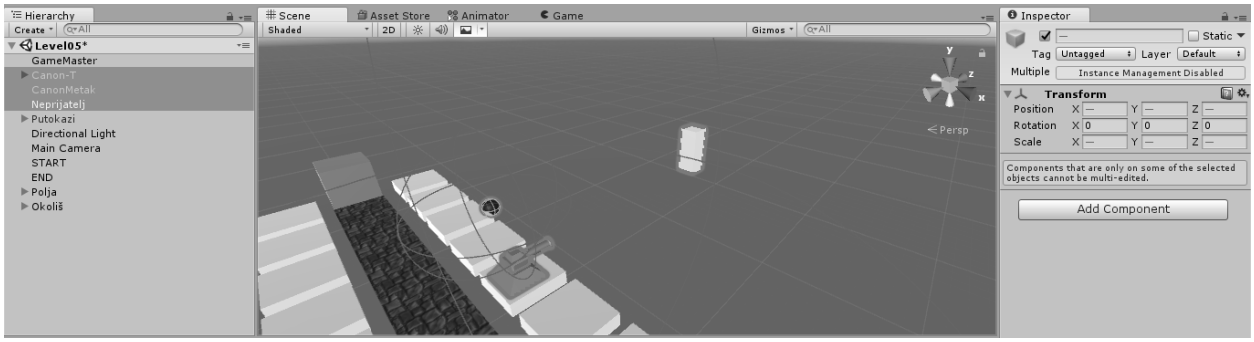
Potrebno je stvoriti C# skriptu naziva *MetakSkripta* koja će biti zakačena na sam metak koji toranj stvara. Kako bi postojala mogućnost raznolikosti tornjeva stvara se mehanika gdje sam metak ima definirane vrijednosti jačine udarca, brzine kretanja prema meti i slično.

```
   public class MetakSkripta:MonoBehaviour{
    private Transform meta;
    public float brzinaMetka = 70f;
    public float radiusMetka = 0f;
    public float damage = 0.3f;
    public GameObject impaktEfekt;

    public void Hvataljka(Transform _meta) {
    meta = _meta; }
   void Update () {
    If (meta == null) {
    Destroy(gameObject);
    return; }
    Vector3 smjer = meta.position - transform.position;
    float duzina = brzinaMetka * Time.deltaTime;
   if (smjer.magnitude <= duzina) {
    MetaPogodena();
    return; }
    transform.Translate(smjer.normalized * duzina, Space.World);
    transform.LookAt(meta);}
    void MetaPogodena() {
   GameObject instancaEfekta = (GameObject)Instantiate(impaktEfekt,
transform.position, transform.rotation =
Quaternion.Inverse(transform.rotation));
    Destroy(instancaEfekta, 1.5f);
    if(radiusMetka > 0) {
    Explode(); }
   else {
    Damage(meta); }
    Destroy(gameObject); }
    void Explode() {
    Collider [] kolajderi = Physics.OverlapSphere(transform.position,
radiusMetka);
    foreach (Collider kolajder in kolajderi) {
    if(kolajder.tag == "Neprijatelj") {
    Damage(kolajder.transform); } } }
   void Damage(Transform enemy) {
    NeprijateljSkripta e = enemy.GetComponent<NeprijateljSkripta>();
    if( e != null) {
    e.TakeDamage(damage); } } }
```

Kod 4.1. Skripta metka tornja

MetakSkripta je napravljena s mogućnosti razlikovanja dvije vrste metka, metak koji ima radijus i metak koji nema. Metak koji ima svoj radijus će nanositi štetu neprijateljima u svom radijusu tako da prepoznaje sudarače (eng. collideres) neprijatelja i traži *NeprijateljSkripta* komponentu objekata na kojima je sudarač. Zatim ulazi u komponentu *healh* objekta na kojem se nalazi *NeprijateljSkripta* i oduzima od te vrijednosti vrijednost *damage* koja je definirana na skripti metka. Pružena je mogućnost stvaranja efekta pri impaktu metka o metu pomoću *Instantiate* metode. *Instantiate* metoda pruža korisniku mogućnost stvaranja željenog objekta na definiranu poziciju s definiranom početnom rotacijom.



Slika 4.5. Metak, top i neprijatelj unutar scene

U trenutnoj fazi prototipiranja postoji funkcionalni toranj koji ima mogućnost praćenja neprijatelja i nanošenja štete. Sljedeći korak je dati igraču neka svojstva poput novca i količine života kako bi igra imala nekakav smisao i cilj te implementirati mehaniku kojom se igraču dodaje i oduzima količina novca i života.

4.6. Karakteristike igrača

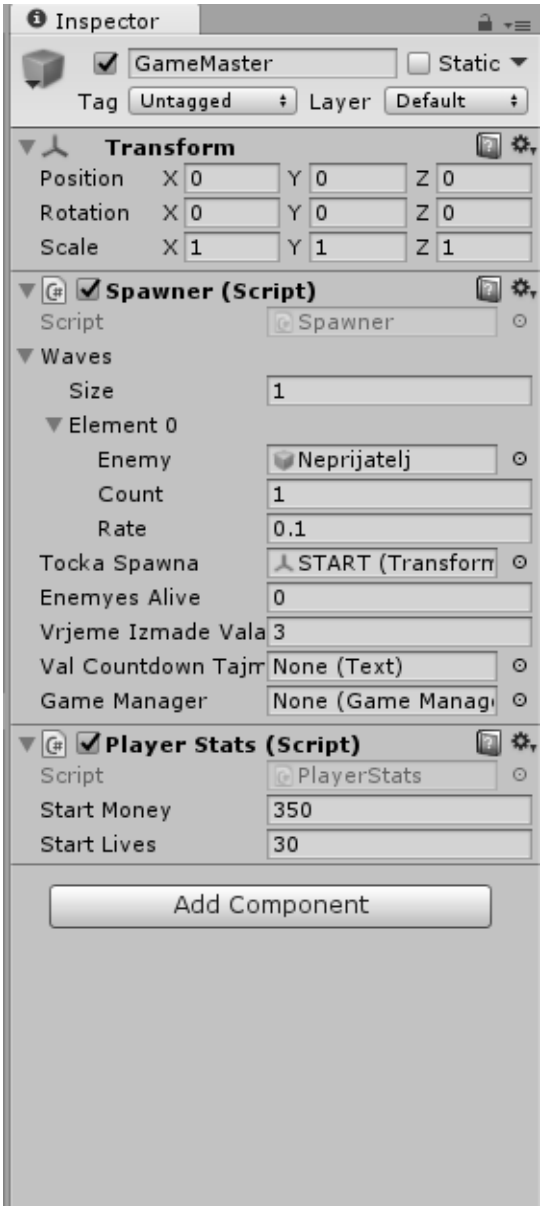
Stvara se nova C# skripta nazvana *PlayerStats* koja će u sebi sadržavati informacije o igraču, točnije količini novca kojom igrač raspolaže na početku igre, brojem preostalog života i broj rundi koje je igrač preživio.

```
public class PlayerStats:MonoBehaviour{
 public static int Money;
 public static int Lives;
 public static int rounds;
 public int startMoney = 350;
 public int startLives = 30;
 void Start() {
 Money = startMoney;
 Lives = startLives;
 rounds = 0; }}
```

Kod 4.2. Skripta karakteristike igrača

Klasa *PlayerStats* je javna (eng. public) jer ostali elementi igre moraju biti u mogućnosti ulaziti u komponente skripte i mijenjati ih, nagrađivat igrača za ubijenog neprijatelja i sl. Varijable tipa *static* su vrste varijabla koje se prenose iz nivoa u nivo, to jest, ako se ne odredi drukčije, količina novaca i života će ostati nepromijenjena prilikom učitavanja novog nivoa. [9]

Koriste se varijable tipa *int* jer su te varijable cijeli brojevi, bez decimalnih vrijednosti. U slučaju ove igre igrač nema količinu novaca i života koje se reprezentiraju brojevima s decimalnim vrijednosti. Ovakvu je mehaniku lakše pratiti i manipulirati njom.



Slika 4.6. Skripta PlayerStats dodana na GameMaster objekt

Slika 4.6. prikazuje komponentu PlayerStats koja se nalazi ispod Spawner komponente. Vrijednosti Start Money i Start Lives moguće je izmijeniti unutar urednika (eng. editor) bez potrebe ulaska u sam kod napisan unutar skripte. S ovime se pokazuje moć Unity programskog sučelja i njegovu fleksibilnost koju nudi.

4.7. Uvjeti pobjede i gubitka

Kada su postavljene karakteristike igrača otvara se mogućnost postavljanja uvjeta pobjede ili gubitka, u igraćoj industriji nazvano win/lose uvjeti. Stvara se nova C# skripta nazvana *GameManager*. Unutar skripte *NeprijateljKretnja* definirano ako je neprijatelj dođe do kraja svoje putanje da oduzme jedan život igraču. Pomoću te mehanike se može stvoriti uvjet gubitka (eng. lose condition). Ako je igrač na nula ili manje života aktivira se sučelje gubitka igre i daje se igraču opcija ponovnog pokretanja nivoa ili povratak na izbornik.

```
public class GameManager:MonoBehaviour{
 public static bool mrtavSi;
 public GameObject gameOverUI;
 public GameObject completeLevelUI;
 void Start() {
 mrtavSi = false; }
void Update (){
 if (mrtavSi)
 return;
 if (PlayerStats.Lives <= 0) {
 EndGame(); }}
 void EndGame() {
 mrtavSi = true;
 gameOverUI.SetActive(true); }
 public void WinLevel() {
 mrtavSi = true;
 completeLevelUI.SetActive(true); } }
```

Kod 4.3. Skripta menadžera igre

Također treba postojati uvjet pobjede nivoa. Stvara se nova C# skripta naziva *CompleteLevel*. U slučaju ove igre, uvjet pobjede je ubiti sve neprijatelje koji se stvore. Nakon smrti svih neprijatelja unutar igre se stvara sučelje koje prikazuje broj preživljenih rundi te daje opciju prelaska na sljedeći nivo ili povratka na glavni izbornik.

```
public class CompleteLevel:MonoBehaviour{
 public SceneFader sf;
 public string MenuSceneName = "MainMenu";
 public string nextLevel = "Level02";
 public int levelToUnlock = 2;
 public void Continue() {
 PlayerPrefs.SetInt("levelReached", levelToUnlock);
 sf.FadeTo(nextLevel); }
 public void Menu() {
 sf.FadeTo(MenuSceneName); } }
```

Kod 4.4. Skripta pobjede nivoa

Skripta *CompleteLevel* se aktivira ako igrač zadovolji uvjete potrebne za prolazak nivoa, te pokreće sljedeći nivo pomoću *SceneFader* komponente. Definirane su varijable tipa *string* koje govore koji nivo treba pokrenuti sljedeće, što znači da svaki nivo ima drukčiju *CompleteLevel* skriptu, to jest drukčiji naziv *nextLevel* varijable.

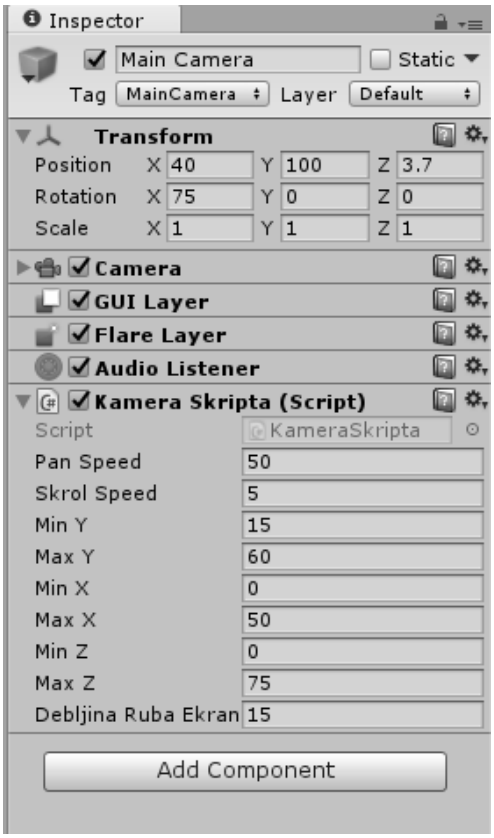
Ova skripta također govori igri koji nivo treba otključati jer je mehanika postavljena tako da svi nivoi nisu otključani od početka igre, nego se trenutni nivo treba prijeći kako bi se sljedeći otključao.

4.8. Upravljanje kamerom

Kako bi se igraču pružio veći nivo interaktivnosti potrebno je složiti nekakvu vrstu mehanike koja omogućuje igraču kontrolu nad kamerom. Stvara se C# skripta *KameraSkripta* i postavlja ju se na glavnu kameru u sceni.

```
   public class KameraSkripta:MonoBehaviour{
    public float panSpeed = 30f;
    public float skrolSpeed = 5f;
    public float minY = 15f;
    public float maxY = 80f;
    public float minX = 0f;
    public float maxX = 50f;
    public float minZ = 0f;
    public float maxZ = 75f;
    public float debljinaRubaEkrana = 15f;
   void Update () {
    if (GameManager.mrtavSi) {
    this.enabled = false;
    return; }
    /* Kretanje Tipkovnicom i Mišem */
    if (Input.GetKey("w") || Input.mousePosition.y >= Screen.height -
debljinaRubaEkrana) {
    transform.Translate(Vector3.forward * panSpeed * Time.deltaTime,
Space.World); }
    if (Input.GetKey("s") || Input.mousePosition.y <=
debljinaRubaEkrana) {
    transform.Translate(Vector3.back * panSpeed * Time.deltaTime,
Space.World); }
    if (Input.GetKey("d") || Input.mousePosition.x >= Screen.width -
debljinaRubaEkrana) {
    transform.Translate(Vector3.right * panSpeed * Time.deltaTime,
Space.World); }
    if (Input.GetKey("a") || Input.mousePosition.x <=
debljinaRubaEkrana) {
    transform.Translate(Vector3.left * panSpeed * Time.deltaTime,
Space.World); }
    /* Skrolanje */
    float skrol = Input.GetAxis("Mouse ScrollWheel");
    //Debug.Log(skrol);
    Vector3 pos = transform.position;
    pos.y -= skrol * 1000 * skrolSpeed * Time.deltaTime;
    pos.y = Mathf.Clamp(pos.y, minY, maxY);
    pos.x = Mathf.Clamp(pos.x, minX, maxX);
    pos.z = Mathf.Clamp(pos.z, minZ, maxZ);
    transform.position = pos; }}
```

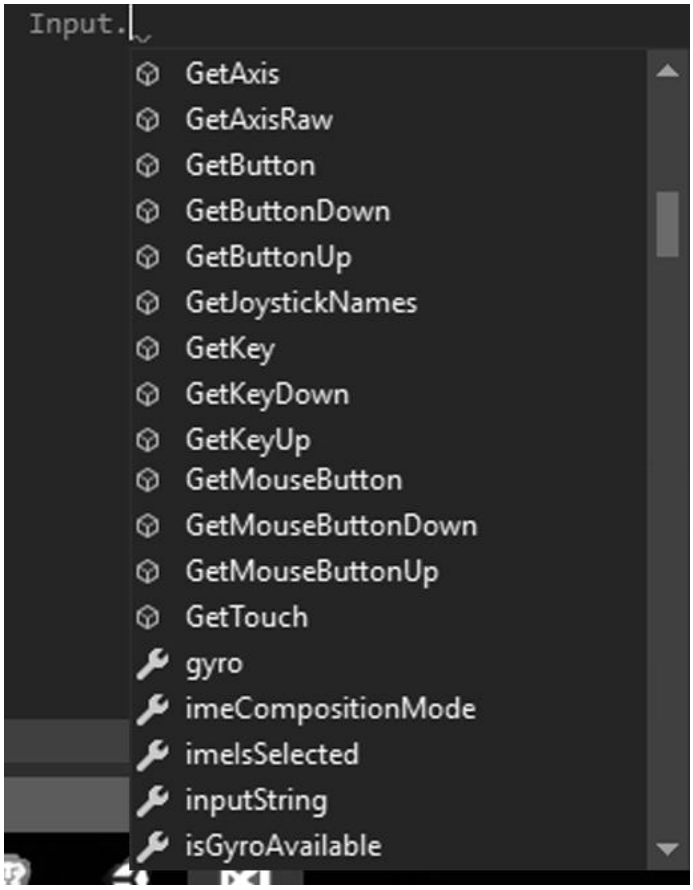
Kod 4.5. Skripta kamere



Slika 4.7. KameraSkripta na kameri scene

Unutar *KameraSkripta* skripte se postavljaju uvjeti unosa određene tipke ili određene pozicije miša. U slučaju da korisnik priđe mišem pojedinom rubu ekrana kamera će se kretati u tom smjeru glatkom tranzicijom pozicije. S obzirom da je mehanika gubitka igre postavljena, ukoliko igrač umre kontrola nad kamerom je onemogućena.

Unity dopušta veliku kontrolu nad praćenjem unosa tipke (eng. input) ili pozicije miša. [10]



Slika 4.8. Neke od Input opcija koje Unity nudi

Slika 4.8. prikazuje popis funkcija koje Unity nudi za kontrolu i praćenje unosa. Svaka od ovih funkcija ima podfunkciju kojom je moguće definirati točnu tipku unosa u smisli pisanja naziva tipke, na primjer „Space“, ali se može koristiti i ugrađenim sistemom definiranja tipke koju Unity nudi, na primjer „Jump“, što je isto kao „Space“ tipka na tipkovnici ili „X“ tipka na kontroleru namijenjenom konzolama.

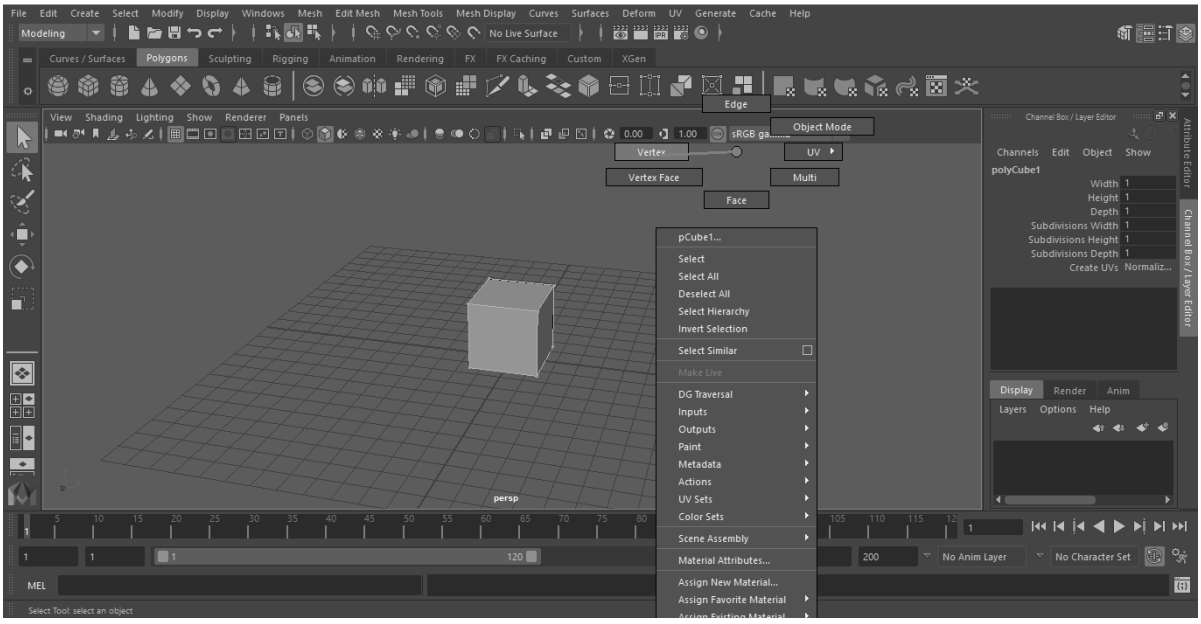
Tu je prikazana još jedna pozitivna karakteristika i količina inteligencije kojom Unity programskog okružje raspolaže.

5. Modeliranje

Bilo je dosta 3D aplikacija koje su došle i otišle tijekom godina, zatim tu su oni programi koji su, ne samo preživjele, nego dostigli novu razinu prepoznatosti u cijeloj industriji kao pouzdano i dobar poznati set alata (eng. toolset). Autodesk Maya je jedan od tih programa. U okviru ovog programskog paketa umjetnik je u mogućnosti modelirati znakove, vozila, okruženja, itd., opremiti ih i biti mobilan, udahnut im život kroz animaciju. Kao i većina 3D aplikacija, tu je prisutna krivulja učenja za navikavanje na rad u Maya ako se nikad niste upoznali sa 3D programima prije. Bazira se na modeliranju pomoću točaka u svom virtualnom trodimenzionalnom prostoru, točke računalnog modela (eng. vertex) koje se spajaju linijama od kojih nastaju rubovi a sa zatvaranjem rubova o određeni oblik nastaje lice, poligon (eng. face).

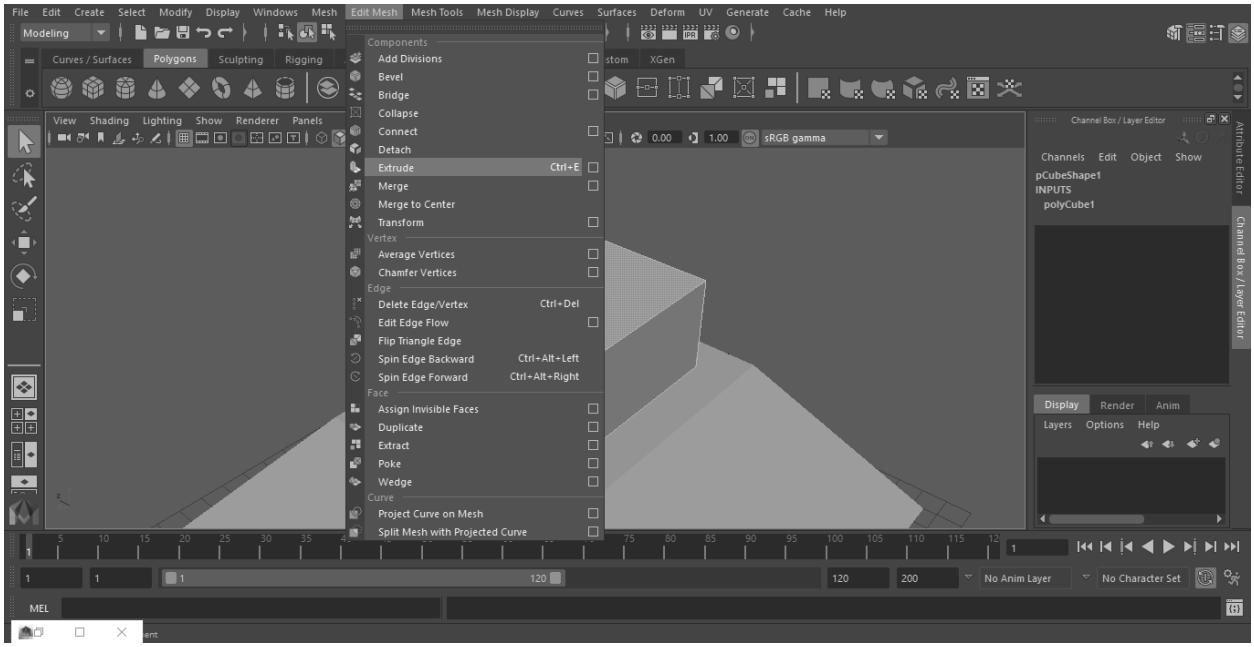
5.1. Top toranj

Modeliranje tornjeva započinje otvaranjem Autodeskovog programa Maya. Od početne scene se stvara kocka kao baza modela. Definirani su zahtjevi za tri različita modela tornja, jedan koji brzo puca male metke i podsjeća na minigun, jedan koji puca sporije i ima veće metke, asocijacija na top, te zadnji model koji bi trebao imati magičan dizajn i ispucava laser iz sebe. Kreće se od tornja koji asocira na top.



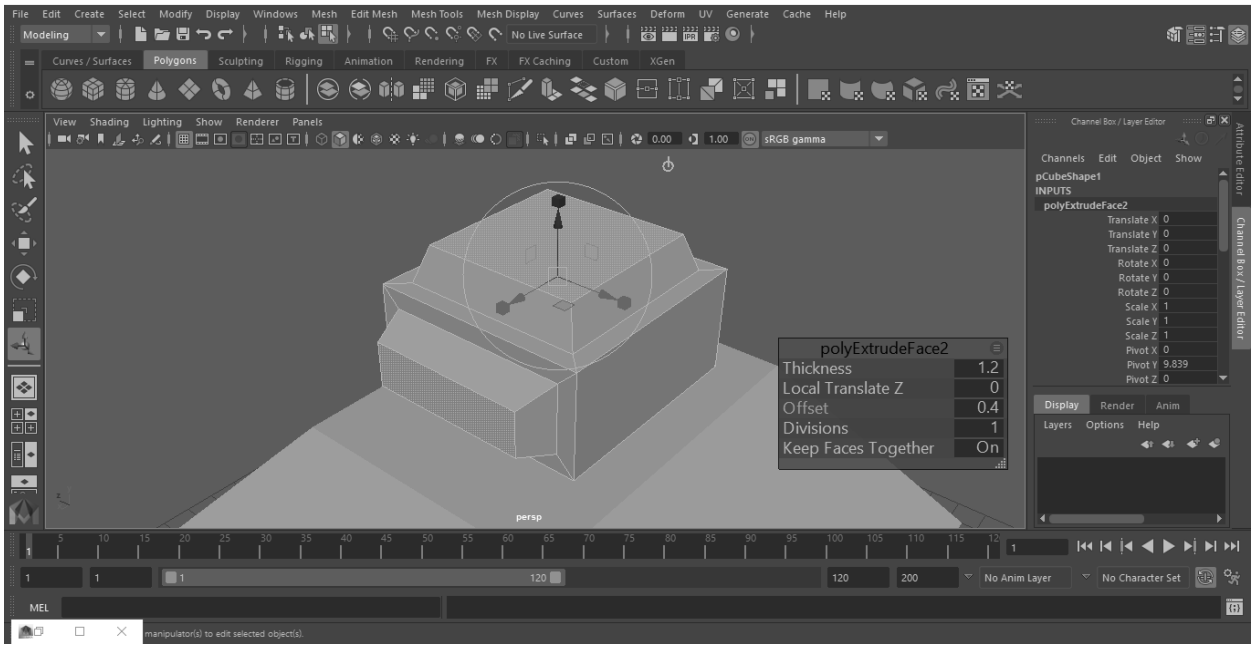
Slika 5.1. Vertex selekcija

Od stvorene kocke unutar Maye dolazi se do selekcije vertexa na modelu držanjem desnog klika miša kojim se oko miša stvara izbornik selekcije. Odabire se vertex selekcija i selektiraju se donji vertex kocke. Rastežu se dok se ne dobije željen izgled. S time je baza tornja gotova. Stvara se nova kocka koja će biti glava tornja i iz koje će ići cilindar iz kojeg toranj puca. S obzirom da će taj dio tornja biti dio koji prati neprijatelje, glava mora biti odvojena od baze, kako se ne bi cijeli toranj pomicao, znači toranj mora biti sastavljen od minimalno dva dijela. Na stvorenoj kocki koja služi kao glava, selektiraju se lijeva, desna i gornja strana kocke, odlazi se na izbornik oblikovanje geometrije (eng. Edit Mesh) i selektira se opcija izvlačenja (eng. Extrude).



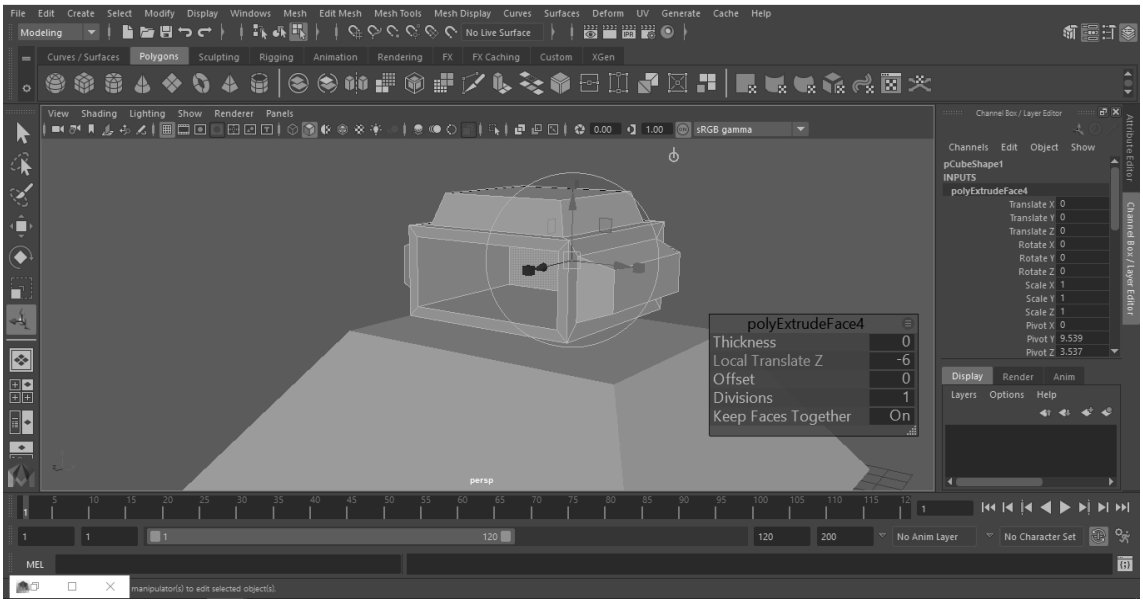
Slika 5.2. Extrude opcija

Odabire se opcija ne držanja poligona zajedno i unutar opcije istupa (eng. offset) upisuje se vrijednost 0.6. Taj proces stvara nova lica na postojećima i odvaja ih od rubova. Pritiskom na tipku "G" unutar Maye radi se zadnja komanda koja je odrađena, u ovom slučaju opcija izvlačenja (eng. Extrude). Novim istisnutim poligonima dajemo vrijednost debljine 1.2 i vrijednost ofseta 0.4, što udaljava poligone od početne pozicije i sužava ih.



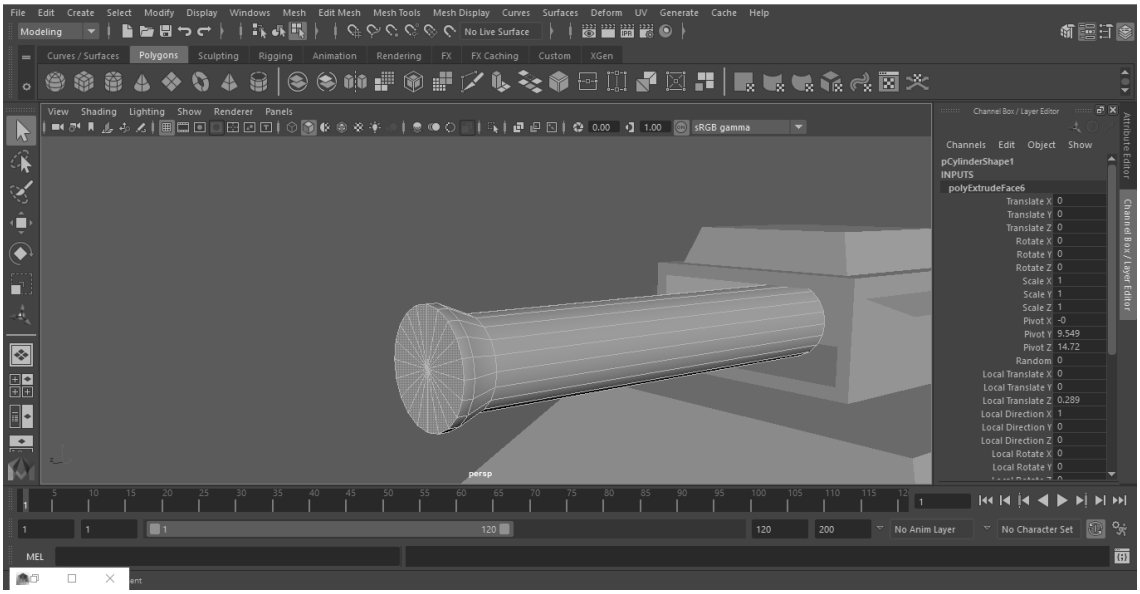
Slika 5.3. Modeliranje opcijom izvlačenja

Poslije bočnih strana, selektira se prednje lice glave tornja te ponavljanjem zadnje odrađene akcije pritiskom na tipku "G" odabire se ofset od 0.4 kako bi se lice odvojilo od rubova i još jednim pritiskom na tipku "G" i postavljanjem vrijednosti debljine (eng. thickness) na -6 dobija se udubljenje u koje će biti smještena cijev tornja.



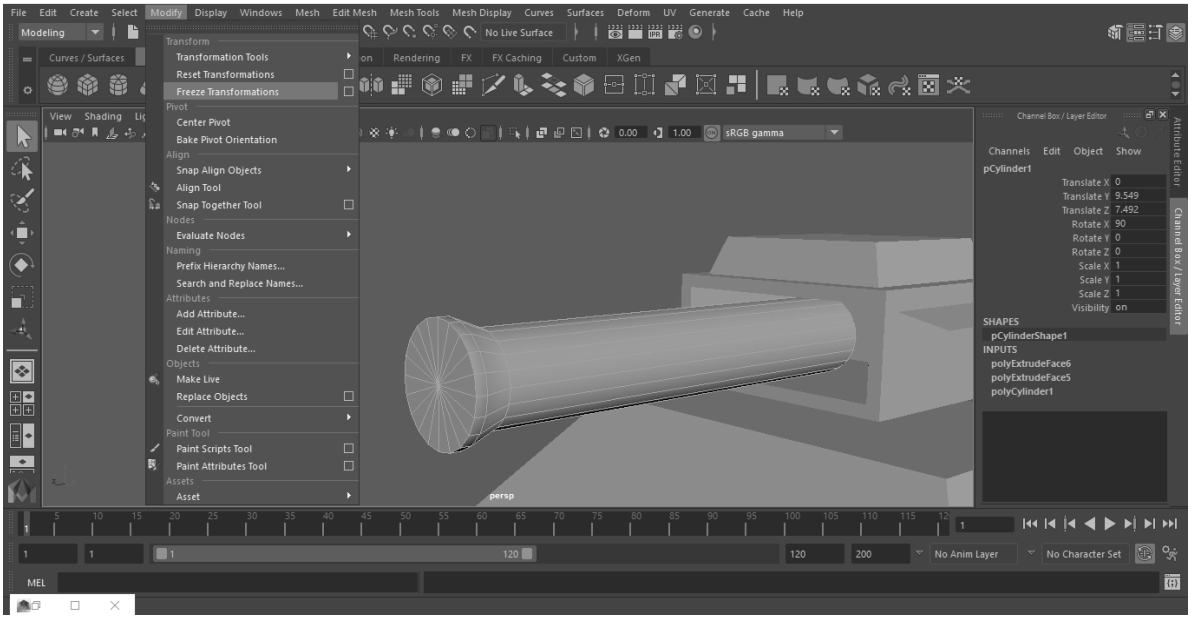
Slika 5.4. Udubljivanje površine

Cijev tornja stvara se od cilindra koji se postavlja unutar udubljenja na glavi koja je upravo napravljena. Selekcijom poligona vrha cijevi i praćenjem istog postupka modeliranja dobija se izbočina na cijevi koja reprezentira oblik cijevi topa.

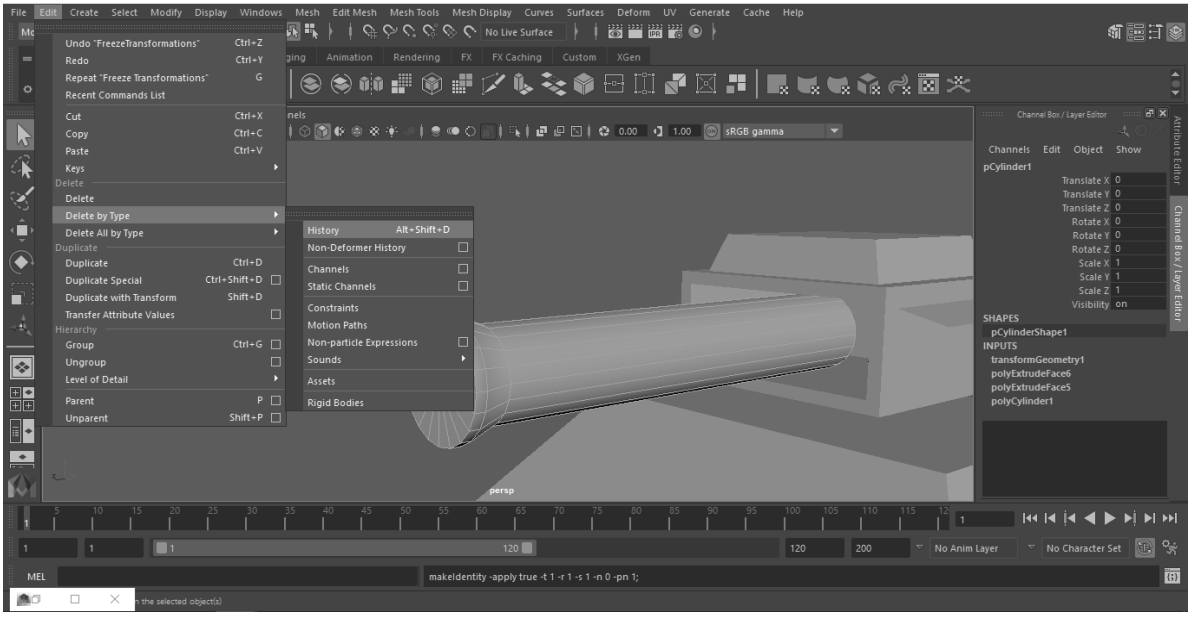


Slika 5.5. Modeliranje cijevi tornja

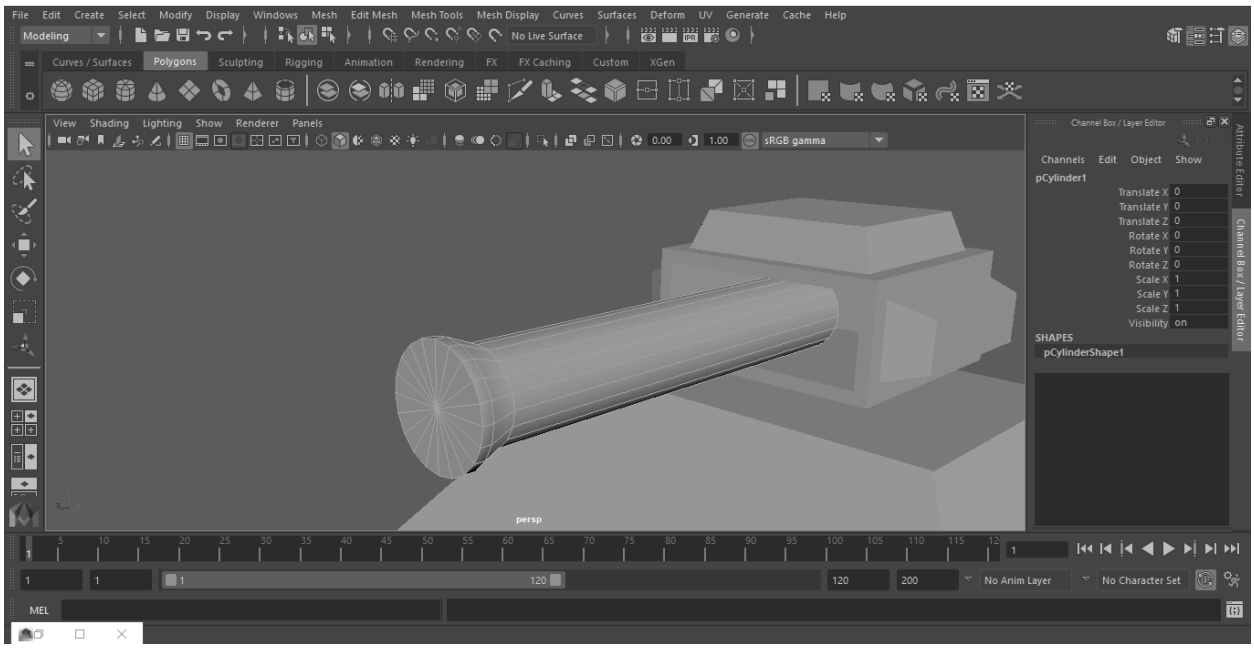
Potrebno je napraviti šupljinu na cijevi. Kako ne bi došlo do neželjenih grešaka, potrebno je selektirati cilindar kao objekt te mu zalediti transformacije i obrisati mu povijest uređivanja poligona koja se nalazi na njemu. Nakon dobijanja čistog modela moguće je udubiti cijev topa bez grešaka.



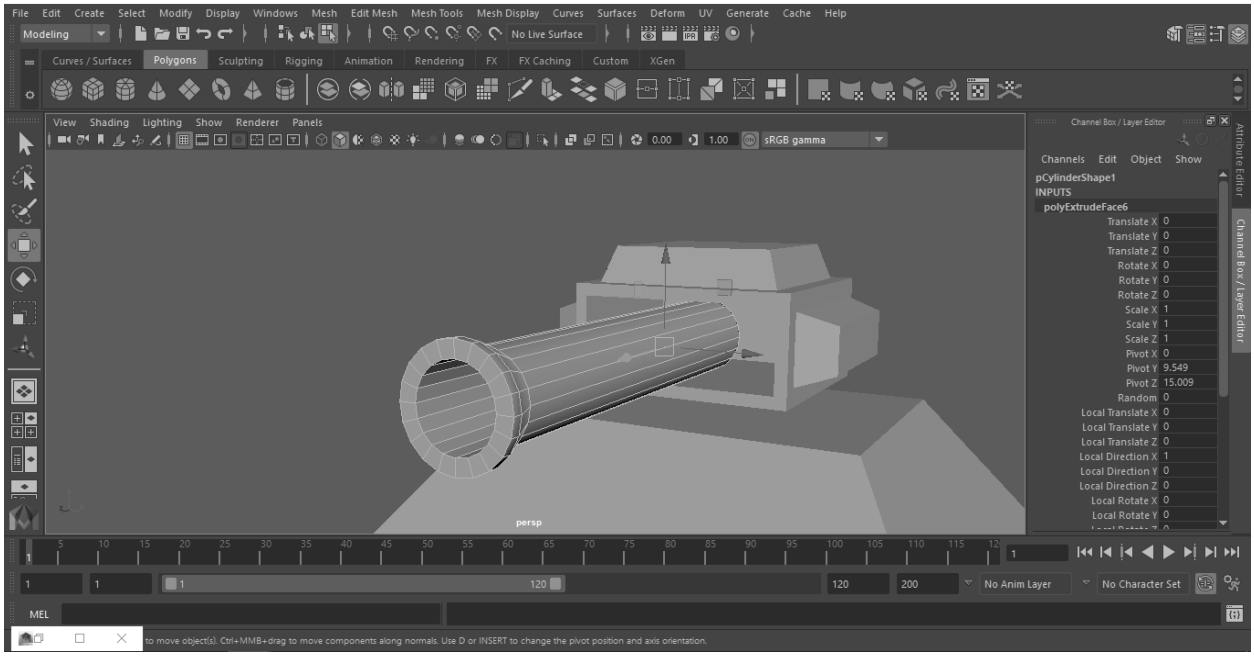
Slika 5.6. Brisanje vrijednosti transformacija



Slika 5.7. Brisanje povijesti modeliranja



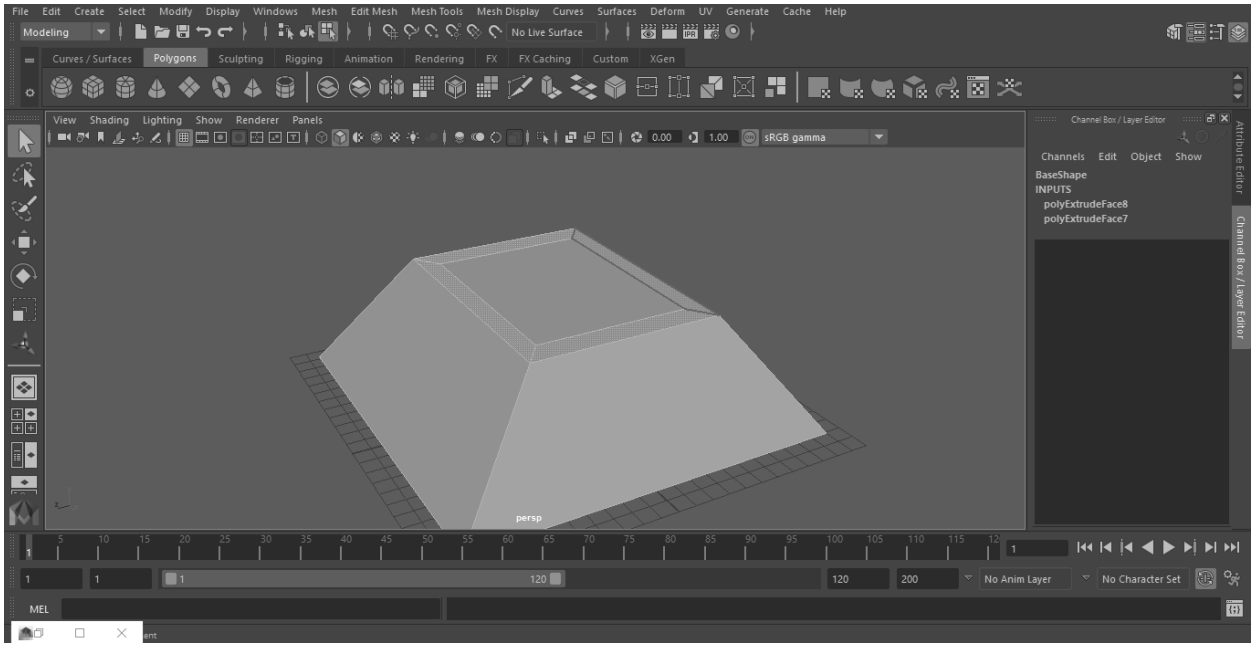
Slika 5.8. Primjer čistog modela



Slika 5.9. Udubljenje cijevi

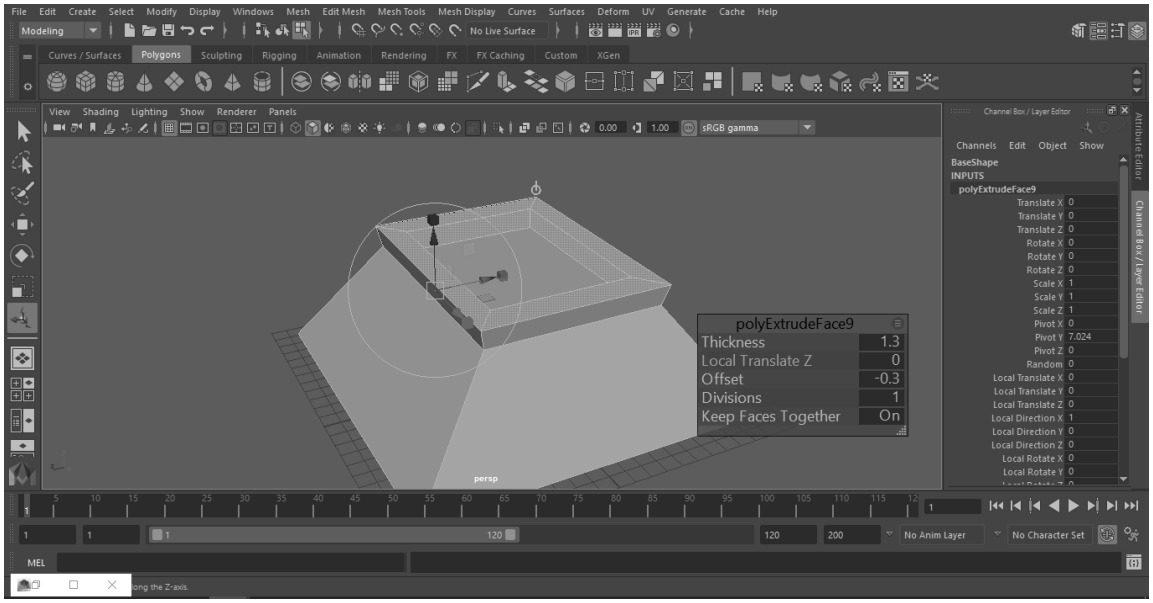
5.2. Minigun toranj

S obzirom da već postoji baza tornja koja je prethodno napravljena, to je odlična startna pozicija za modeliranje ostalih tornjeva. Kako bi se dobila raznolikost potrebno je izmijenit neke detalje. Selekcijom na vrh baze tornja proširuje se gornji poligon te se izvlači kako bi se dobio poligon s odvojenim rubovima.



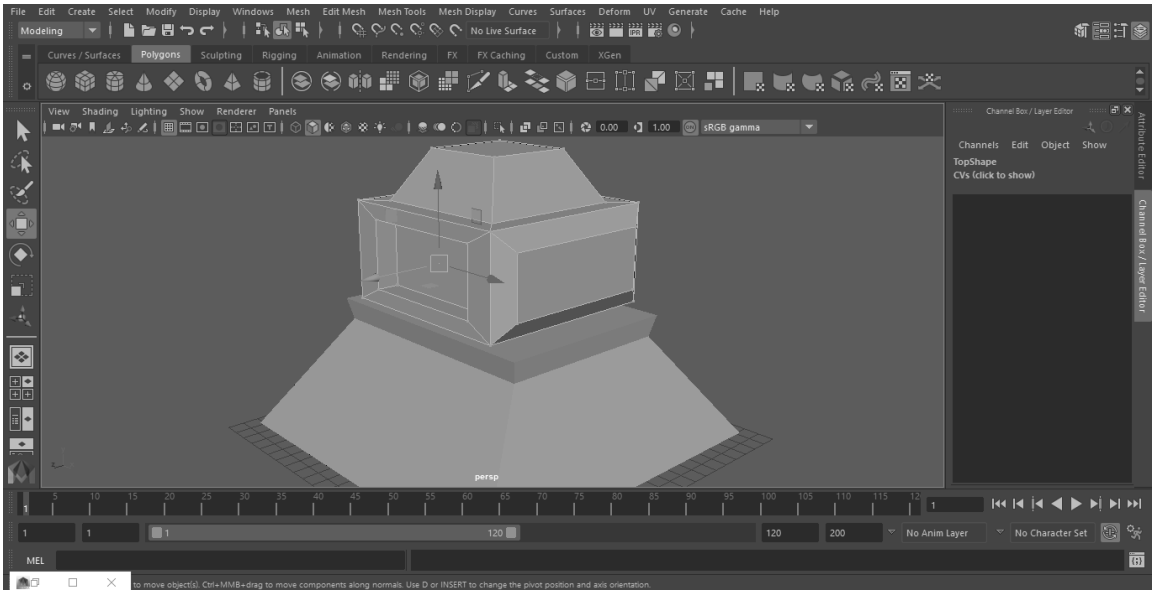
Slika 5.10. Početna baza minigun tornja

Ti rubovi će biti uzdignuti i prošireni. Lak način selekcije rubova je selektirati lice u sredini i pritisnuti kraticu Shift + >, što proširuje selekciju za jedan element oko selektiranog elementa. Zatim se shift+klikom na srednje lice deselektira to lice. Selektirana lica rubova sad je moguće uzdignuti opcijom izvlačenja.



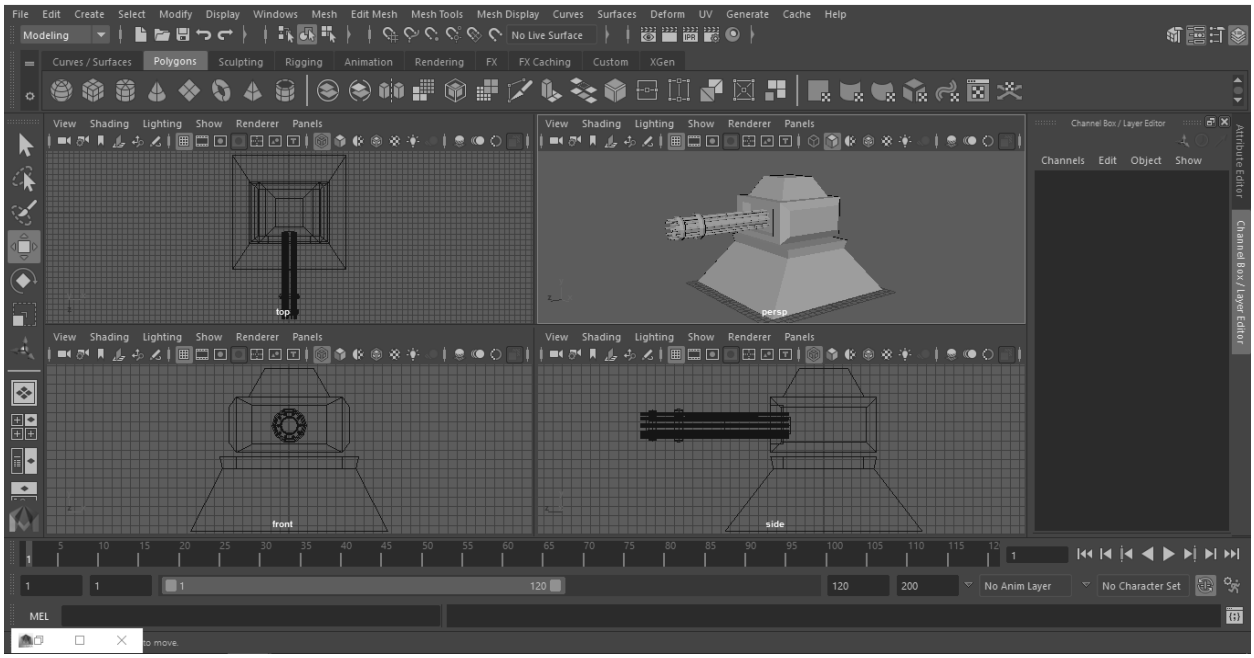
Slika 5.11. Uzdignute površine baze minigun tornja

Kako bi se dobila raznolikost glave tornja selektiraju se rubne točke modela (eng. vertex) udubljenja prednje strane i izvlače se na početak.



Slika 5.12. Glava minigun tornja

Sama cijev iz koje toranj puca sastoji se od dupliciranih cilindara koji su smješteni unutar još jednog šupljeg cilindra.



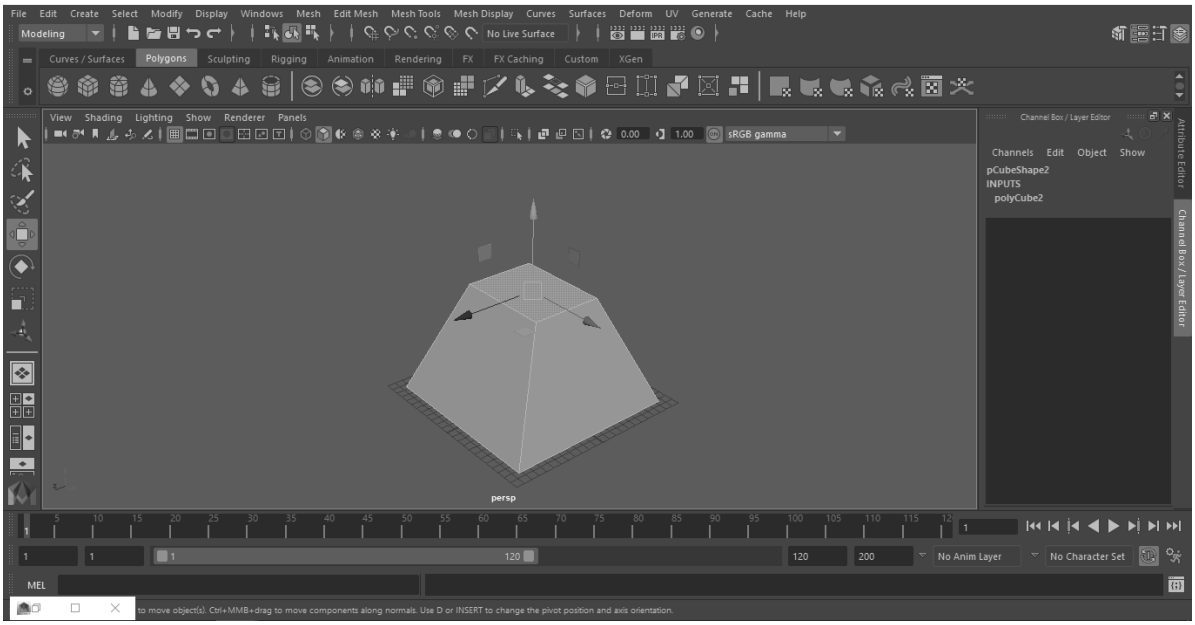
Slika 5.13. Završna verzija minigun modela tornja

Slika 5.13. prikazuje završni model dobiven korištenjem svega par alata unutar Autodesk Maya programskog okruženja kroz četiri perspektive: gornja, bočna i frontalna izometrična perspektiva te jedna klasična perspektiva. Proces modeliranja je odrađen manipulacijom elemenata poput točaka gdje se lica modela spajaju, podešavanjem njihovim međusobnih razmjera, visine i širine.

Dodavanje detalja poput drukčijih rubova glave modela je ključni dio u dobivanju raznolikosti modela. Niska i široka baza tornja daje dojam nabijenosti i strukturalne čvrstoće što se i očekuje od jednog tornja vojnog kapaciteta.

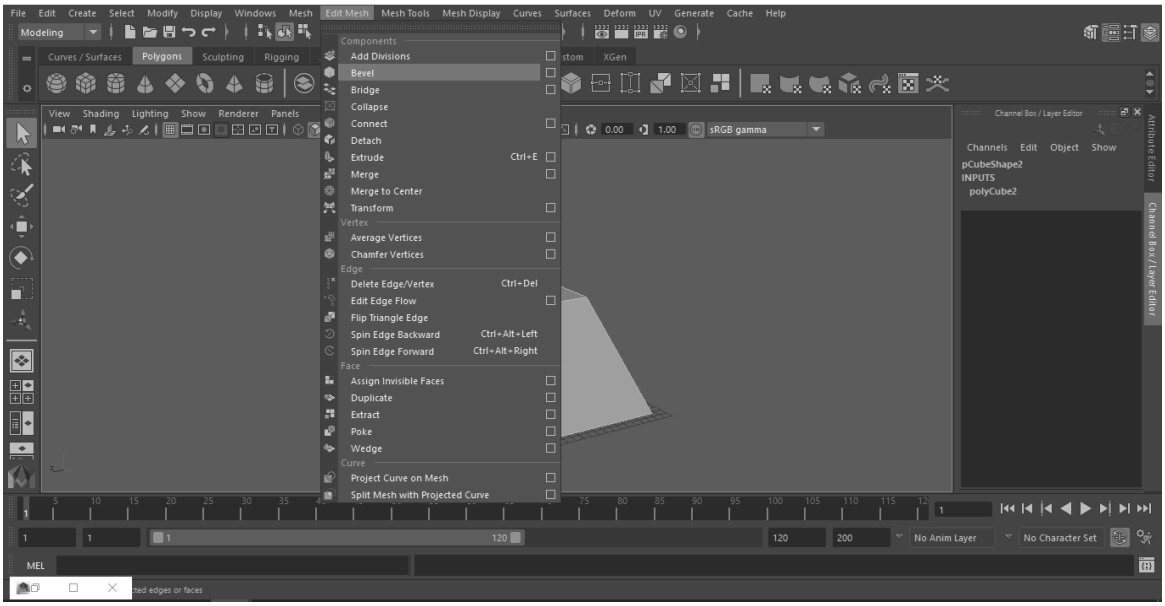
5.3. Vatreni toranj

Također, kao i minigun toranj, kreće se od postojeće baze. Kako bi se više razlikovao od drugih, vatreni toranj će imati izduženiji oblik. Selektira se gornji poligon, izvlači se prema gore i sužava.

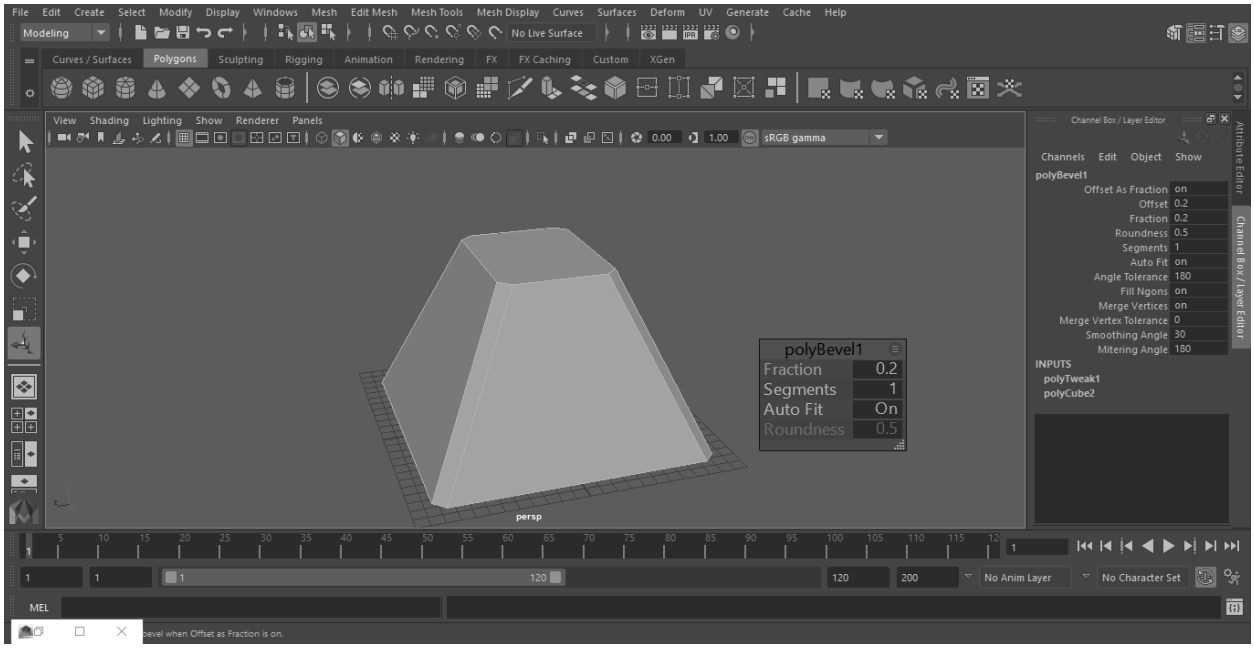


Slika 5.14. Početna baza vatrenog tornja

Označavaju se bočni rubovi baze te se pomoću alata za stvaranje nagiba (eng. bevel).

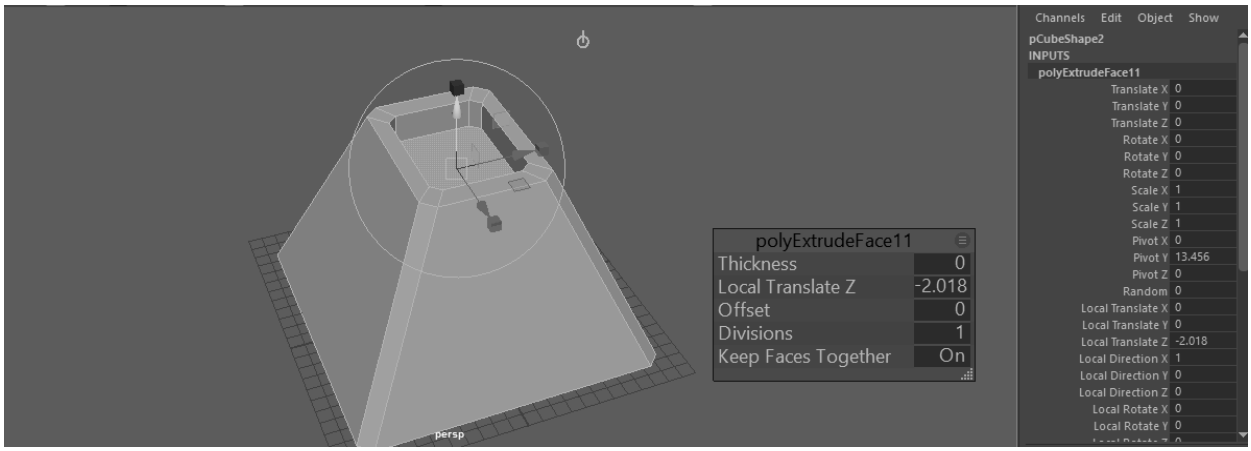


Slika 5.15. Bevel opcija



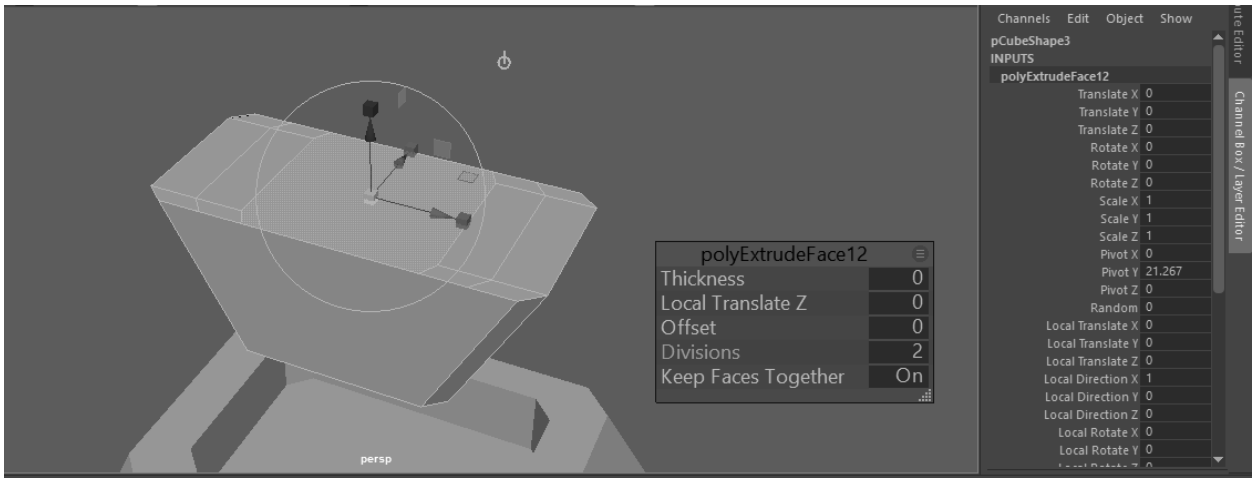
Slika 5.16. Rubovi baze zaobljeni bevel opcijom

Gornji dio baze se udubljuje metodom izvlačenja iz prijašnjih primjera.



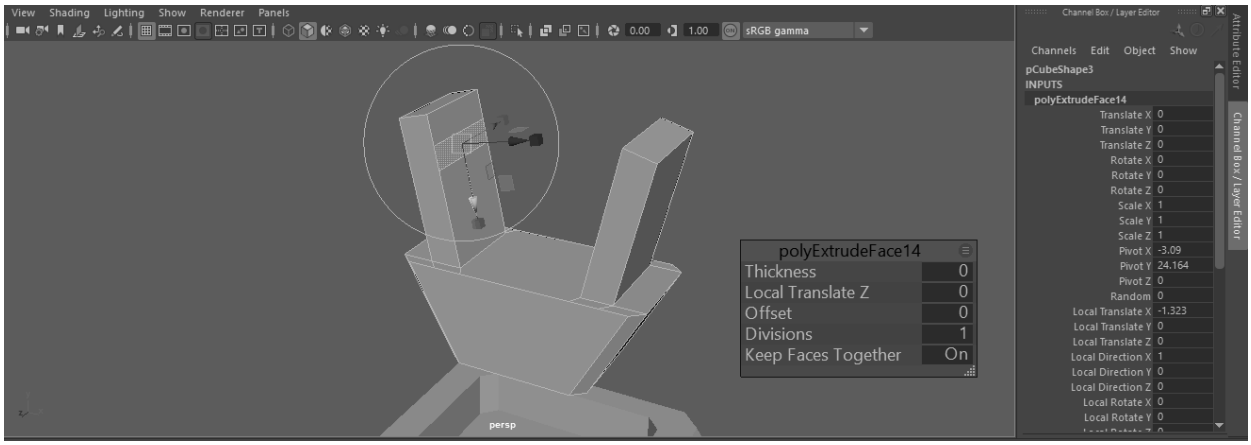
Slika 5.17. Završna baza vatrenog tornja

Glava tornja se stvara od kocke kojoj se, nakon pozicioniranja, odabiru se gornje rubne točke i razvlače se. Također se na rubovima glave tornja koristi opcija nagiba kako bi se dizajn glave uklopio s bazom.



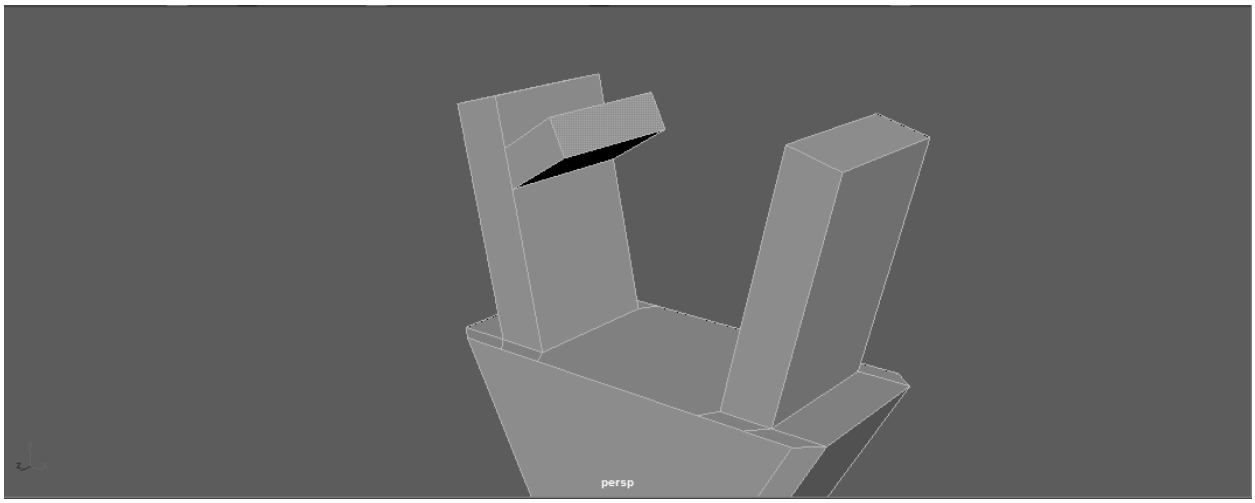
Slika 5.18. Početni oblik glave vatrenog tornja

Gornja površina glave se izvlači te se unutar opcije podjele (eng. divisions) odabire broj 2 kako bi se lice razdvojilo za sljedeći korak. Odabiru se dva lica te se metodom izvlačenja izdižu gore i malo odvajaju jedno od drugog.

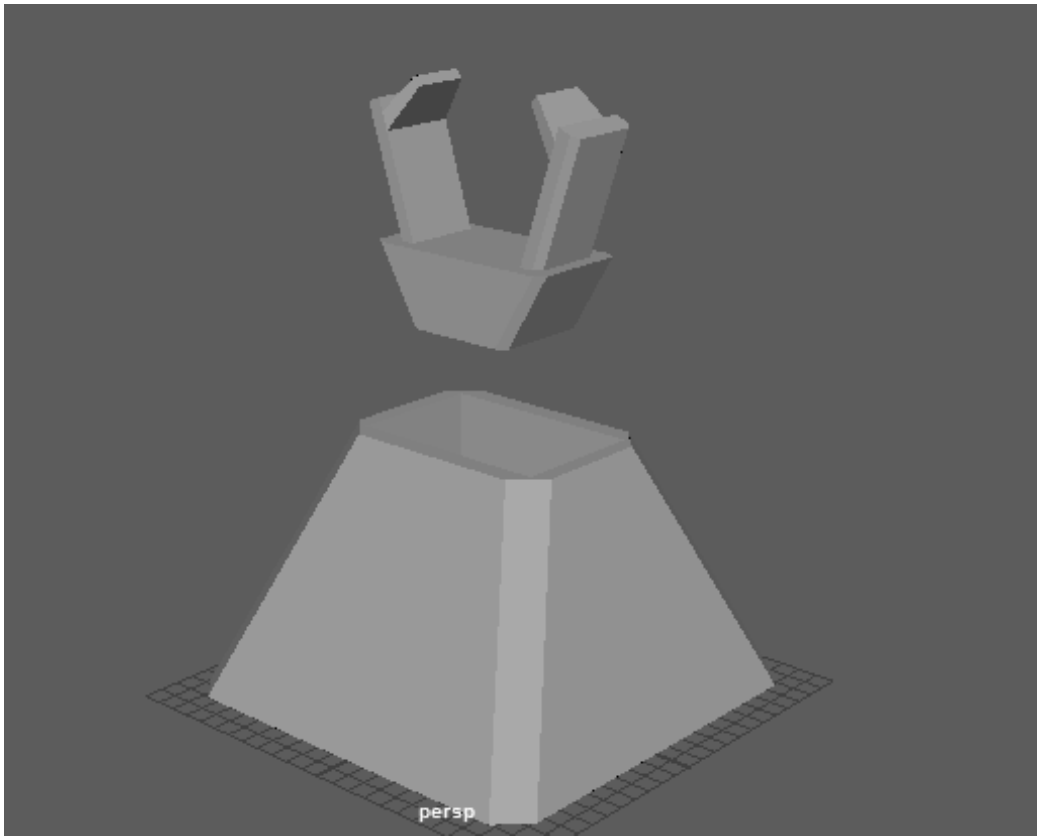


Slika 5.19. Oblikovanje glave vatrenog tornja

Unutarnja lica dva nastala dijela se odvajaju u zasebno lice koje se izvlači prema centru glave i podiže gore gdje će biti točka pucanja.



Slika 5.20. Završno oblikovanje glave vatrenog tornja



Slika 5.21. Završni model vatrenog tornja

6. Animacija

U sve brže rastućem svijetu digitalne umjetnosti sve veću ulogu imaju računalni programi koji nam omogućuju iluziju trodimenzionalnosti. Od razvijača Indie igara, CG efekata, marketinških reklama i kampanja, sve je veća potražnja za 3D umjetnicima i onima koji se snalaze u raznim alatima namijenjenim razvoju trodimenzionalnih likova u virtualnom prostoru. Ovaj rad će se fokusirati na aspekt animacije, te će obuhvatiti sve stavke koje su nam bitne kao animatorima. Rad ne sadrži nazive računalnih programa koji služe za animaciju, jer je sam računalni program nebitan za razumijevanje toka animacije. [11]

6.1. 12 Načela Animacije

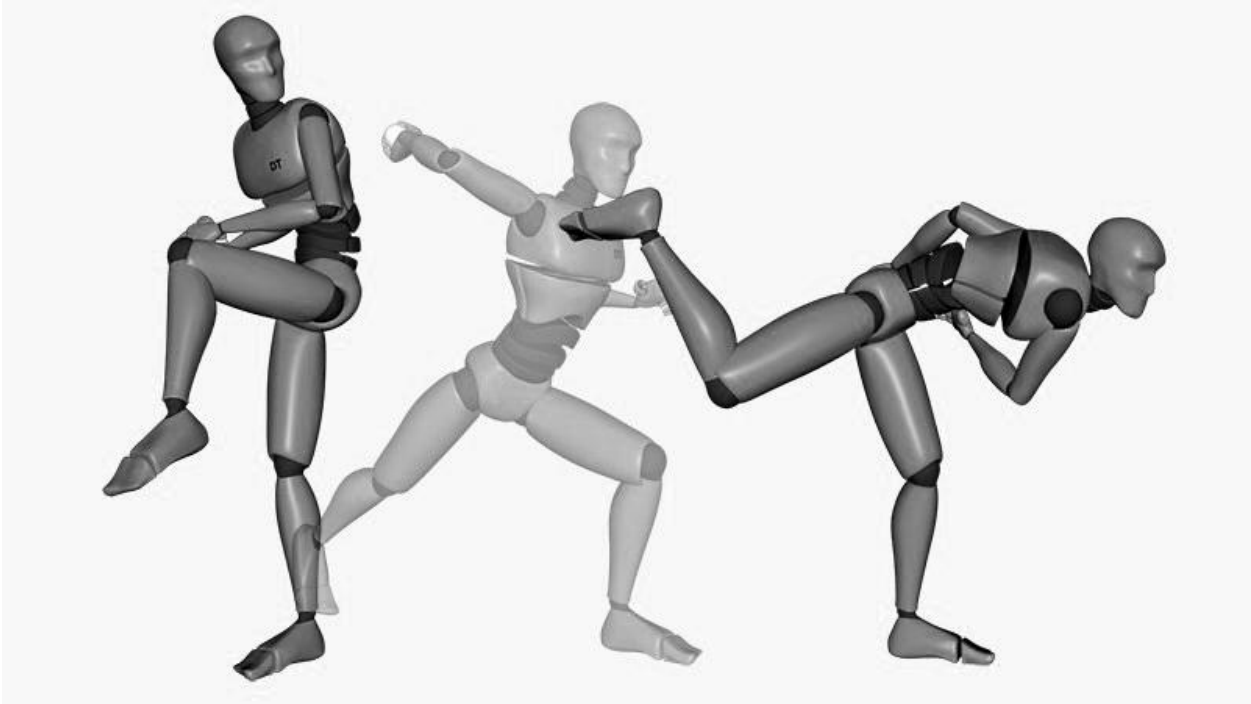
1. Izduživanje, spljoštavanje (eng. Squash & Strech)

Načelo koje govori da će se animirani objekti izduživati ili spljoštavati kako bi naglasili svoju brzinu, momentum, težinu i/ili masu. Kod tehnike squash&strecha je od ekstremne važnosti održavanje proporcija, to jest, ne gubiti masu tijela koje se kreče. Ne mora biti vezano samo za objekte, squash&strech se može odnositi i na ekspresije lica.

Slika 6.1. Primjer izduživanja i spljoštavanja [13]

2. Predviđanje (eng. anticipation)

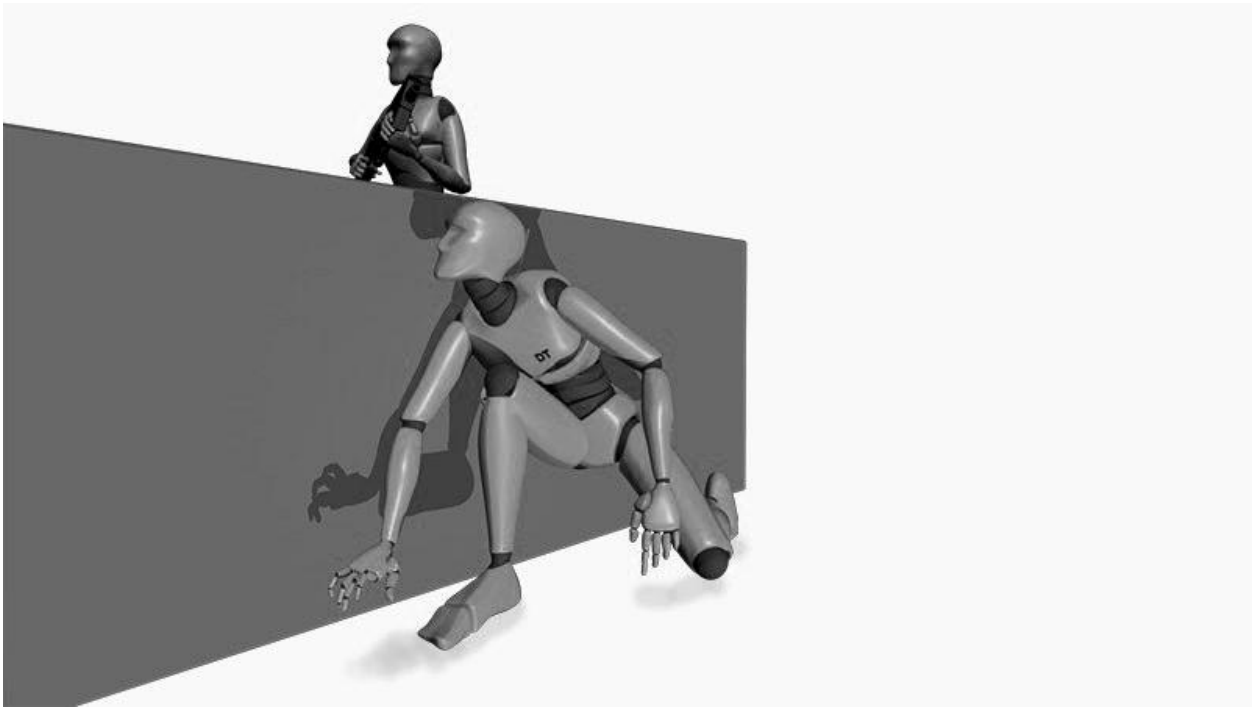
Načelo predviđanja se odnosi na „akciju prije akcije“, to jest kada se lik priprema izvesti neku akciju, kako bi dao publici ideju što će napraviti i također kako bi se izvedena akcija činila realnijom. Svojom akcijom lik priprema publiku za sljedeću akciju. Najbolji su primjeri skakanje lika (jer energija skoka ne dolazi od ničega, lik se spušta prvo), priprema za trčanje, baseball igračevo bacanje lopte i slično.



Slika 6.2. Primjer predviđanja [13]

3. Sceniranje (eng. Staging)

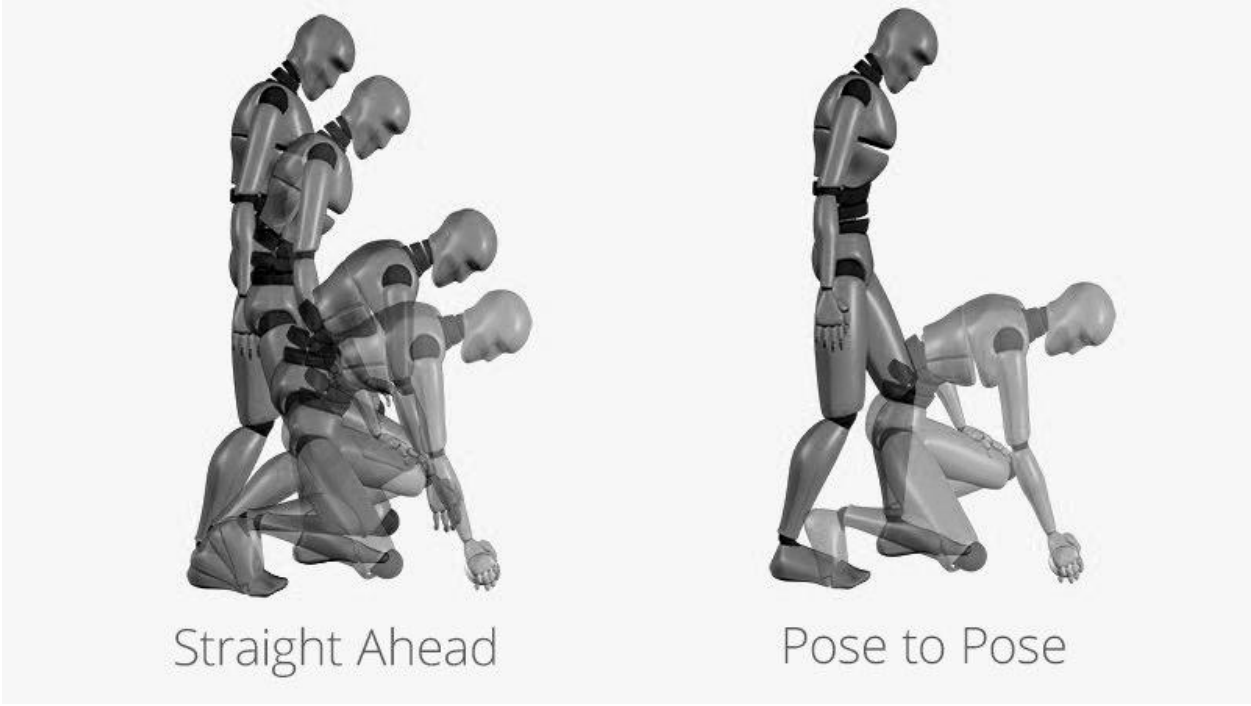
Prezentiranje bilo koje ideje tako da je kompletno i nedvojbeno jasna. Široko načelo jer pokriva puno područja animacije. Može se povezat sa tajmingom, glumom, kutom i pozicijom kamere, i postavkom scene. Kontroliramo gdje publika gleda, jednostavno. Kamera igra veliku ulogu, bitno je znati kada raditi close up a kada široki kadar. Široki kadar je savršen za velike akcije dok je close up najbolji za emotivne ekspresije. Glavne akcije se trebaju odrađivati ili u centru kadra, ili je jednoj od trećina ekrana. Tipa, ako lik gleda negdje, treba gledat prema prostoru s vise mjesta na ekranu, osim ako mu se neko ne prikrada s leđa, s čime više nije glavni lik scene. Ako je neki detalj bitan, trebao bi se zadržati na ekranu dovoljno dugo da ga se naglas pročita tri puta. (ime proizvoda, marka automobila, i sl.)



Slika 6.3. Primjer sceniranja [13]

4. Izravno – Poza u pozu (eng. Straight Ahead - Pose to Pose)

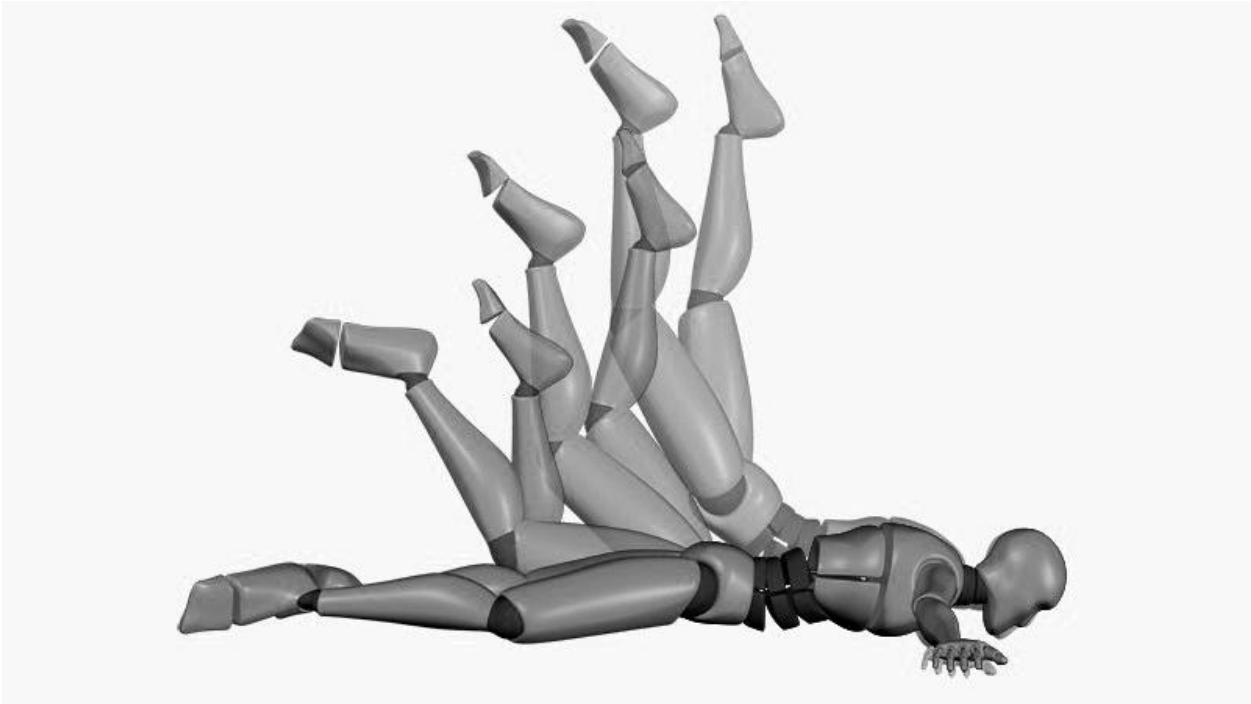
Govori o tipovima animiranja. Izravna metoda (eng. straight ahead) bi bilo crtanje (ili postavljanja poze) jednog okvira ili trenutka, pa sljedećeg, pa sljedećeg, manje više animacija u toku, jedan smjer (od A do B). Druga metoda, poza u pozu (eng. pose to pose), se bazira na crtanju početnog okvira, zadnjeg, onog između, te se te glavne poze spajaju sa među pozama. Postoje prednosti kod obje metode. Poza u pozu je generalno bolja metoda za više akcija, jer daje veću kontrolu animacije. Daje rane rezultate i jasnu ideju kako će animacija izgledat u ranim fazama razvoja. Izravna metoda animiranja je dobra za akcije koje su nepredvidljive, tipa vatra, čestice vode, vjetar, eksplozije. Također se sa izravnom metodom može animirati kretanje kose, repa, kretanje dugačkih ušiju, sve što prati radnju lika. Kod metode poze u pozu postoji neki vokabular ili terminologija koja se koristi: glavne ili ključne poze (eng. keys), sekundarne ili ekstremne poze (eng. extreams), dalje razrađene poze (eng. breakdowns). Prvo se rade ključne poze, usavršavaju se, onda se odlučuje koliko će daleko lik ići u svakoj pozi koristeći ekstreme, te se na kraju odlučuje kako će se ekstremi povezati koristeći dalje razrađene poze.



Slika 6.4. Primjer izravnog animiranja i animiranja iz poze u pozu [13]

5. Sila akceleracije (eng. Folow Through - Overlaping Action)

Dijelovi tijela, nakit, šeširi, plašt i td. se vuku za tijelom, i nastavljaju kretnju kada tijelo stane. Ova tehnika se često veže uz fizičko svojstvo otpora tijela (eng. drag). *Folow Through*, *Overlaping Action*, i *Drag* ustvari opisuju istu stvar al na drukčije načine. *Folow Through* se odnosi na način na koji se dijelovi tijela kreću nakon sto je tijelo stalo. *Overlaping Action* se odnosi na istup tijela u odnosu na druge dijelove tijela. *Drag* opisuje pauzu u kretanju dijelova tijela u odnosu na glavno tijelo. Svo troje opisuju isti aspekt iste stvari.



Slika 6.5. Primjer načela sile akceleracije[13]

6. Tranzicija animacije (eng. Slow in - Slow out)

Govori o tome kako kretnje počinju sporo i završavaju sporo. Bez toga kretnja bi izgledala mehanički. Roboti se kreču konstantnom brzinom. Crteži, pozicije i okviri bi trebali biti bliži na početku, sredini i na kraju, to jest na dijelovima za koje želimo da publika primijeti više.

Slika 6.6. Primjer tranzicije animacije [13]

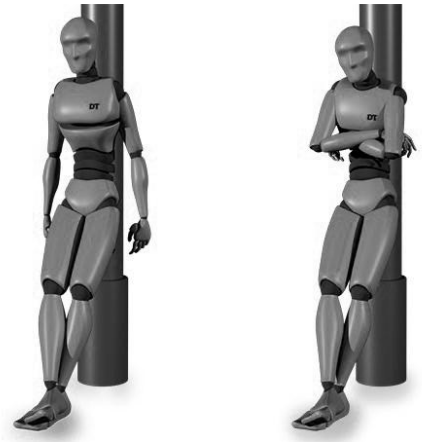
7. Lukovi (eng. Arcs)

Malo stvari se može kretati sa monotonim linearnim kretnjama (gore dolje, lijevo desno). Većina živih bića se kreću eliptično, to jest po luku. Znaju biti jako mali lukovi u određenim tranzicijama animacije iz poze u pozu. Da nema tih lukova akcije koje se izvode bi izgledale čudno i neprirodno. Na sve utječe gravitacija.

Slika 6.8. Primjer lukova u animaciji [13]

8. Sekundarne radnje (eng. Secundary Action)

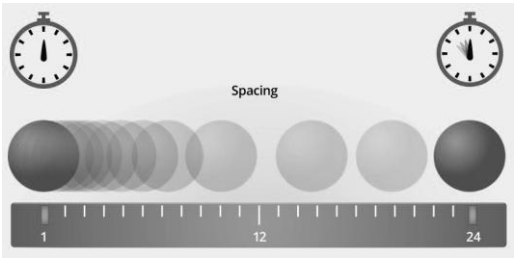
Opisuje geste koje dodatno naglašavaju glavnu akciju. Tipa lik koji hoda ljut, primarna (glavna) akcija su noge, dok je sekundarna akcija sve ostalo, zamah ruku, ekspresije lica, i td. Primjer kucanja na vrata, ako je druga ruka stisnuta znači da je ljut, ako je uz prsa dok glava gleda lijevo desno znači da ne želi biti neprimjetan.



Slika 6.9. Primjer sekundarne radnje [13]

9. Proračun vremena (eng. Timing)

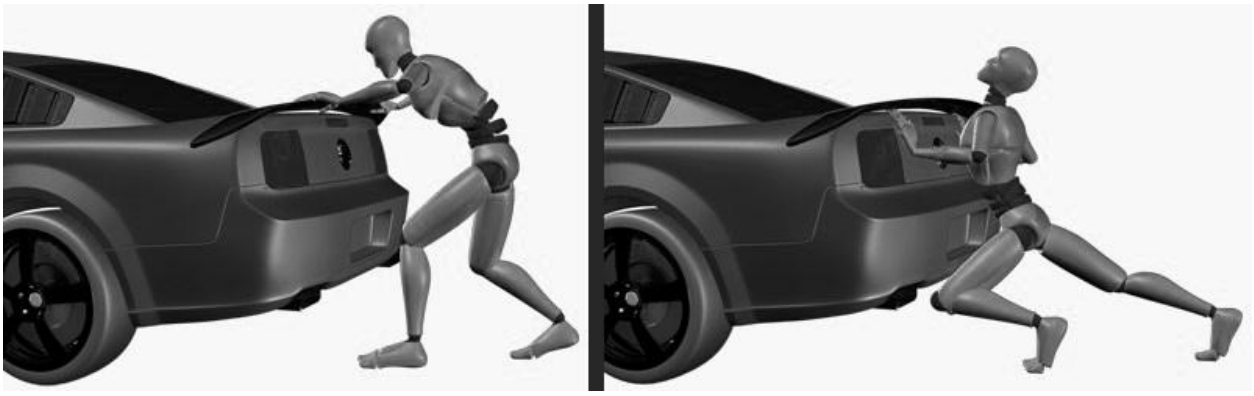
Broj okvira između glavnih akcija (eng. timing) ima veliki utjecaj na osobnost i prirodu animacije. Ako postoji puno crteža ili veliki vremenski razmak između dvije akcije, akcija će biti spora, ako je malen razmak akcija, akcija će biti brza. Jedna akcija može imati više značenja ovisno o vremenu koje izdvojimo za njeno prikazivanje. Za razliku od klasične animacije koja zahtjeva crtanje svakog okvira, u 3D animaciji postoji veća sloboda kad je riječ o proračunu vremena. Ne postoji opasnost „štekavog“ izgleda animacije, jer se ne može krivo nacrtati okvir, samo krivo postaviti.



Slika 6.10. Primjer tajminga [13]

10. Pretjerivanje (eng. Exaggeration)

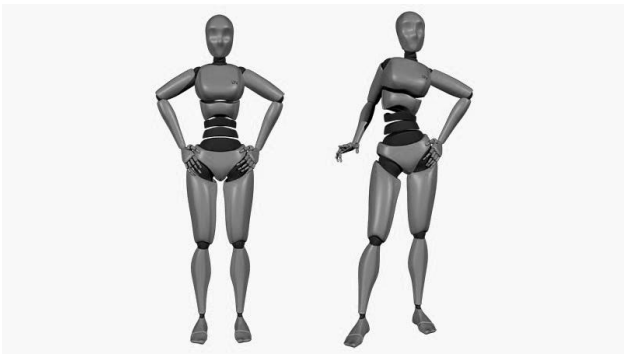
Svaka akcija, poza, djelovanje lika, se može dovesti do novog nivoa kako bi se povećao utjecaj na gledatelja. Pretjerivanje ne znači više distorzije, nego veća uvjerljivost. Kada bi Walt Disney govorio svojim animatorima da učine scenu „realnijom“, oni bi doslovno crtali realističkim stilom, Walt bi negodovao jer ga nisu shvatili što je tražio od njih. Neka radnja, sa određenom dozom pretjerivanja će uvijek biti privlačnija oku od one „normalne“ radnje. Dobra navika je vući animaciju do granica pucanja, to jest pretjerati sa radnjom pa kasnije smanjivati.



Slika 6.11. Primjer pretjerivanja [13]

11. Pravilno postavljanje (eng. Solid Drawing)

U 3D-u nam ne treba toliko, to jest ne zamaramo se crtežima nego pažnju obraćamo na siluete, manje više je bitno zadržavanje proporcija i održavanje načela perspektive. U 3D-u se načelo odnosi više na stabilizaciju likova, pravilna raspodjela težine i tako dalje. Potrebno je izbjegavat simetrično kretanje ruku i nogu (eng. twinning), zbog svoje ne prirodnosti. Treba se naglasit da lik mora balansirat svoju težinu, to jest da gledatelj zaboravi da se radi o virtualnom prostoru.

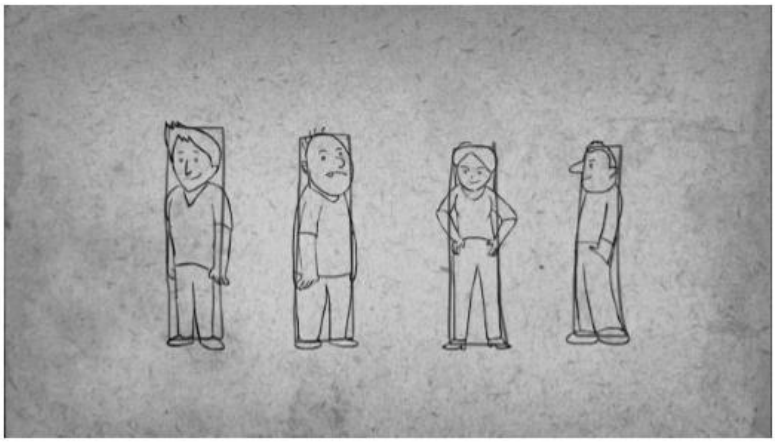


Slika 6.12. Primjer pravilnog postavljanja [13]

12. Vizualna ugodnost (eng. Apeal)

Likovi bi trebali biti ugodni za gledanje. Ne odnosi se samo na heroje scene, nego na sve likove. Vizualna ugodnost ne mora značit zgodan, ili lijep, više zanimljiv za gledat. Problem kod ovoga je što nije svim ljudima ista stvar privlačna, to jest zanimljiva. Dinamično dizajniranje likova pomaže kod modeliranja privlačnih likova. Primjer: igrati se sa oblicima tako da nisu svi likovi u jednoj kocki nego da su različiti, neki eliptični, neki špicasti, i tak dalje. Igranje sa proporcijama: povećavanje ruku, glave, onoga što označava lika. NEKA BUDE JEDNOSTAVNO, svest detalje na minimum kako ne bi zatrpali gledatelje nebitnim informacijama koje nisu bitne za ono što želimo prenijet ili dočarati.

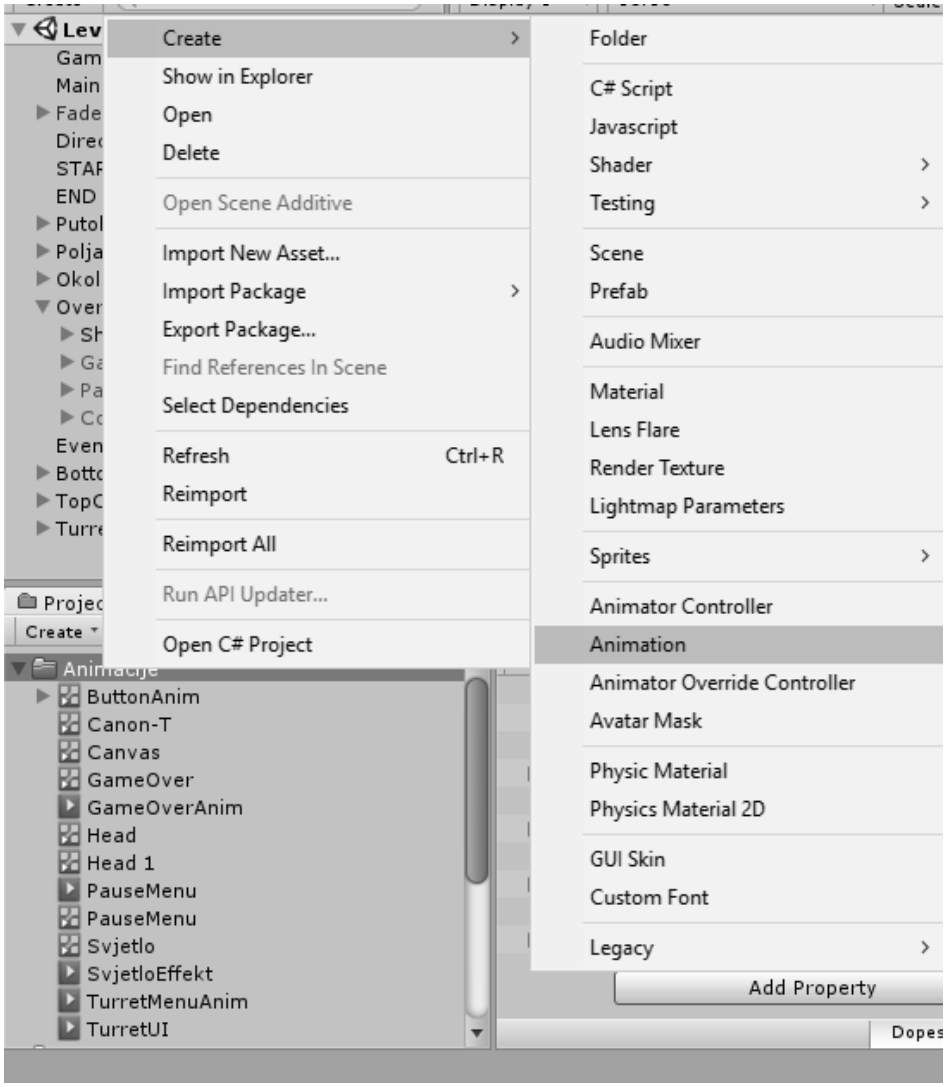
Slika 6.13. Primjer vizualne ugodnosti 1 [14]



Slika 6.14. Primjer vizualne ugodnosti 2 [14]

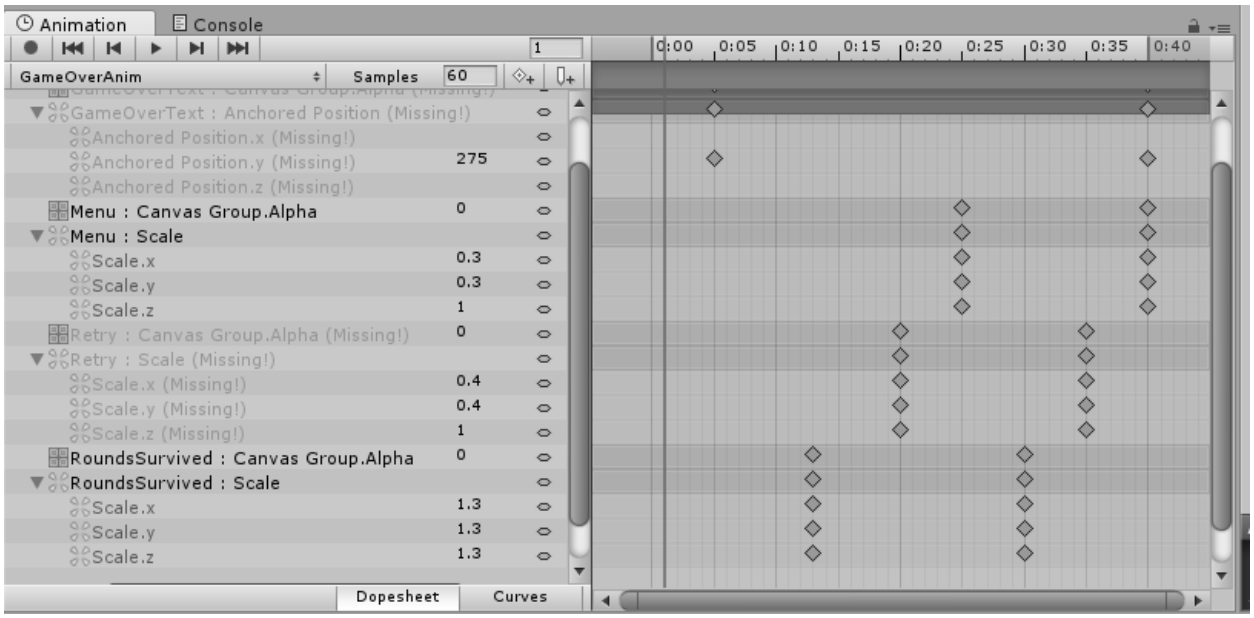
6.2. Animacija korisničkog sučelja

Kako korisnik ne bi dobio dojam monotonije i statičnosti potrebno je primijenit neka svojstva animacije na korisničkom sučelju. Jednostavan način izbijanja monotonije je sa načelom animacije "Tranzicija animacije". Potrebno je na neki način stvoriti laganu tranziciju elemenata korisničkog sučelja. To se može napraviti sa laganim prijelazom alfa kanala (prozirnosti) samih prozora poput *GameOver*, *PauseMenu* i *CompleteLevel* platna (eng. canvas). Trenutna mehanika igre govori da ako je igrač zadovoljio uvijete koji su potrebni za prelazak trenutnog nivoa, treba se omogućiti, to jest aktivirati *CompleteLevel* sučelje. Stvara se nova animacija desnim klikom na mapu u koju se želi animacija stvoriti, odlaskom na *Create* odabire se *Animation*.



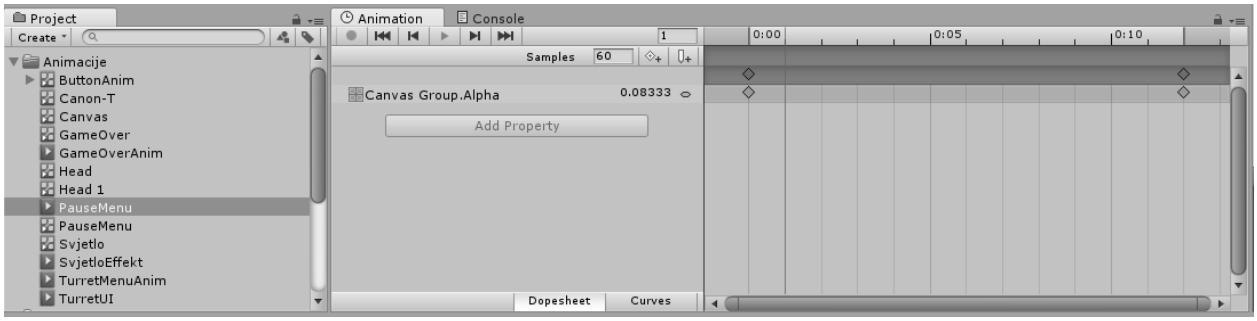
Slika 6.15. Stvaranje nove animacije objekta

Unutar ovog prozora prikazani su sve komponente koje se mogu animirati, uključujući alfa kanal samog elementa. Princip animiranja je isti kao u svim drugim programima, postavljanje početnog i završnog okvira i njihovih vrijednosti.



Slika 6.16. Komponente koje je moguće animirati

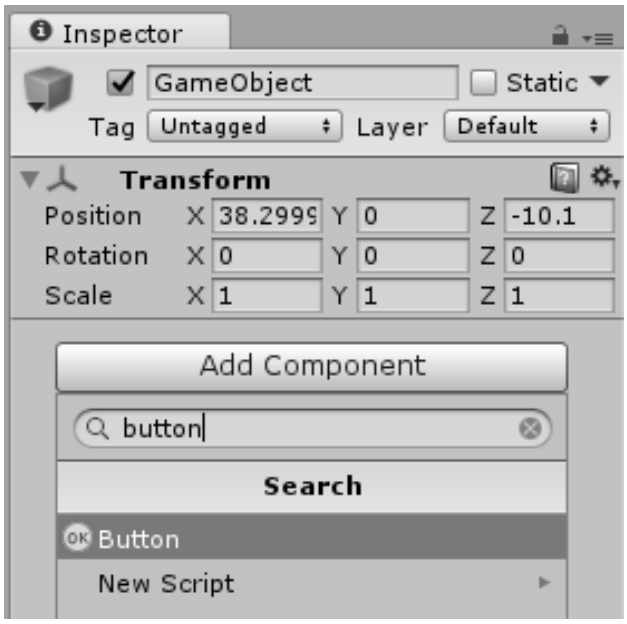
U ovom slučaju animacija će biti puštena pri aktivaciji samog objekta, tako da se ne trebaju implementirati stvari poput petlje animacije (eng. loopTime), animacija je jednostavna, od A do B. Objektu se govori da napravi tranziciju alfa kanala od 0 do 1 kroz neko vrijeme t, te da ga također kroz to vrijeme poveća (eng. scale).



Slika 6.17. Animacija alfa kanala objekta

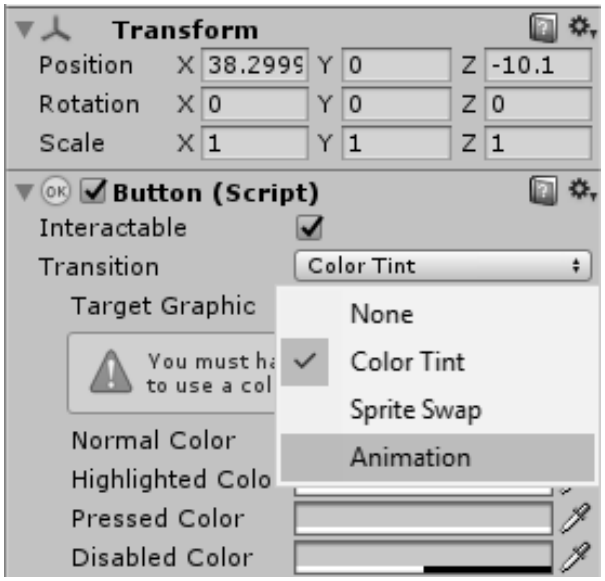
Slična animacija se radi i za *PauseMenu* objekt, s time da ta animacija nema nikakvu drugu vrijednost animiranu, samo alfa kanal.

Animacije vezane za gumbove unutar scene za odabiranje nivoa animirane su putem Unityjevog ugrađenog *ButtonAnimatora*.



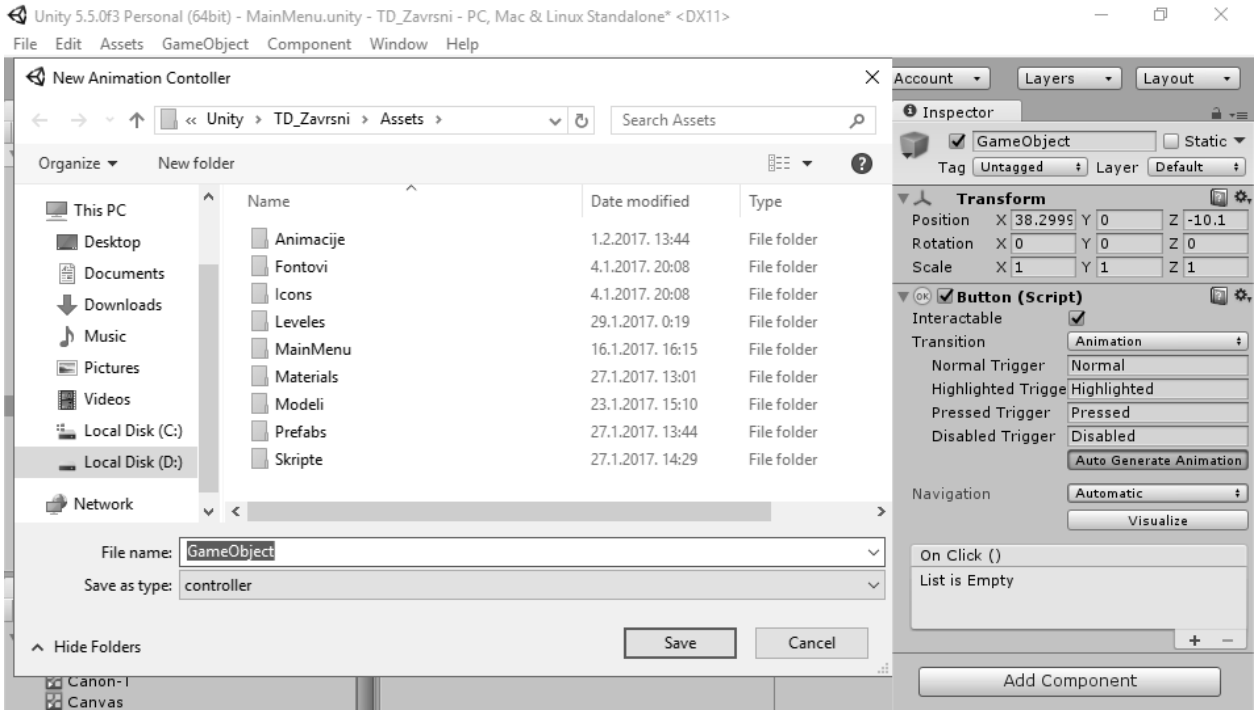
Slika 6.18. Dodavanje Button skripte na objekt

Selekcijom objekta gumba koji želimo animirati dodajemo komponentu zvanu *Button* te se toj komponenti mijenja *Transition* vrijednost sa *ColorTint*, koja je automatski postavljena, na *Animation*.



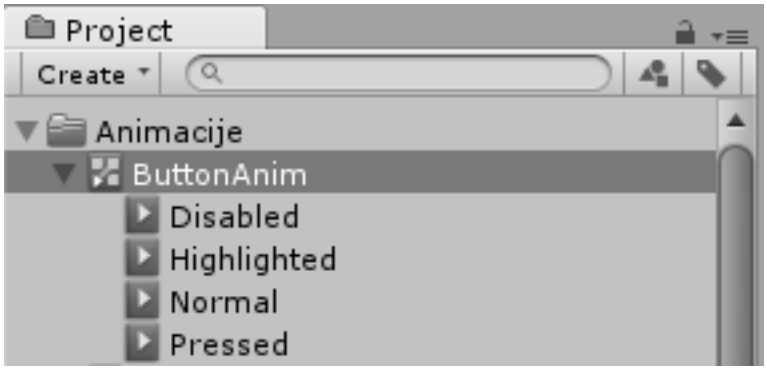
Slika 6.19. Izmjena vrste tranzicije

Zatim se klikom na *Auto Generate Animation* otvara prozor koji nudi mjesto gdje će sam kontrolor animacije biti spremljen. Nakon odabira lokacije i imena kontrolora stvara se komponenta kontrolora na odabranoj lokaciji.



Slika 6.20. Spremanje kontrolera animacije

Unity automatski stvara četiri stanja animacije u kojem gumb može biti, normalno stanje, stisnuto, onemogućeno i označeno. Svako stanje se može zasebno animirati. Velika prednost ovog sistema je ta što može postojati jedan kontrolor za više gumbova.



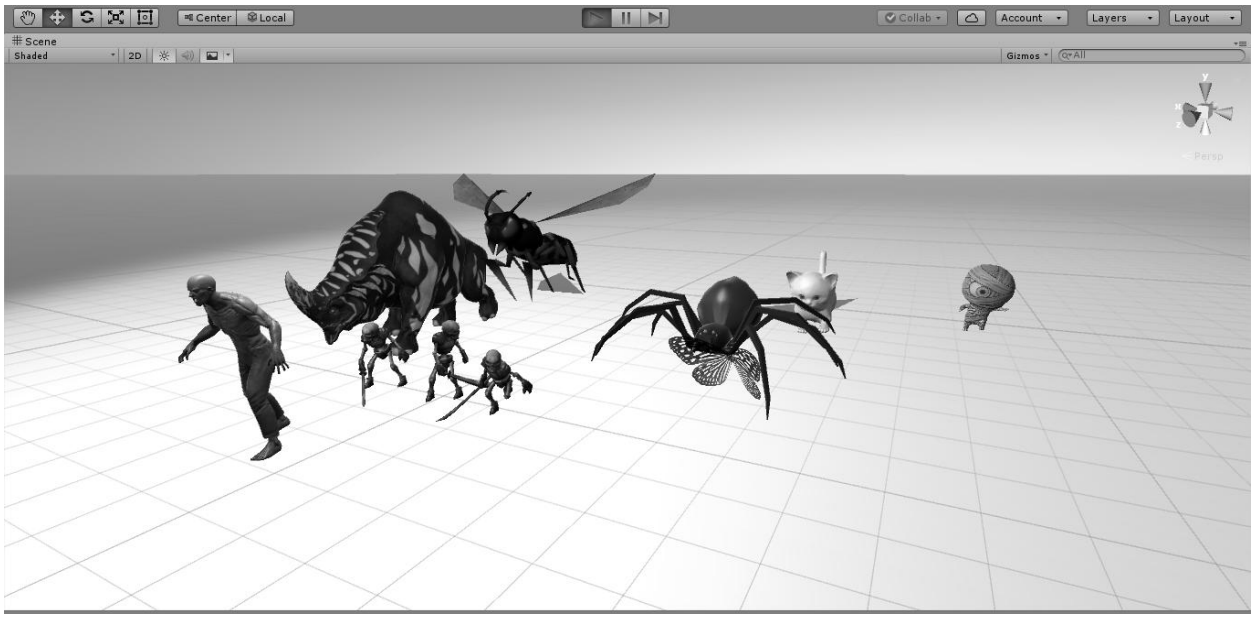
Slika 6.21. Automatska stanja animacije

7. Implementacija modela

U ovoj fazi razvijanja igre stvari se privode kraju. Ogromna vrijednost Unityja kao okružja za razvijanje je njegova velika zajednica ljudi koja ga koristi i veliki broj modela, tekstura i svega ostalog što može biti korisno pri razvijanju. Stranice poput *UnityAssetStore* su vrlo popularne u svijetu. Trgovina imovine (eng. *AssetStore)* je također povezan sa samim programom Unity te se direktno iz programa može skidati potrebna imovina. Postoje razne kategorije pretraživanja kao i rang cijene koju si razvijač može priuštiti. Nakon odabira željenog proizvoda korisnik jednostavnim klikom na *Download* i *Import* ubacuje proizvod unutar svog projekta. U ovom projektu odabrano je sedam modela koji će služiti kao modeli neprijatelja. Modeli dolaze sa svojim animacijama i ovaj princip je ogromna ušteda vremena.

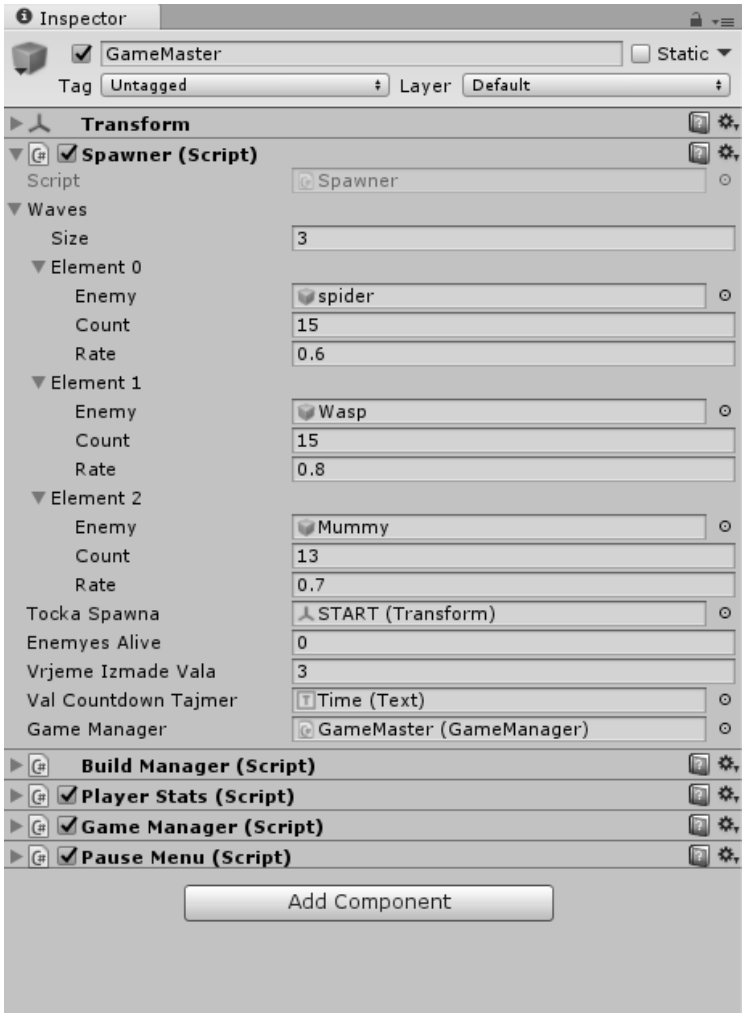
Slika 7.1. Asset Store unutar Unity programskog okružja

Zahvaljujući fazi prototipiranja sve je spremno za završnu implementaciju. Modeli se dorađuju u smislu dodavanja skripta, sudarača i oznaka potrebnih kako bi tornjevi mogli ciljati neprijatelje.



Slika 7.2. Testiranje skinutih modela i animacija

Također se neprijateljima dodaje animacija smrti koja će biti puštena nakon što se život neprijatelja spusti ispod ili bude jednak nuli. U ovom slučaju tornjevi uništavaju sam objekt neprijatelja te neprijatelj stvara novi objekt sebe koji je čist u smislu da nema nikakve skripte i sudarače na sebi te ima samo jednu animaciju, animaciju smrti. Nakon toga objekt se u kratkom roku uništava. Unutar *Spawner* objekta određuje se koji će se neprijatelji stvarati i kojom brzinom.



Slika 7.3. Završni izgled GameMaster objekta

Slika 7.3. prikazuje završni izgled *GameMaster* komponente koje je u konačnici zadužena za stvaranje raznolikih neprijatelja s međusobnim vremenskim razmakom, skriptom *BuildManager* koja je zadužena za procesiranje označavanja tornja za izgradnju, njegovom cijenom i pametnom kopijom modela, *PlayerStats* skriptom zaduženom za definiranja karakteristike igrača, početnom količinom novaca i života, *GameManager* skriptom zaduženom za praćenje i postavljanje stanja pobjede i gubitka igre te *PauseMenu* skriptom zaduženom za pauziranje igre, zaleđivanje vremene unutar igre koja je u procesu igranja, nudeći opcije odlaska na početni izbornik ili nastavka igre.

8. Zaključak

U ovom radu pod nazivom *Razvoj igre unutar Unity okruženja* objašnjen je proces izrade igre žanra obrane tornja, od prototipiranja mehanike igre do završnih implementacija modela. U radu je objašnjen tok rada, razvoj igrivog prototipa, implementacija logike i mehanike, modeliranje tornjeva i animacija korisničkog sučelja.

Pristupačnost informacijama putem otvorene mreže širi znatiželju i zaigranost ljudi. Dio ljudske rutine postaje gledanje u neku vrstu ekrana tražeći zabavu i izbjegavanje dosade. Kako bi zadovoljili masu, veliki broj malih studija počinju s razvojem igara za velik broj platformi. Za razliku od prošlosti, gdje je bila potrebna neka vrsta konzole za igranje, ili kućni kompjuter, sada to više nije faktor.

Razvijaju se igre za mobilne telefone i tablete, koji su u većini vremena vlasniku na dohvat ruke. Snaga samih uređaja je sve jača što znači da mogu pokretati sve zahtjevnije igre. Razvoj igara se može pokazati kao profitabilni posao ukoliko se sadržaj i igrivost svidi ciljanoj publici koja je u pravilu opširna. Igre služe kako bi okupirale korisnika i na neko vrijeme ih odvukle od svakodnevnih problema, tako da korisnik unutar igre traži neku svrhu, neki cilj, neko putovanje od A do B i nagradu za to što je uspio postići neki rezultat.

Razvoj igara može biti zabavan, kompleksan, frustrirajući ali i obrazovni proces u kojem krajnji proizvod može biti nagrađen. Zahvaljujući brojnim postojećim igrama i mehanikama inspiracije za stvaranje ne fali. Uvijek će postojati neko ko će pomisliti "Bilo bi bolje da je ovako" i zahvaljujući razvitkom tehnologija isto kao i globalizacije informacija moguće je napraviti to nešto bolje i zabavnije. Broj alata koji se pruža razvijačima raste a s time rastu i mogućnosti stvaranja vizualno privlačnijih igara i aplikacija koje ne zahtijevaju veliku performansu operativnog sistema. Velike baze podataka, isto kao i ogromne zajednice u kojima ljudi dijele svoje probleme, mišljenja i savjete su od velike pomoći, ne samo pri ulasku u vode razvijanja, nego i općenito u rješavanju svakodnevnih problema, jer ako se naiđe na neki problem, velika je mogućnost da je neko već naišao na njega i riješio ga uz pomoć zajednice. Zahvaljujući postojećim alatima cijeli razvoj igre se može odraditi uz pomoć samo par programa.

U Varaždinu, datum 10.3.2017 Potpis

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

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

Ja, DARKO GOTAL *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom RAZVOJ IGRE UVOĐAK UNITY OKRUŽENJA *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

DARKO GOTAL
(vlastoručni potpis)

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

Ja, DARKO GOTAL *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom RAZVOJ IGRE UVOĐAK UNITY OKRUŽENJA *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

DARKO GOTAL
(vlastoručni potpis)

9. Literatura

[1] Unity Documentation, Working In Unity, dostupno na docs.unity3d.com/Manual/UnityOverview.html

[2] Microsoft Nevelopment Network, Welcome to Visual Studio 2015, dostupno na www.msdn.microsoft.com/en-us/library/dd831853.aspx

[3] Autodesk, Maya LT, dostupno na www.autodesk.com/products/maya-lt/features/all

[4] Tim Chamillard, Beginning Game Programming with C#, www.coursera.org/learn/game-programming

[5] Unity Documentation, Navigation and Pathfinding, dostupno na docs.unity3d.com/Manual/Navigation.html

[6] Unity Documentation, Object.Instantiate, dostupno na docs.unity3d.com/ScriptReference/Object.Instantiate.html

[7] Unity Documentation, Time.deltaTime, dostupno na docs.unity3d.com/ScriptReference/Time-deltaTime.html

[8] Unity Documentation, MonoBehaviour.Update(), dostupno na docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html

[9] Unity, Statics, dostupno na www.unity3d.com/learn/tutorials/topics/scripting/statics

[10] Unity Documentation, Input, dostupno na docs.unity3d.com/ScriptReference/Input.html

[11] Richard Williams, The Animator's Survival Kit, dostupno na www.theanimatorssurvivalkit.com

[12] Nic Simmonds, Tower Defense – A brief History, dostupno na www.mygaming.co.za/news/features/6341-tower-defense-a-brief-history.html

[13] Mark Masters, Understanding the 12 Principles of Animation, dostupno na blog.digitaltutors.com/understanding-12-principles-animation

[14] James Aberdeen, Animation Principles, dostupno na jaberdeen.blogspot.hr/p/animation-principles.html

Popis slika

Popis koda