# Planiranje, modeliranje i izrada računalne igre

**Jurić, Antonio**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Antonio Jurić**

# PLANIRANJE, MODELIRANJE I IZRADA RAČUNALNE IGRE

**ZAVRŠNI RAD**

**Varaždin, 2022.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Antonio Jurić**

**Matični broj: 0016136577**

**Studij: Informacijski sustavi**

# PLANIRANJE, MODELIRANJE I IZRADA RAČUNALNE IGRE

**ZAVRŠNI RAD**

**Mentor:**

Izv. prof. dr. sc. Mario Konecki

**Varaždin, 2022.**

Antonio Jurić

**Izjava o izvornosti**

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

# Sadržaj

# Sažetak

Računalne igre su danas vrlo rasprostranjen, širok i opće poznat pojam. Njihov razvoj i korištenje krenulo je rasti proporcionalno rastućem eksponencijalnom trendu razvoja računala, mikroprocesora, grafičkih kartica i ostale računalne opreme. Kako vrijeme ide dalje, računalne komponente se unaprjeđuju i time otvaraju nove mogućnosti svijetu igara. Od nekadašnjih arkadnih igrica, prvih konzola i 3D sustava, došli smo do kompetitivnih igrica za više igrača, virtualne stvarnosti te višeplatformskih igrica i mnogih drugih. Računalne igre se danas koriste u svrhu zabave, edukacije, natjecanja i slično, a kroz ovaj rad ću prikazati proces nastanka jedne takve igre, koji alati se najčešće koriste, koje metode su najbolje za odgovarajuće situacije i o čemu treba najviše voditi računa. Svi ti koraci bit će prikazani na stvarnom primjeru igre „Platform Escape“ koju sam radio po istim principima.

***Ključne riječi:*** *planiranje igre; modeliranje; programiranje; računalna igra; proces izrade računalne igre*

# 1. Uvod

Razvoj računalne igre, neovisno o složenosti, veličini i kategoriji, vrlo je detaljan proces koji se sastoji od tri glavna dijela, a to su planiranje, modeliranje i izrada ili programiranje. Svi ti glavni dijelovi se dalje mogu podijeliti na razne potkategorije poput osmišljanja priče, animiranja, testiranja i slično, ali nisu nužno potrebne prilikom izrade svake igrice. Kroz ovaj rad ću se fokusirati na glavne dijelove te ih detaljno objasniti putem razvoja vlastite 3D igrice.

„Platform Escape“ je praktičan primjer jedne 3D računalne igre koja bi pripadala kombinaciji „RPG“ (skraćeno „role play game“), „Platform“ i „Endless“ kategorije. Značajke su da igrač ima vlastitu ulogu s jedinstvenim karakteristikama koje ovise o odabranoj ulozi te pokušava riješiti probleme dane platforme, bilo to pobijediti čudovišta, izbjeći zamke ili pronaći izlaz. Svakim uspješnim rješavanjem zadataka platforme, dozvoljava mu se prolaz na iduću platformu s novim izazovima koji s vremenom postaju sve teži. Igrač gubi onog trena kada mu ponestaje života ili padne s platforme te mu se bilježi ostvareni rezultat. Detaljnije informacije o samoj igri nalaze se pod poglavljem „Planiranje računalne igre“.

Nakon što je igra osmišljena slijede procesi modeliranja i izrade. Na temelju vlastitog dosadašnjeg iskustva shvatio sam da ti procesi nisu nužno linearni te je moguće prvo programirati igricu i naknadno modelirati dijelove ili programirati igricu s već gotovim modelima, sve ovisi o vlastitoj praksi. Na konkretnom primjeru moje igrice, modele sam izrađivao sinkrono s programiranjem, odnosno model sam izrađivao tek u onda kada mi je bio potreban unutar igre te ću cijeli postupak i način rada opisati pod poglavljima „Modeliranje računalne igre“ i „Izrada računalne igre“.

Čak i nakon izrađene računalne igre, bilo da je ona ostvarila sve ili samo neke ciljeve zacrtane prilikom procesa planiranja, može se i poželjno je da se dalje usavršava i nadograđuje. Prilikom prvog izbacivanje gotove igre, korisnici prijavljuju svoja iskustva, molbe i greške unutar igre koje razvojni programer nije uočio. Programer tada radi na ispravljanju prijavljenih grešaka te prema molbama ili vlastitim željama unaprjeđuje igricu dodajući joj nove mogućnosti ili sadržaj.

# 2. Planiranje računalne igre

Planiranje računalne igre neupitno dolazi na prvo mjesto prije samog modeliranja ili programiranja jer je važno znati vrstu igre, sustav ili svijet u kojem se nalazi, dijelove igre koji će se modelirati, atmosferu i slično. Nemoguće je modelirati objekt igre bez da znamo o kojem se objektu radi te je isto tako nemoguće programirati taj objekt bez znanja o njegovoj mehanici, okruženju ili cilju .

Prije svega bitno je odrediti o kojem će se tipu igre raditi kako bi znali koji su nam alati potrebni za izradu i modeliranje jer različiti su za 2D, 3D, VR igre i slično. Zatim je potrebno odabrati generalni žanr igre koji specificira mehaniku, detalje, cilj i osnovne komponente igre. Iako žanr sam po sebi definira vrstu igre, njega koristimo više kao smjer u kojemu igrica treba težiti te mu pridodajemo vlastitu mehaniku i svojstva koja će igricu učiniti što originalnijom. Nakon toga dolazimo do samog igrača, način njegovog prikaza unutar same igre te njegov pogled na svijet koji ga okružuje. Taj se igrač zatim smješta unutar dobro osmišljene priče igre koja prati igrača tijekom igranja i upotpunjuje igricu nekim kontekstom te daje igraču neki smisao i cilj. Uz kvalitetno priču igre lako je doći do informacija o ostalim ključnim elementima igre, poput neprijatelja ako ih ima, zamki ili prepreka, slagalica ili zagonetki i slično. Na kraju je poželjno odrediti neki način skaliranja same igrice tijekom vremena kako bi se osiguralo konstantno izazovno okruženje koje forsira igrača na napredak.

Kombinacijom svih navedenih elemenata dobivamo okvirnu ideju od čega krenuti prilikom izrade i na koje se elemente fokusirati tijekom izrade kako bi ostvarili zamišljeni cilj. Bitno je naglasiti da je ta ideja okvirna jer se često tek duboko u procesu razvoja igre odlučimo na neke promjene koje se ipak čine bolje i izvedivije u odnosu na prvobitne. U sljedećim potpoglavljima ću proći kroz elemente koje sam ja koristio prilikom izrade svoje igrice, zašto sam za svoju igru odabrao specifičnu vrstu elementa te koja je bila moja prvobitna ideja priče s kojom sam krenuo u izradu i modeliranje.

## 2.1. Tip računalne igre

Kao što je naglašeno prilikom uvoda u poglavlje, postoji više tipova računalnih igara, a bitno je istaknuti 2D, 3D i VR („Virtual Reality“). Iako bi VR pripadao 3D tipu računalne igre, njegova primjena se uvelike razlikuje pa sam ga zbog toga izdvojio kao zaseban tip.

Na samom početku razvoja računalnih igara postojali su samo 2D tipovi te je zbog toga taj tip i najrašireniji, ali pojavom treće dimenzije u svijetu igara cijela perspektiva se promijenila te se i dan danas vodi debata o tome koji je tip „bolji“. VR je trenutno najnoviji tip u velikom usponu, ali se i dalje razvija te se očekuju veliki projekti u bliskoj budućnosti koji bi mogli ponovno preokrenuti pogled i stvoriti novi pristup računalnim igrama.

Prilikom planiranja računalne igre važno je to uzeti u obzir jer ovisno o odabiru specifičnog tipa igre, odlučujemo raspolagati s različitim razvojnim okruženjem, kako modeliranja, tako i programiranja. Stoga ćemo kod 2D modeliranja imati na raspolaganju alate poput Adobe Photoshop-a, GIMP-a i sličnih te dvodimenzionalan Game Engine s prikladnim programskim jezicima. Kod 3D modeliranja to će biti „Blender“, „Autodesk Maya“ ili slično te za programiranje trodimenzionalan Game Engine sa svojim programskim jezicima. VR kao podvrsta 3D-a koristi iste alate s dodatnim modifikacijama unutar samog Game Engine-a kako bi prikazao taj dodatni nivo virtualnog svijeta. Na sljedećoj slici jasno je prikazana razlika između modela igrača 2D i 3D računalne igre.


*Slika 1: 2D vs 3D Animation*

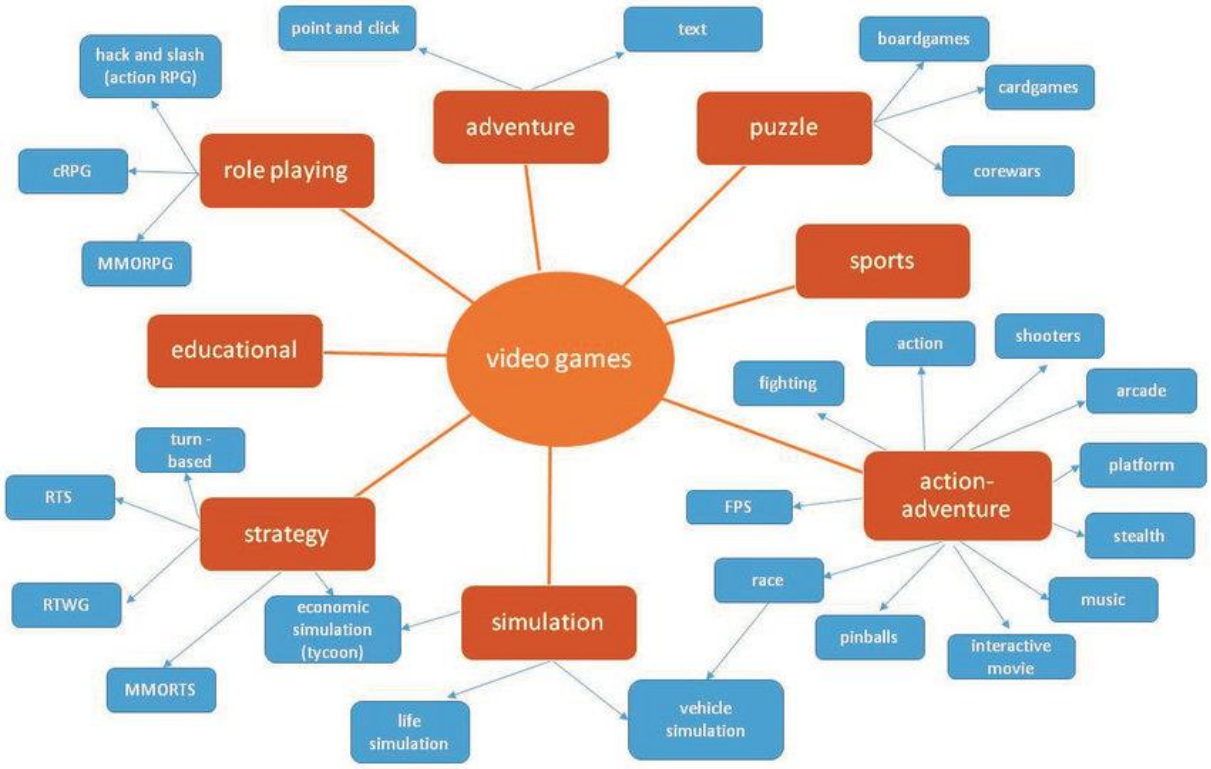
*Izvor: https://www.capermint.com/blog/difference-between-2d-and-3d-animation/*

Za svoju računalnu igru „Platform Escape“ odlučio sam odabrati 3D tip te se osloniti na „Autodesk Maya“ kao glavni alat za modeliranje te „Unity Engine“ kao glavni Game Engine za razvoj i programiranje. Odabrao sam ovaj tip jer je trenutno najpopularniji na tržištu, iako je modeliranje i programiranje dodatno otežano s obzirom na 2D upravo zbog te dodatne treće dimenzije.

## 2.2. Žanr računalne igre

Nakon što smo odredili tip naše igre, sljedeći korak je odrediti žanr igre. Danas u svijetu računalnih igara postoji veliki broj različitih žanrova, a svaki od njih nam daje neki generalnu sliku o tome kako bi igrica trebala izgledati, koje mehanike se najčešće koriste unutar određenog žanra, kako izgledaju modeli i slično. Neki žanrovi direktno ovise i o tipu računalne igre tako da i odabirom tipa igre sužavamo broj mogućih žanrova za biranje.

Kako bismo se odlučili za najprikladniji žanr prije svega valjalo bi pogledati neke od najopćenitijih žanrova koji nam stoje na izboru. Na sljedećoj slici prikazana je podjela žanrova računalnih igara na nekoliko općih, a to su: „RPG“ („role playing game“), „Adventure“, „Puzzle“, „Sports“, „Action-Adventure“, „Simulation“, „Strategy“ i „Educational“.




*Slika 2: Variety of video game genres*

*Izvor: https://www.researchgate.net/figure/Variety-of-video-game-genres-an-attempt-to-classify-genres-and-kinds-source-authors_fig2_292128935*

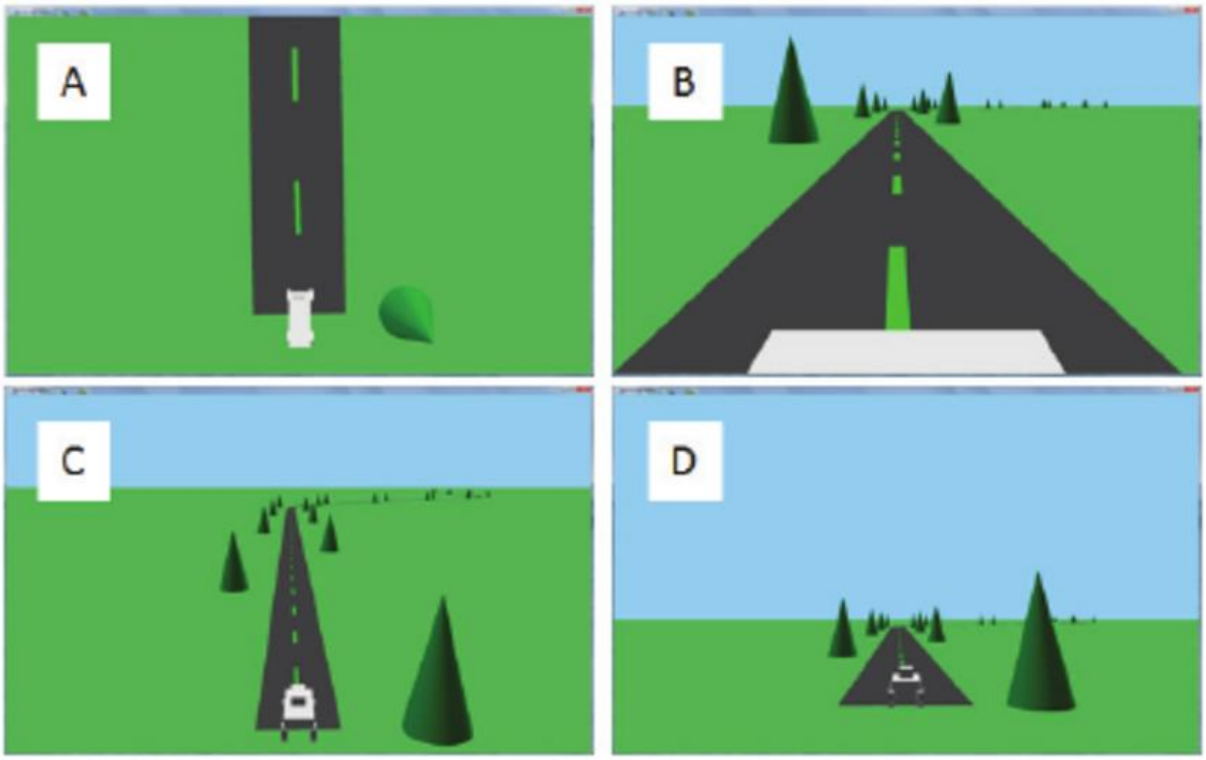
Ovdje bi napomenuo kako je prethodni graf samo jedan od mogućih načina podjela računalne igre prema žanrovima te ne definira neku striktnu strukturu jer npr. žanr „Adventure“ može, ali ne mora uključivati i dijelove „Puzzle“ žanra ili „RPG“ žanra. Graf nam više služi kako bi generalno odredili jedan glavni žanr kojemu težimo i koji određuje glavnu mehaniku igre. Svaki od žanrova na grafu ima i dodatnu podjelu na „podžanrove“ ovisno o konkretnom tipu pojedinog glavnog žanra. Prije svega ukratko ću objasniti glavnu podjelu te se nakon toga izjasniti koji sam žanr odabrao kao glavni, koje mehanike on uključuje te koje elemente ostalih žanrova planiram također koristiti.

„Adventure“ žanr sam po sebi govori da je riječ o igrici koja ima opsežnu i dobro razrađenu priču koja bi bila i glavni element ovog žanra, a kretanje pričom može biti odrađeno jednostavnim mehanikama poput običnog odabira između ponuđenih izbora ili klikanjem po ekranu. „Puzzle“ žanr određuje igru kao neku vrstu rješavanja problema, to mogu biti zagonetke, društvene igre ili slično. „Sports“ žanr obuhvaća igre s elementima sporta, npr. nogomet, košarka, skijanje ili više njih zajedno. „Action-Adventure“ je žanr koji kao i „Adventure“ ima dobro razrađenu priču, ali ja naglasak više na interakciji igrača s tom pričom koja je u ovom slučaju mnogo življa i intenzivnija. „Simulation“ predstavlja žanr koji simulira neke od aktivnosti stvarnog svijeta implementiranih unutar računalne igre gdje je glavna ideja što vjernije i točnije prikazati tu aktivnost kao npr. utrke, upravljanje zrakoplovom i slično. „Strategy“ žanr fokusira se na vrstu igre za koju je potrebno mnogo strateškog razmišljanja te igrač svojim odlukama određuje nastavak igre, bilo da se radi o osvajanju svijeta, pomorskim bitkama ili sličnim vrlo složenim scenarijima. „RPG“ žanr predstavlja igru u kojoj igrač na početku bira jednu od uloga te se fokusira na napredovanju odabrane uloge kako bi riješio različite vrste problema na putu do cilja. „Educational“ žanr obuhvaća neki od prethodno navedenih žanrova, no kao glavni cilj ima neku vrstu edukacije igrača.

Sada nakon što sam svojim riječima ukratko objasnio glavnu podjelu, mogu potvrditi da sam za svoju igru „Platform Escape“ odabrao „RPG“ žanr kao glavni. Time određujem da će igrač na samome početku odabrati jednu od uloga sa svojim karakteristikama i vještinama te će tijekom igranja imati priliku nadograđivati iste u svrhu lakšeg nošenja s nadolazećim preprekama. Kao sporedan žanr iz kojeg sam uzeo neke elemente i mehanike naveo bi „Action-Adventure“ tj. konkretno „Platform“ žanr budući da se igrač bori s preprekama raznovrsnih nadolazećih platformi.

## 2.3. Igrač, ideja i priča igre

Tip i žanr igre određuju mehaniku i alate, no potrebno je još odrediti igrača tj. njegov pogled na svijet oko njega te stavku koja definira originalnost svake igre, a to je njena priča. U prethodnom potpoglavlju odabrao sam „RPG“ kao glavni žanr igre te iz potpoglavlja prije znamo da se radi o 3D tipu igre, ali ni jedno ni drugo ne govori o načinu pogleda tj. odnosu same kamere na igrača i svijet oko njega. Sljedeća slika prikazuje nekoliko vrsta mogućih pogleda.




*Slika 3: The four views*

*Izvor: https://www.researchgate.net/figure/The-four-views-A-overhead-B-first-person-C-third-person-high-D-third-person-low_fig1_221474898*

Na slici je prikazan pogled iz prvog lica (B) i pogledi iz trećeg lica s različite visine i rotacije. Za konkretno svoju igru odlučio sam se na pogled iz treće perspektive, nešto slično pogledu „C“ sa slike. Odabrao sam taj pogled kako bi igrač vidio samoga sebe, ali i svijet oko njega što je klasično za većinu „RPG“ igara.

Sada kada smo odredili sve bitne elemente igre, možemo započeti s osmišljanjem priče. Ovisno o žanru same igre, potrebna je više ili manje detaljna priča te kod nekih nije potrebna uopće. Primjer za to bi bile kartaške igre ili simulacija vožnje broda koje same po sebi govore sve što je potrebno kako bi igrač znao o čemu se radi i koji mu je cilj. Budući da sam se odlučio za „RPG“ žanr, potrebno je ipak dati neki kontekst igri kako bi bila originalnija i zanimljivija te kako bi pripomogla u osmišljanju nekih od mehanika i scenarija prilikom programiranja.

Kao pozadinsku priču postavio sam glavnog protagonista unutar sklopa platformi dizajniranim od strane moćne ilegalne korporacije koja se primarno bavi genetskim inženjeringom i virtualnom stvarnošću. Protagonist je jedan od subjekata testiranja i postavljen je u sustav platformi kako bi znanstvenicima korporacije omogućio istraživanje njegovih novih sposobnosti. Budući da znanstvenici žele otkriti krajnje granice izdržljivosti svojih testnih subjekata, nije im u cilju da subjekt preživi već da izvuku što više informacija kako bi sljedeći pokus bio što uspješniji. Upravo zbog tog ekstremnog rada i istraživanja, koriste se različite atmosfere, genetski modificirana čudovišta i testne prepreke od strane korporacije. Protagonist je jedan od nesretno odabranih slučajnih prolaznika kojega su protiv svoje volje uvrstili u spomenuti program testiranja te mu je sada jedina opcija preživjeti nadolazeće platforme dok ne smisli na koji način pobijediti sustav te kako pronaći izlaz.

Ideja je da igrač nakon prvobitnog postavljanja na platformu rješava nadolazeće prepreke, prolazi kroz sustav platformi i bori se s različitim vrstama izazova koji budu bačeni pred njega. Sustav nadolazećih platformi kompletno je nasumičan te svakih deset platformi igrač ima priliku odahnuti na glavnoj platformi. Cilj je da s vremenom platforme postaju sve teže i teže te igrač kako bi se nosio s njima mora konstantno nadograđivati svoje vještine i karakteristike. Iako se u priči spominje pokušaj pobjede sustava i izlazak iz kompleksa, ideja je da igra bude „Endless“, odnosno da testira krajnje granice i sposobnosti igrača dok ne „umre“, ali upravo nasumičnost i različite karakteristike igrača osiguravaju jedinstveno iskustvo prilikom svakog novog pokušaja igranja.

Ovime završavamo poglavlje planiranja računalne igre i krećemo dalje u realizaciju zacrtanih ideja i ciljeva.

# 3. Modeliranje računalne igre

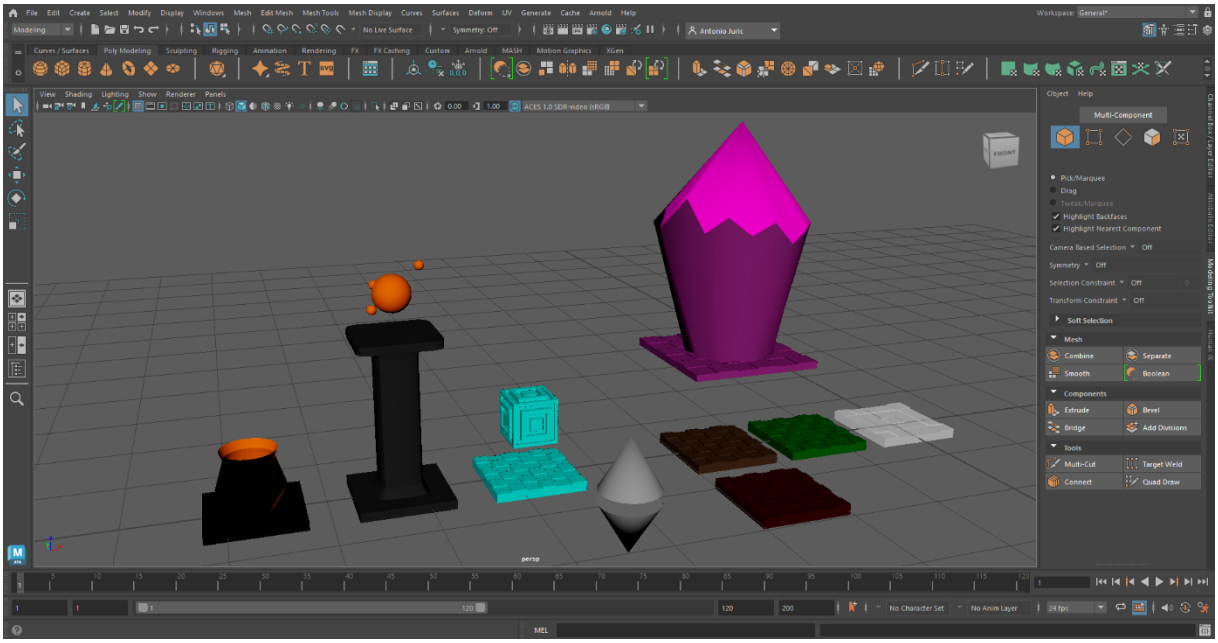
Kako bi računalna igra bila ugodnija oku i time privukla veću pozornost tržišta velike računalne kompanije većinu svog vremena i novaca troše upravo na procese modeliranja i animiranja. Primjer toga su „FPS“ („First Person Shooter“) igre čiji vodeći naslovi poput „Call of Duty“ serijala već godinama imaju vrlo sličnu mehaniku te nema nekih velikih promjena što se samog dijela programiranja tiče, ali kako s vremenom napreduju grafičke sposobnosti računala tako modeli unutar igrice postaju sve realističniji i detaljniji. Činjenica je da korisnici igara procjenjuju igru prema onome što vide pa će tako i bolje modelirana igrica privući i veću publiku nego igrica s dobrom mehanikom i idejom, ali vrlo općenitim modelima i detaljima. Upravo zbog toga modeliranje je ključan dio svake računalne igre pa sam tako i u ovom radu odvojio zasebno poglavlje kako bi objasnio neke od alata i tehniku modeliranja.

U prethodnom poglavlju odredio sam kompletnu ideju igre te s obzirom na to da se radi o 3D „RPG“ igrici, znači da će alati za modeliranje biti također 3D orijentirani. Neki od trenutno najaktualnijih alata za 3D modeliranje su „Autodesk Maya“, „Blender“, „Autodesk 3ds Max“, „Cinema 4D“ i mnogi drugi. Prilikom odabira prikladnog alata za modeliranje moje igre gledao sam više kriterija, poput cijene, zahtjevnosti i korisničkog sučelja te se na kraju odlučio upravo za „Autodesk Maya 2023“ alat.

Ovo sam poglavlje podijelio na dva potpoglavlja kako bi pokrio korisničko sučelje i neke od osnovnih funkcionalnosti „Maya“ alata te način modeliranja jednog od konkretnih modela koje ću implementirati unutar svoje igre.

## 3.1. „Autodesk Maya 2023“

Kao što sam spomenuo unutar uvoda u poglavlje, „Maya“ je jedan od najaktualnijih alata za 3D modeliranje u svijetu, bilo da se koristi za izradu animacija i modela računalne igre, modela za 3D printanje ili kompletnih animiranih filmova. Najveći problem alata je njegova cijena, no postoje besplatne opcije edukacijskog perioda od jedne godine koji sam iskoristio kako bi naučio i izradio modele potrebne unutar moje igrice.



*Slika 4: Korisničko sučelje „Autodesk Maya 2023“*

*Izvor: autor*

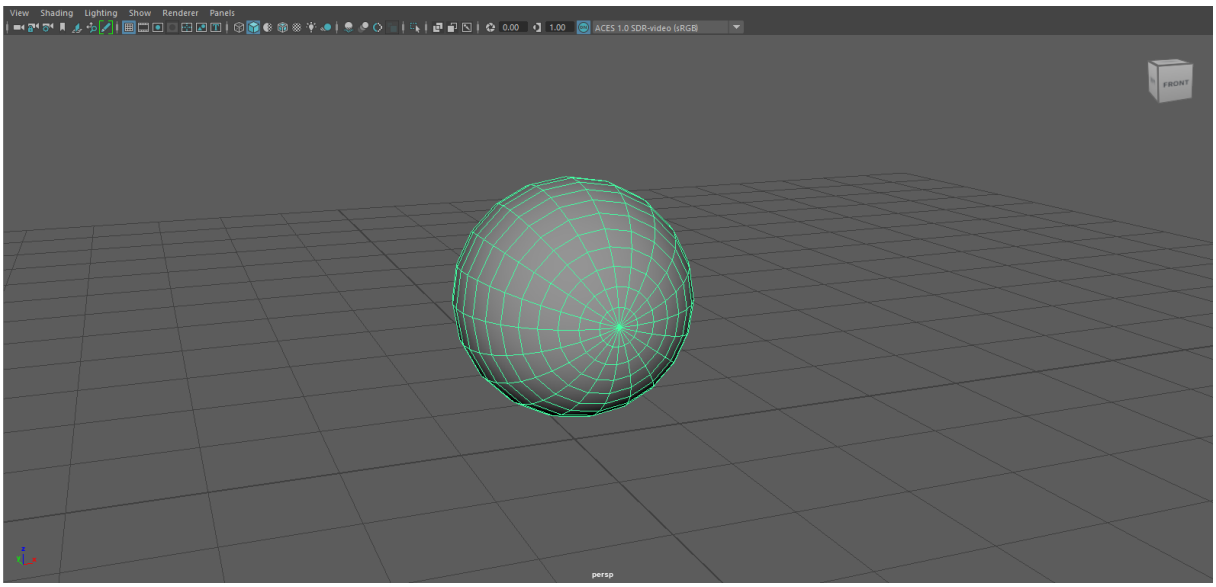
Prethodna slika prikazuje korisničko sučelje alata unutar kojeg se već nalaze neki od modela koje sam završio i implementirao u igrici. Najveći prozor u sredini je glavni pogled na modele koji se trenutno modeliraju i koji se nalaze unutar „Maya“ projekta. Lijevo od prozora nalazi se traka s alatima za selektiranje, pomicanje, rotiranje i skaliranje modela te različite opcije prikaza glavnog prozora. Donja traka se sastoji od različitih pomoćnih alata za animiranje modela te statusna traka s povratnim informacijama nakon izvršavanja određenih akcija unutar alata. Desna traka sadrži razne alate koji služe za detaljno određivanje i uređivanje pojedinih detalja modela. Na njoj postoji odabir između sekcije alata za modeliranje, alata za uređivanje likova, raznih postavki trenutno korištenih alata i slično. Najbitnija sekcija su upravo alati za modeliranje gdje imamo mogućnost odabrati i uređivati modele u cjelini, vrhove, rubove ili lica modela. Pomicanjem, rotacijom i skaliranjem tih dijelova zapravo se modelira početni oblik koji može biti kocka, sfera ili nešto slično i od tog se oblika transformacijom dobiva željeni oblik modela koji koristimo unutar igre. Zadnja, gornja alatna traka sastoji se od velikog broja gotovih komponenti krivulja, modela, „kiparskih“ alata te alata za animiranje, pregledavanje i obrađivanje.

Mnoge od dostupnih alata nisam ni koristio prilikom izrade svojih modela te svaki od postojećih alata zahtjeva praksu i vještinu koja se dobije tek nakon velikog broja sati provedenih koristeći upravo ovaj program. Zbog njegove kompleksnosti, „Autodesk Maya“ savršen je za izradu virtualno svih modela koji se mogu zamisliti, a granica je samo razina znanja i kreativnosti osobe koja ga koristi.

## 3.2. Proces izrade modela za igru

U ovom potpoglavlju prikazat ću kompletan proces kreiranja jednog od modela kojega planiram koristiti unutar svoje igre. Ideja mi je modelirati oblik duha, jednog od neprijatelja glavnog igrača.

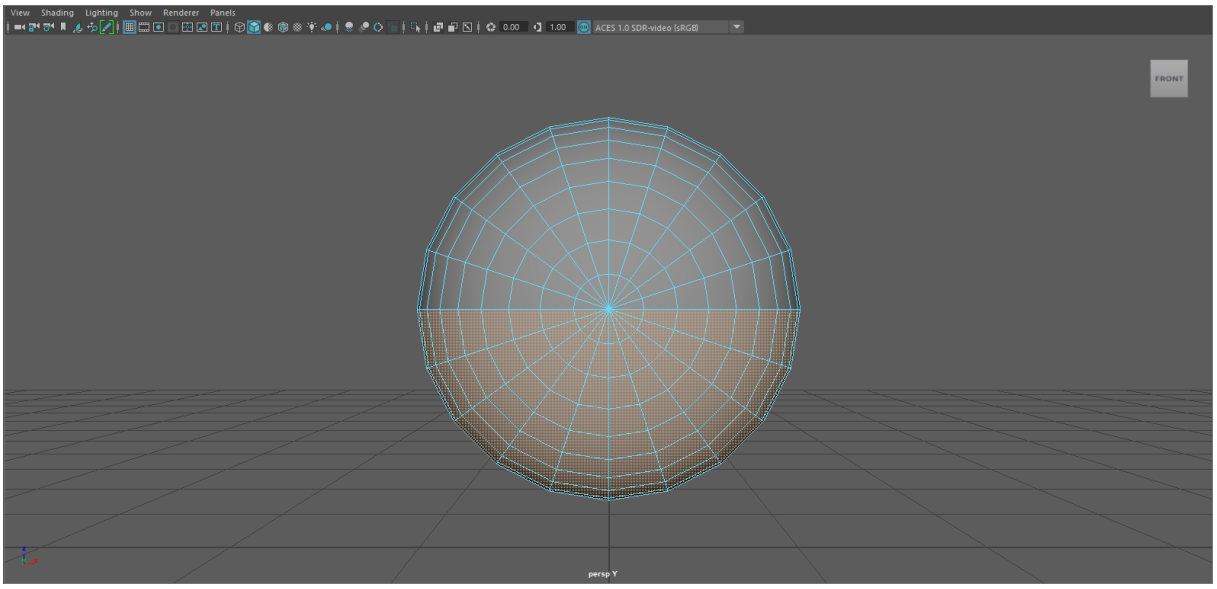
Cijeli postupak započinje pokretanjem novog „Maya“ projekta te biranjem između ponuđenih predložaka početnih modela. Ovisno o modelu kojeg planiramo napraviti, biramo najprikladniji predložak poput kocke, sfere, cilindra, poligona ili slično. Budući da se radi o klasičnom obliku duha koji je napola zaobljen i napola produžen, odlučio sam kao početni model koristiti sferu što je i prikazano na slici ispod.




*Slika 5: Izrada modela duha – 1. korak*

*Izvor: autor*

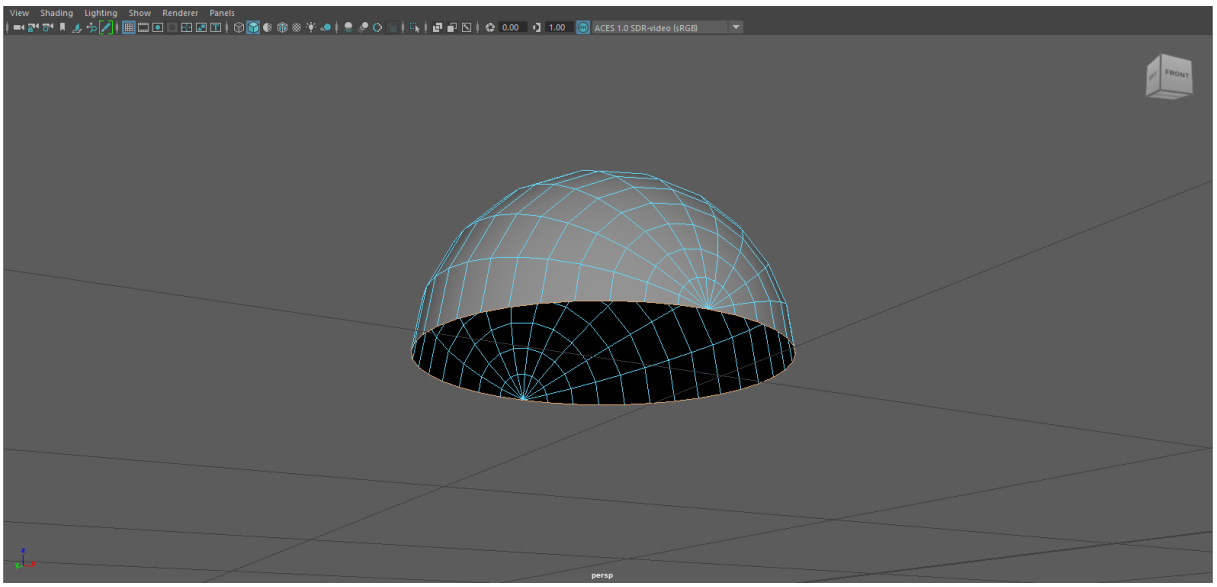
Selektiranjem sfere kao početnog modela ona se postavlja na sredinu prostora unutar glavnog prozora. S obzirom na to da je duh samo napola zaobljen s gornje strane, idući korak bio mi je obrisati donju polovicu sfere. Kako bi to učinio morao sam se prebaciti iz objektnog načina modeliranja, koji služi za modeliranje objekta kao cjeline, u alate za modeliranje lica objekta. Zatim sam u pogledu sprijeda označio sva lica koja sam planirao odstraniti kao što je prikazano na sljedećoj slici.




*Slika 6: Izrada modela duha – 2. korak*

*Izvor: autor*

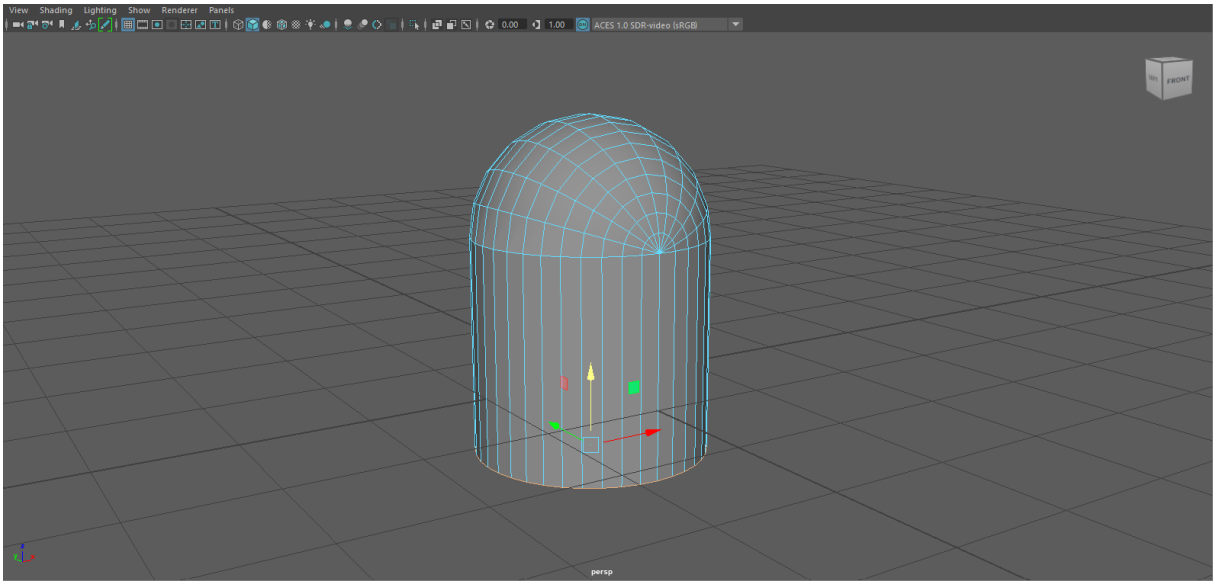
Selektiranjem donjeg dijela sfere označio sam sva vidljiva lica, ali i ona lica koja se nalaze s druge strane te ih u idućem koraku brišem jednostavnim pritiskom tipke „delete“.




*Slika 7: Izrada modela duha – 3. korak*

*Izvor: autor*

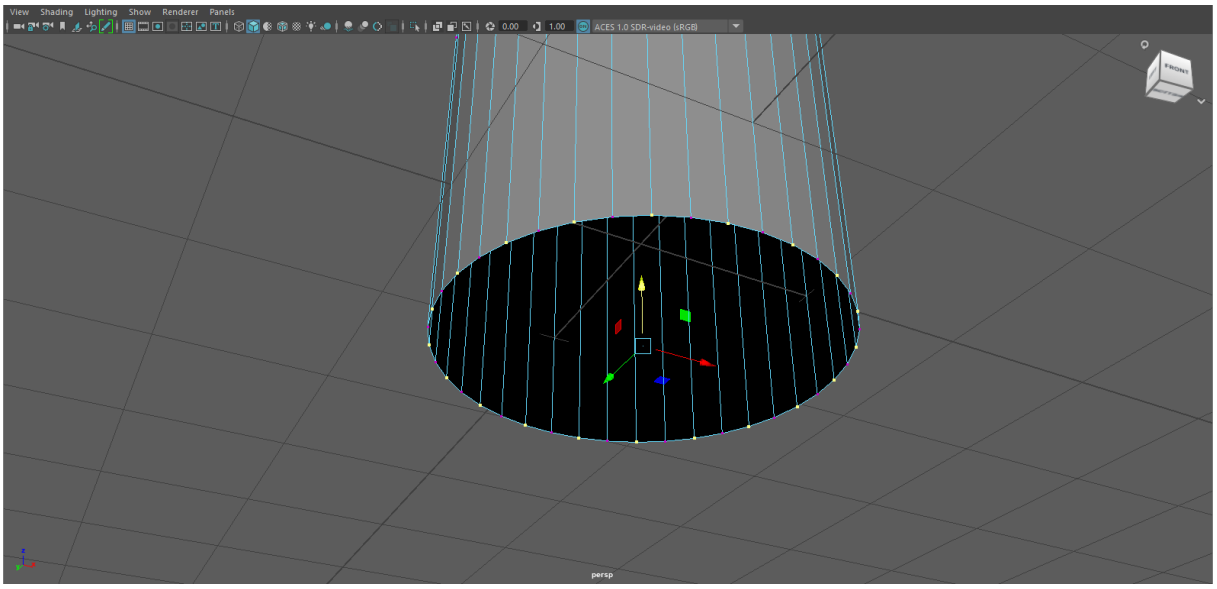
Brisanjem lica dobio sam „polu-sferu“ koju je sada trebalo produžiti od linije presjeka prema dolje kako bi stvorio tijelo duha. Prebacio sam s alata za modeliranje lica na alat za modeliranje linija i rubova modela jer je donji rub upravo ono što sam trebao povući prema dolje. Selektirao sam cijelu donju kružnicu, no kako bi ju okomito izdužio prema dolje nije ju bilo dovoljno samo pomaknuti jer bi zbog neravne površine sfere dobio tijelo u obliku piramide. Zbog toga sam koristio alat „Extrude“ kojim selektiranu kružnicu izdužujem i povlačim dok naposljetku ne dobijem željenu visinu tijela kao što se vidi na slici ispod.




*Slika 8: Izrada modela duha – 4. korak*

*Izvor: autor*

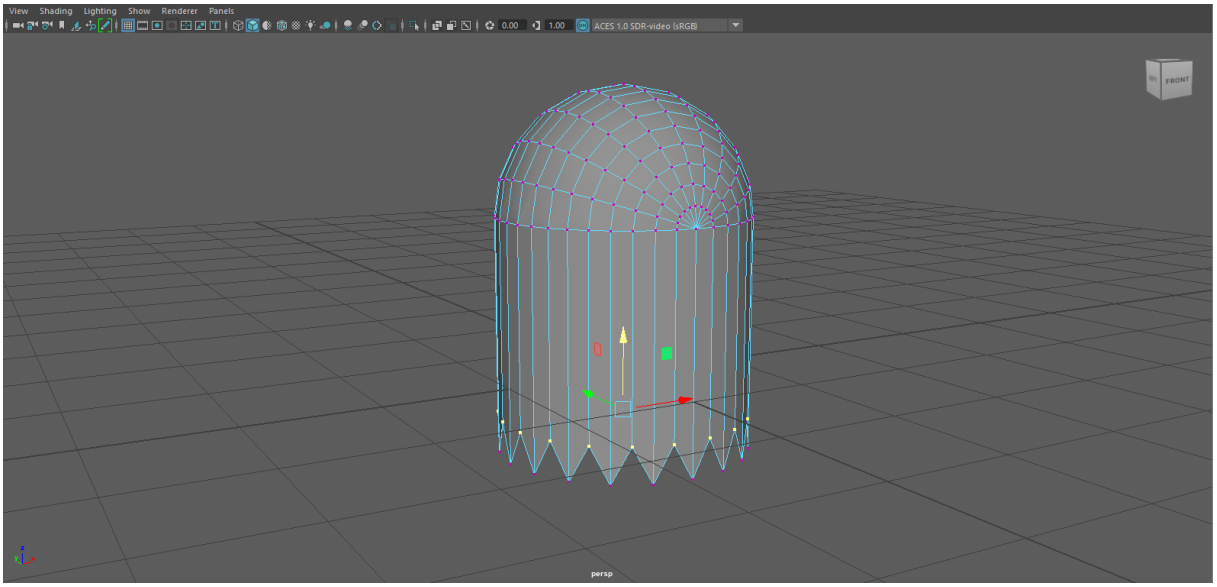
Nakon kreiranog tijela duha, ideja mi je bila kreirati vrhove ili „pipke“ koje klasični duh ima. Kao što sama riječ govori, za uređivanje vrhova prebacio sam se na alate za modeliranje vrhova objekta te selektirao svaki drugi vrh donje kružnice.




*Slika 9: Izrada modela duha – 5. korak*

*Izvor: autor*

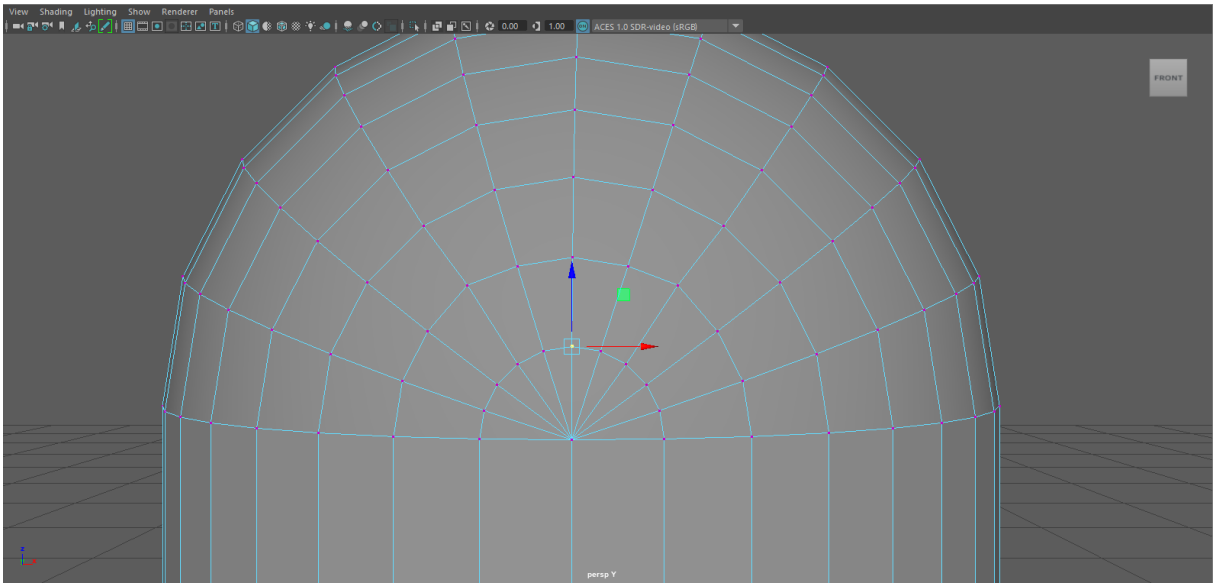
Kao što je prikazano na slici iznad, odabrane vrhove podigao sam prema gore dok nisam postigao željeni oblik sa slike ispod.




*Slika 10: Izrada modela duha – 6. korak*

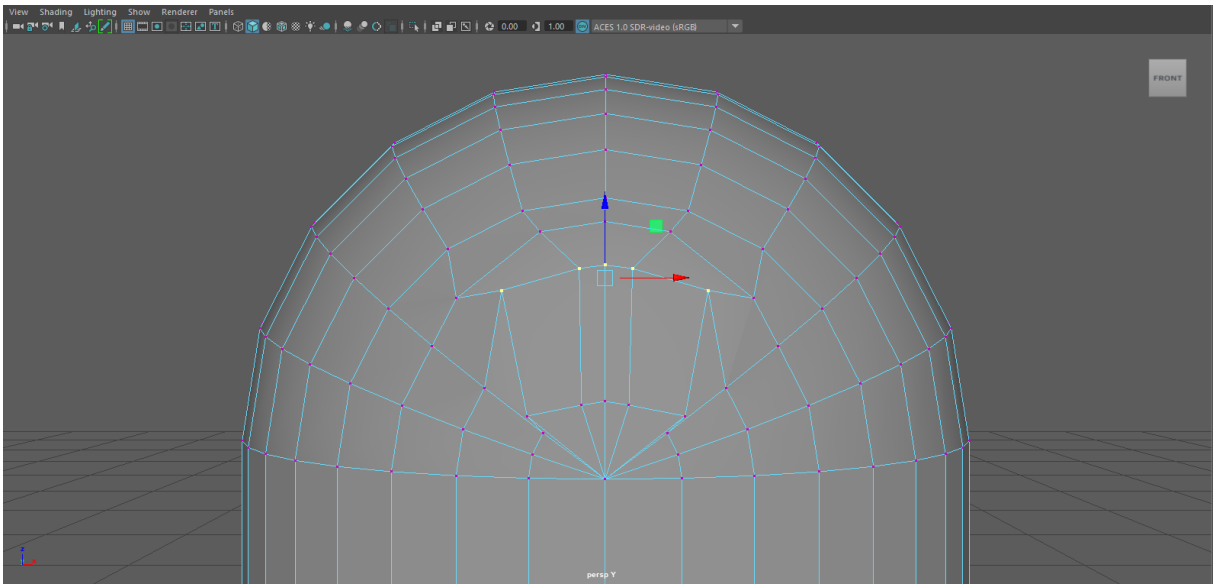
*Izvor: autor*

Kako bi ostvario dojam lika u igrici, a ne samo oblika duha, odlučio sam dodati mu i oblik očiju. I dalje unutar alata za modeliranje vrhova počeo sam sa selektiranjem mjesta gdje bi oči bile pozicionirane. Pomicanjem vrhova oblikovao sam mu oči dok nisam dobio željeni oblik.




*Slika 11: Izrada modela duha – 7. korak*

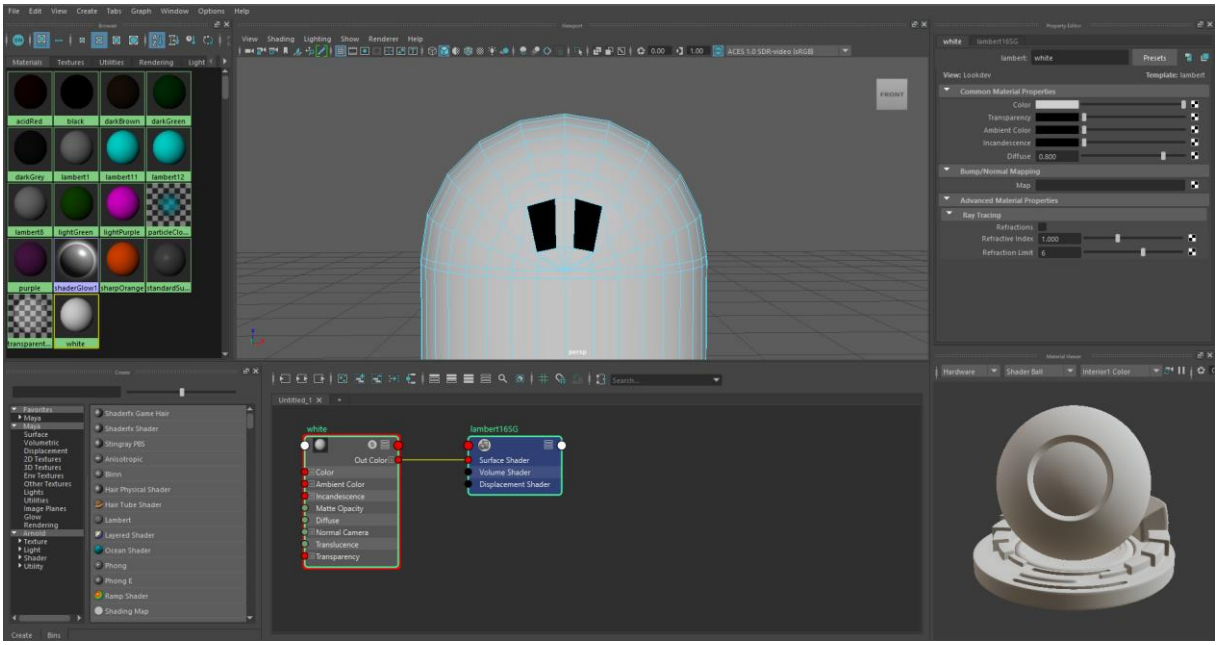
*Izvor: autor*




*Slika 12: Izrada modela duha – 8. korak*

*Izvor: autor*

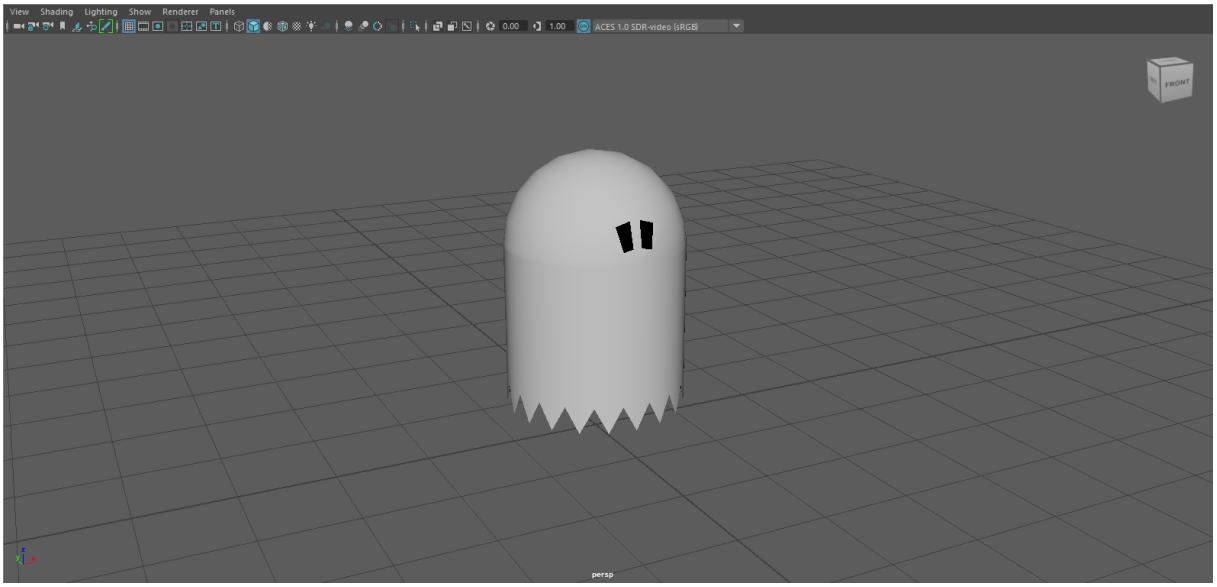
Tim korakom završio sam oblikovanje i modeliranje duha te je jedino preostalo dodati materijale oblicima kako bi model bio spreman za korištenje u igrici. „Maya“ koristi „Hypershade“ alat za uređivanje materijala i njihovu primjenu. Prvo se kreiraju svi materijali koji se planiraju primijeniti i ovdje „Maya“ također nudi velik spektar različitih svojstva materijala, a ja sam koristio predložak „lambert“ te mu kao svojstvo odredio boju. Za model duha kreirao sam dva materijala, bijeli za tijelo i crni za oči. Prvo sam označio kompletan model i dodijelio mu materijal bijele boje te sam nakon toga selektirao područja očiju kreirana iz prethodnog koraka te im dodijelio crnu boju kao što se može vidjeti na slici ispod.



*Slika 13: Izrada modela duha – 9. korak*

*Izvor: autor*

Nakon primijenjenih materijala završio sam cijeli proces modeliranja duha te je sad spreman za prebacivanje unutar moje igre. Naravno model se mogao još detaljnije oblikovati te postoje razne druge opcije i načini na koje se to može izvesti. Obično nakon samog procesa modeliranja slijedi proces animiranja u kojemu se stvaraju animacije i pokreti, ali ja sam taj dio pojednostavio i odradio ga programerski što ću više objasniti u sljedećem poglavlju same izrade računalne igre. Za kraj poglavlja ostavljam sliku završenog modela duha dobivenog praćenjem svih gore spomenutih koraka.




*Slika 14: Izrada modela duha – 10. korak*

*Izvor: autor*

# 4. Izrada računalne igre

U prethodna dva poglavlja obuhvatio sam najbitnije elemente koji su potrebni kako bi ispravno isplanirali računalnu igru i modelirali njene komponente. Ovo poglavlje pokriva sljedeći i najbitniji korak, odnosno samu izradu računalne igre.

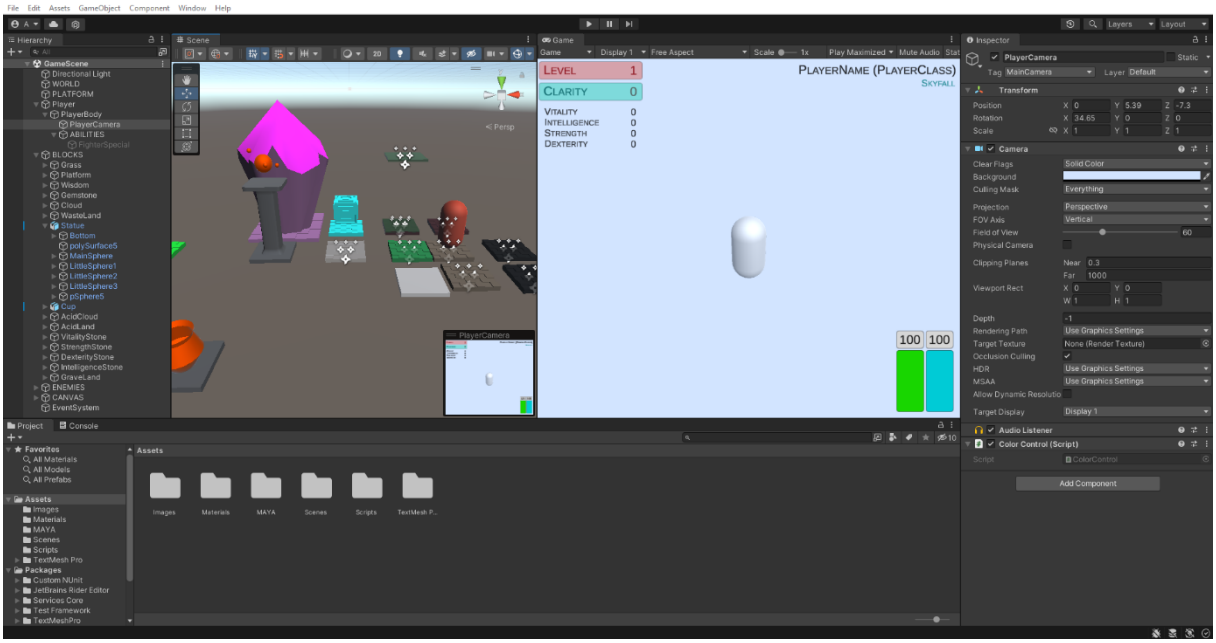
Izrada bi bio onaj korak u kojemu kreiranim modelima uz pomoć odabranog programskog jezika i njegovog razvojnog okruženja programiramo određeno ponašanje ili dodjeljujemo neke vrijednosti s ciljem ostvarivanja krajnjeg rezultata zacrtanog prilikom planiranja. To bi značilo da modelu igrača dodjeljujemo određene funkcije za pokretanje, promatranje svijeta oko njega i interakciju sa svijetom, dok npr. modelu platforme dodjeljujemo određene vrijednosti poput njene pozicije ili veličine. Programiranjem tih dijelova definiramo njihovu ulogu unutar računalne igre te njihovim spajanjem dobivamo računalnu igru u cjelini.

S time na umu prvo je potrebno odabrati programski jezik i razvojno okruženje koje se planira koristiti za izradu. Danas postoji mnogo različitih razvojnih okruženja ili poznatije „Game Engine“ pa je jednako kao i kod alata za modeliranje nužno odabrati onaj koji zadovoljava najveći broj kriterija. Neki od popularnijih razvojnih okruženja za izradu igrica su „Unreal Engine“, „Unity“, „GameMaker“, „CryEngine“ i slični, a za izradu svoje igre odlučio sam se na „Unity“.

Ostalo mi je još samo odlučiti se koji programski jezik planiram koristiti. „Unity“ nudi širok izbor dodataka za skoro svaki veći programski jezik, no ipak sam se odlučio za C# jezik koji je najpopularniji i u isto vrijeme ugrađeni jezik tog razvojnog okruženja. S odabranim okruženjem i programskim jezikom mogu krenuti na realizaciju vlastite računalne igre.

Kroz sljedeća potpoglavlja ukratko ću objasniti „Unity“ kao razvojno okruženje te nakon toga najvažnije dijelove koda koji su zaslužni za uspješan rad računalne igre. Radi lakšeg razumijevanja podijelio sam potpoglavlja prema skriptama za generiranje svijeta, upravljanje igračem i kontrolu neprijatelja. Naravno tu se nalaze i mnoge druge skripte za kontrolu atmosferskih uvjeta, prikaza informacija na ekranu, rad glavnog izbornika i slične, ali zbog vrlo širokog opsega skripti prikazat ću samo najbitnije.

## 4.1. „Unity Engine 2021“



*Slika 15: Unity Engine sučelje*

*Izvor: autor*

„Unity Engine 2021“ je razvojno okruženje za izradu 2D ili 3D igara koje mogu biti namijenjene za igranje na osobnim ili prijenosnim računalima, igraćim konzolama ili pametnim uređajima te je prilagođen svim većim operativnim sustavima. Samim time ima velik opseg prilagodljivosti te je pogodan za izradu vrlo kompatibilnih igrica za primjenu na širokom tržištu.

Na slici gore vidimo primjer korisničkog sučelja razvojnog okruženja „Unity“ koji je vrlo prilagodljiv te se može namjestiti tako da odgovara praksi programera koji ga koristi.

Na sredini sučelja nalaze se dva prozora s različitim pogledima, odnosno „Scene“ i „Game“ pogledi. Unutar „Scene“ pogleda namještamo i prilagođavamo položaj, rotaciju i svojstva modela, dok u „Game“ pogledu vidimo prikaz tih modela na onaj način kako ih igrač vidi tijekom igranja, odnosno pogled kroz glavnu kameru.

Lijevo se nalazi prozor „Hierarchy“ koji predstavlja hijerarhiju svih modela i objekata unutar trenutne scene. Također jedan objekt ili model može imati vlastitu hijerarhiju te u tom slučaju on predstavlja objekt roditelja, a objekti koji mu pripadaju predstavljaju njegovu djecu. Tako imamo primjer objekta „Player“ koji je roditelj objektima „PlayerBody“ i „PlayerCamera“.

S desne strane nalazi se prozor „Inspector“ s postavkama trenutno označenog objekta sa scene. Unutar tog prozora se numerički određuju generalne postavke objekta te mu se pridodaju razne implementirane komponente iz „Unity“ okruženja. Kao primjer na slici je označen objekt „PlayerCamera“ pa je tako u „Inspector“ prozoru namještena njegova pozicija, rotacija i veličina te mu je pridodana komponenta „Camera“, koja predstavlja razne postavke kamere i prilagodbe prikaza slike, komponenta „Audio Listener“, koja govori da se radi o komponenti s povratnim informacijama o zvuku unutar scene, i komponenta „Color Control“, koja je skripta za mijenjanje pozadinske boje kamere ovisno o platformi na kojoj se igrač nalazi.

„Project“ i „Console“ su preostali prozori koji se nalaze s donje strane razvojnog okruženja. Unutar „Project“ prozora nalaze se konkretne datoteke koje se koriste unutar računalne igre podijeljene po mapama prema tipu, a unutar „Console“ prozora vidimo povratne informacije o potencijalnim greškama i upozorenjima unutar koda ili scene te vlastite povratne informacije koje su definirane unutar koda te se većinom prikazuju prilikom razvoja računalne igre radi testiranja.

Postoje i dodatni prozori koji mogu biti postavljeni na odabrana mjesta poput „Lightning“ koji služi za postavke osvjetljenja, „Animation“ za uređivanje animacija, „Audio Mixer“ za miješanje različitih zvukova te mnogi drugi. Kao što sam naveo u početku prozori te njihov raspored ovisi striktno o preferencijama programera te je gore obrazložen raspored koji sam ja koristio prilikom izrade vlastite igre.

## 4.2. Generiranje svijeta

Kako bi uopće započeo s izradom računalne igre, prvi korak je bio stvoriti tj. generirati svijet kojim se igrač kreće te postaviti određena ograničenja i mehanike tog svijeta. Budući da se unutar igre svijet kreira zasebno te mu se dodjeljuju svojstva ovisno o tipu platforme na kojoj se igrač trenutno nalazi, prvo ću prikazati i objasniti dijelove koda zaslužene za generalno definiranje i stvaranje svijete te dodatno svojstva svijeta različitih tipova platformi.

```
// Funkcija koja se pokreće prva na početku
    void Start(){
        INFO = WorldParent.GetComponent<InfoControl>();
        PLAYER = PlayerParent.GetComponent<PlayerControl>();
        COLOR = ColorParent.GetComponent<ColorControl>();

        worldGenerated = new string[worldSize,worldSize];
        generateWorld("tutorialWorld");
        COLOR.SetColor("tutorialWorld");
    }
```

Na samom početku igre pokreće se funkcija „Start“ u kojoj se definiraju osnovne stvari potrebne za generiranje svijeta i pozivaju odgovarajuće funkcije. Sukladno tome dohvatio sam skriptu za prikaz informacija na ekranu, kontrolu igrača i kontrolu boje te pohranio unutar objekata „INFO“, „PLAYER“ i „COLOR“. Zatim sam deklarirao dvodimenzionalno polje definirano varijablom „worldSize“ koja može biti promjenjiva. Nakon toga sam pozvao funkciju „generateWorld“ s parametrom početne platforme koja se treba kreirati te sam „SetColor“ funkcijom postavio boju te platforme.

```
// Funkcija koja s obzirom na varijablu worldType generira svijet
    private void generateWorld(string worldType){
        switch (worldType)
        {
            case "tutorialWorld":
                generateTutorial();
                buildWorld();
                INFO.UpdatePlatform("Home");
                break;
            case "skyfallWorld":
                generateSkyfall();
                buildWorld();
                INFO.UpdatePlatform("Skyfall");
                break;
            case "zombieWorld":
                generateZombie();
                buildWorld();
                INFO.UpdatePlatform("ZombieLand");
                break;
            case "acidWorld":
                generateAcid();
                buildWorld();
                INFO.UpdatePlatform("AcidField");
                break;
            case "ghostWorld":
                generateGhost();
                buildWorld();
                INFO.UpdatePlatform("GraveYard");
                break;
            default:
                break;
        }
    }
```

Funkcija „generateWorld“ ovisno o proslijeđenom parametru poziva odgovarajuće funkcije za generiranje specifičnih platformi kroz koje ću proći kasnije te poziva funkciju „buildWorld“ i ažurira informacije o platformi putem funkcije „UpdatePlatform“.

```
// Funkcija koja s obzirom na generirani svijet postavlja blokove na
predviđene pozicije
    private void buildWorld(){
        for(int i = 0; i < worldSize; i++){
            for(int j = 0; j < worldSize; j++){
                if(worldGenerated[i,j] == "platform" && !platformBuild)
Instantiate(PlatformBlock, new Vector3(i, 0f, j), Quaternion.identity,
PlatformParent.transform);
                else if(worldGenerated[i,j] == "acidcloud") {
                    GameObject acidCloud = Instantiate(AcidCloudBlock,
new Vector3(i, 0f, j), Quaternion.identity, WorldParent.transform);

acidCloud.GetComponent<AcidCloudControl>().SetPosition(i,j);
                }
                else if(worldGenerated[i,j] != "platform")
if(getBlock(worldGenerated[i,j]))
Instantiate(getBlock(worldGenerated[i,j]), new Vector3(i, 0f, j),
Quaternion.identity, WorldParent.transform);
            }
        }

        if(!platformBuild) platformBuild = true;
    }
```

Funkcija „bulidWorld“ prolazi kroz dvodimenzionalno polje generiranog svijeta te ovisno o vrijednosti konkretnog elementa polja klonira odgovarajući predložak objekta. Jednostavnije rečeno prolazi kroz svaki element generiranog svijeta te instancira onaj objekt na odgovarajuće mjesto za koje je predviđen. Nakon završetka instanciranja svih objekata kreira se platforma ispunjena njoj pripadajućim elementima. Za dohvaćanje predložaka objekta koji se kloniraju koristi se funkcija „getBlock“ s parametrom bloka koji se želi dohvatiti te zatim klonirati. U nastavku ću pokazati proces popunjavanja polja za generiranje svijeta ovisno o platformi koja se generira.

### 4.2.1. Početni svijet

```
private void hardcodeObjects(){
        // Hardcodane informacije o položaju stupova
        worldGenerated[0,0] = "statue";
        worldGenerated[0,19] = "statue";
        worldGenerated[19,0] = "statue";
        worldGenerated[19,19] = "statue";

        // Hardcodane informacije o položaju vrčeva
        worldGenerated[0,1] = "cup";
        worldGenerated[1,0] = "cup";
        worldGenerated[1,19] = "cup";
        worldGenerated[19,1] = "cup";
        worldGenerated[0,18] = "cup";
        worldGenerated[18,0] = "cup";
        worldGenerated[19,18] = "cup";
        worldGenerated[18,19] = "cup";

        // Hardcodane informacije o položaju pomične platforme
        worldGenerated[9,1] = "platform";
        worldGenerated[10,1] = "platform";
        worldGenerated[8,2] = "platform";
        worldGenerated[9,2] = "platform";
        worldGenerated[10,2] = "platform";
        worldGenerated[11,2] = "platform";
        worldGenerated[8,3] = "platform";
        worldGenerated[9,3] = "platform";
        worldGenerated[10,3] = "platform";
        worldGenerated[11,3] = "platform";
        worldGenerated[9,4] = "platform";
        worldGenerated[10,4] = "platform";

        // Hardcodane informacije o položaju Gemstona
        worldGenerated[9,18] = "0";
        worldGenerated[10,18] = "0";
        worldGenerated[9,17] = "gemstoneGround";
        worldGenerated[10,17] = "0";
    }
```

Funkcija „hardcodeObjects“ popunjava svijet objektima koji su konstanta kod kreiranja svake platforme. Izvodi se kako bi se smanjili količinu koda prilikom dodavanja novih ili povećali jednostavnost za vrijeme čitanja postojećih funkcija generiranja platformi.

```
// Funkcija koja generira početnu tutorial platformu
    private void generateTutorial(){
        for(int i = 0; i < worldSize; i++){
            for(int j = 0; j < worldSize; j++){
                worldGenerated[i,j] = "grass";
            }
        }

        hardcodeObjects();

        worldGenerated[3,6] = "vitality";
        worldGenerated[3,8] = "intelligence";
        worldGenerated[3,10] = "dexterity";
        worldGenerated[3,12] = "strength";

        worldGenerated[9,9] = "wisdom";
        worldGenerated[10,9] = "wisdom";
    }
```

Za kreiranje početne uvodne platforme koristi se funkcija „generateTutorial“ koja popunjava platformu objektom „grass“, konstantnim objektima te „wisdom“ kutijama i alatima za nadogradnju karakteristika igrača. Ova se platforma kreira svakih deset razina te daje priliku igraču za odmor i povećanje karakteristika.

### 4.2.2. „Skyfall“ svijet

```
// Funkcija koja generira platformu sa oblacima
    private void generateSkyfall(){
        for(int i = 0; i < worldSize; i++){
            for(int j = 0; j < worldSize; j++){
                worldGenerated[i,j] = "cloud";
            }
        }

        hardcodeObjects();
    }
```

Generiranje „Skyfall“ platforme s padajućim oblacima postiže se na vrlo jednostavan način popunjavanja svijeta konstantnim objektima i objektima oblaka. Cilj ove platforme je uzeti „Gemstone“ i vratiti se natrag do pomične platforme prije nego svi oblaci propadnu.

### 4.2.3. „Zombie“ svijet

```
// Funkcija koja generira platformu sa zombijima
    private void generateZombie(){
        for(int i = 0; i < worldSize; i++){
            for(int j = 0; j < worldSize; j++){
                if(Random.Range(1, 20) == 5){
                    worldGenerated[i,j] = "zombie";
                }
                else{
                    worldGenerated[i,j] = "wasteland";
                }
            }
        }

        hardcodeObjects();
    }
```

Generiranje „Zombie“ platforme sastoji se od objekata „wasteland“ i „zombie“ te klasičnih konstantnih objekata. Nasumičnim omjerom gustoće postavlja neprijatelje zombije na nasumična mjesta unutar platforme. Cilj igrača je preživjeti napade zombija borbom ili izbjegavanjem, uzeti „Gemstone“ i vratiti se na pomičnu platformu kako bi nastavio dalje.

### 4.2.4. „Ghost“ svijet

```
// Funkcija koja generira platformu sa duhovima
    private void generateGhost(){
        for(int i = 0; i < worldSize; i++){
            for(int j = 0; j < worldSize; j++){
                if(Random.Range(1, 20) == 5){
                    worldGenerated[i,j] = "ghost";
                }
                else{
                    worldGenerated[i,j] = "graveyard";
                }
            }
        }

        hardcodeObjects();
    }
```

Slično kao i kod „Zombie“ platforme, „Ghost“ platforma generira groblje s duhovima te igrač ima jednak cilj, no ovaj put suočen s drugačijom mehanikom napada i kretanja duhova.

### 4.2.5. „Acid-Rain“ svijet

```
// Funkcija koja generira platformu sa kiselom kišom
    private void generateAcid(){
        for(int i = 0; i < worldSize; i++){
            for(int j = 0; j < worldSize; j++){
                if(Random.Range(1, 7) == 5){
                    worldGenerated[i,j] = "acidcloud";
                }
                else{
                    worldGenerated[i,j] = "acidland";
                }
            }
        }

        hardcodeObjects();
    }
```

U malo gušćem nasumičnom omjeru „Acid-Rain“ svijet popunjava se oblacima kisele kiše koju igrač mora izbjegavati kako bi uzeo „Gemstone“ te nastavio dalje. Polje kisele kiše dodatno je definirano te se nasumično premješta nakon određenog vremenskog perioda i time još više otežava prolazak ove platforme.

## 4.3. Igrač i kontrola igrača

```
// Podklasa za Playere tipa 'Fighter'
public class FighterClass:PlayerMain{
    public FighterClass(string name = "JackTheFighter")
    {
        SetBasic(name, "Fighter");
        SetCharacter("dexterity", "set", 4);
        SetCharacter("strength", "set", 10);
        SetCharacter("vitality", "set", 7);
        SetCharacter("intelligence", "set", 2);
        SetHealth("set", 20 * playerVitality);
        SetMana("set", 20 * playerIntelligence);
        SetLevel("set", 1);
        SetClarity("set", 0);
        specialDistance = 4f;
        specialSpeed = 20f;
    }
}

// Podklasa za Playere tipa 'Mage'
public class MageClass:PlayerMain{
    public MageClass(string name = "RudolphTheMage")
    {
        SetBasic(name, "Mage");
        SetCharacter("dexterity", "set", 6);
        SetCharacter("strength", "set", 5);
        SetCharacter("vitality", "set", 4);
        SetCharacter("intelligence", "set", 10);
        SetHealth("set", 20 * playerVitality);
        SetMana("set", 20 * playerIntelligence);
        SetLevel("set", 1);
        SetClarity("set", 0);
        specialDistance = 8f;
        specialSpeed = 40f;
    }

}
```

Prije nego sam krenuo u programiranje mehanika kretanja i ostalih funkcija samog igrača, kreirao sam klasu „PlayerMain“ s osnovnim karakteristikama svakog igrača neovisno o izabranom tipu te klase „FighterClass“ i „MageClass“ u čijim se konstruktorima podešavaju vrijednosti karakteristične za odabrane tipove. Jednom postavljen objekt tog tipa lako je mijenjati, čitati i dodatno definirati u bilo kojem trenu igre.

Kao što vidimo neke od važnijih vrijednosti vezane uz samog igrača su „Health“ i „Mana“ te predstavljaju trenutne životne bodove i količinu magije. Te vrijednosti se skaliraju karakteristikama „Vitality“ i „Intelligence“, dok se snaga igrača skalira karakteristikom „Strength“ i brzina karakteristikom „Dexterity“.

```
void Update(){
        if(playerAlive){
            movePlayer();
            checkFront();
            interactPlayer();
            attackPlayer();
        }
    }
```

Funkcija „Update“ poziva se prilikom svakog osvježavanja zaslona („frame“) i provjerava zastavicu „playerAlive“ te ako je igrač živ prolazi kroz nekoliko osnovnih funkcionalnosti igrača. Funkcije kroz koje prolazi su „movePlayer“ koja služi za pomicanje samog igrača i kuta kamere, „checkFront“ koja služi za provjeru prostora ispred igrača, „interactPlayer“ koja služi za interakciju igrača s objektima i svijetom oko njega te „attackPlayer“ koja služi za kontrolu napada.

```
// Funkcija za pomicanje Playera
    private void movePlayer(){

        // Dash Playera
        if(Input.GetButtonDown("Jump") && isGrounded()){
            if(dashCooldownCounter <= 0 && dashCounter <= 0){
                speed = dashSpeed;
                dashCounter = dashLength;
            }
        }

        if(dashCounter > 0){
            dashCounter -= Time.deltaTime;

            if(dashCounter <= 0){
                speed = moveSpeed;
                dashCooldownCounter = dashCooldown;
            }
        }

        if(dashCooldownCounter > 0){
            dashCooldownCounter -= Time.deltaTime;
        }

        // Pomicanje Playera
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        if(isGrounded()){
            playerRB.velocity =
PlayerBody.transform.TransformDirection(new Vector3(moveHorizontal, 0f,
moveVertical) * speed);
        }

        // Rotacija Playera

        angleVelocity = new Vector3(0, Input.GetAxis("Mouse X") *
rotationSpeed, 0);
        Quaternion deltaRotation = Quaternion.Euler(angleVelocity);
        playerRB.MoveRotation(playerRB.rotation * deltaRotation);
```

```
        // Provjera je li player pao sa platforme
        if(PlayerBody.transform.position.y < deathDistance){
            playerAlive = false;
            destroyPlayer();
        }
    }
```

Funkcija za pomicanje igrača „movePlayer“ čita određene ulaze horizontalnog i vertikalnog pomicanja, pritisak razmaknice i kut miša te ovisno o tim ulazima primjenjuje odgovarajuće promjene vezane uz poziciju igrača i kut njemu pripadajuće kamere.

Pritiskom razmaknice igraču se brzina mijenja s uobičajene na ubrzanu kratak period vremena te predstavlja ubrzanje igrača u onom smjeru u kojem se kretao. Služi kako bi brže i lakše izbjegao određene prepreke i neprijatelje. Tipke horizontalnog i vertikalnog pomicanja postavljaju brzinu igraču u smjeru ovisnom o ulaznim parametrima tipki. Čitanjem i spremanjem pomicanja miša određuje se i  zakretanje kamere za proporcionalnu vrijednost kuta. Također ovdje se provjerava i ako je igrač pao s platforme te se zastavica „playerAlive“ spušta i pokreće se funkcija „destroyPlayer“ koja je odgovorna za proceduru nakon smrti igrača.

```
// Funkcija za provjeru objekata ispred Playera
    private void checkFront(){
        RaycastHit hit;

        if (Physics.Raycast (PlayerBody.transform.position,
PlayerBody.transform.forward, out hit, raycastFront)) {
            if(interactionObject != hit.collider.gameObject &&
interactionObject != null){
                focus(interactionObject.transform.name, false);
            }
            interactionObject = hit.collider.gameObject;
            focus(interactionObject.transform.name, true);
        } else if(interactionObject != null){
                focus(interactionObject.transform.name, false);
                interactionObject = null;
        }
    }
```

Funkcija „checkFront“ provjerava ispred igrača te sukladno time postavlja fokus na objekte u relativnoj blizini. To radi pomoću funkcije „focus“ s parametrima imena objekta i zastavicom fokusa, o kojoj ovisi postavlja li se fokus ili se pomiče s navedenog objekta.

```
// Funkcija za kontrolu interakcije Playera
    private void interactPlayer(){
        if(interactionObject != null){
            if(Input.GetKeyDown(KeyCode.E)){
                interactWith(interactionObject.transform.name);
            }
        }
        if(Input.GetKeyDown(KeyCode.Q)){
            if(isPlatformed() && PLAYER.HasKey()){
                playerUntouchable = true;
                PLAYER.SetKey(false);
                WORLD.MoveUp(true);
            }
        }
        if(Input.GetMouseButtonDown(0)){ // lijevi klik
            attackState = true;
        }
        if(Input.GetMouseButtonDown(1)){  // desni klik
            if(PLAYER.GetMana() >= specialCost){
                PLAYER.SetMana("decrease", specialCost);
                specialState = true;
                special.SetActive(true);
            }
        }
    }
```

Funkcija „interactWith“ odgovorna je za interakciju sa svijetom te provjerava ako postoji fokusirani objekt i ako je pritisnuta tipka za interakciju („E“) i tada ovisno o objektu izvršava određen proces interakcije. Ovdje se također provjerava pritisak lijeve i desne tipke miša čime se inicijaliziraju obični i specijalni napadi. Pritiskom tipke za nastavljanje pomičnom platformom igrač provjerava se je li igrač na platformi i posjeduje li „Gemstone“ te mu u tom slučaju dozvoljava prolaz i inicijalizira pokretanje platforme.

## 4.4. Kontrola neprijatelja

```
// Podklasa za Enemy tipa 'Zombie'
public class ZombieClass:EnemyMain{
    public ZombieClass()
    {
        HEALTH = 200;
        DAMAGE = 35;
        attackRange = 1f;
        spotRange = 5f;
        moveSpeed = 2.0f;
    }
}

// Podklasa za Enemy tipa 'Zombie'
public class GhostClass:EnemyMain{
    public GhostClass()
    {
        HEALTH = 150;
        DAMAGE = 50;
        attackRange = 5f;
        spotRange = 7f;
        moveSpeed = 3.0f;
    }
}
```

Kao i kod tipova igrača, prvo sam kreirao generalnu klasu za sve neprijatelje te je poslije toga bilo lako kreirati različite tipove klasa s različitim karakteristikama ovisno o tipu neprijatelja. U ovom primjeru konkretno se radi o „Ghost“ i „Zombie“ neprijateljima. Svaki tip ima svoje životne bodove, napadačke bodove i brzinu te različitu udaljenost na kojoj primjećuje igrača te se pomiče prema njemu i različitu udaljenost s koje napada igrača.

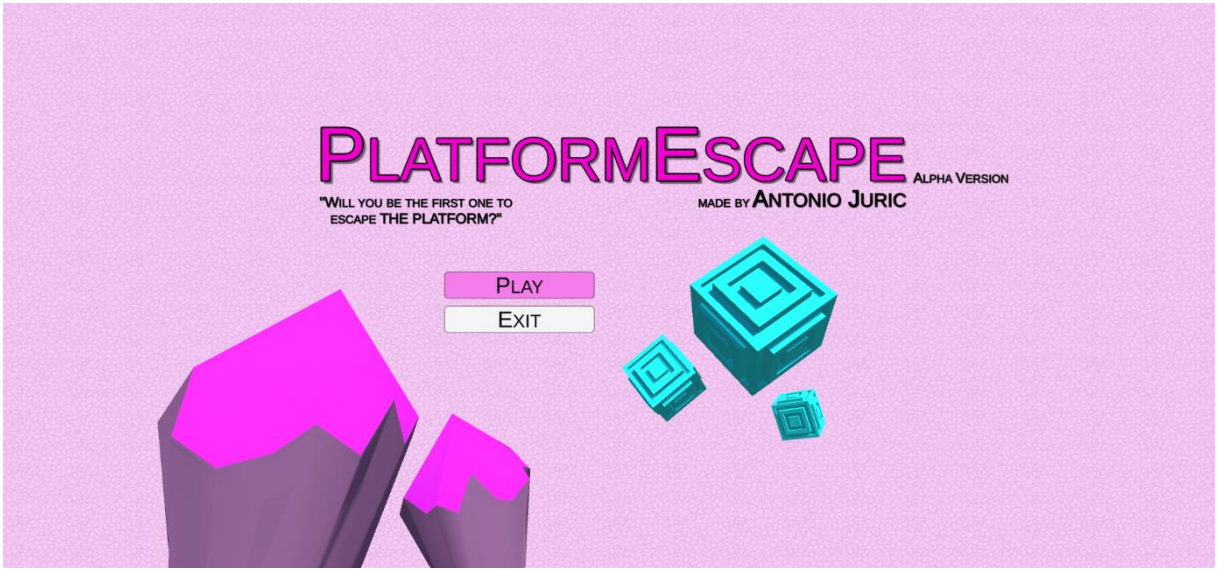
```
private void zombieMove(){
        distance = Vector3.Distance(PLAYER.transform.position,
transform.position);
        if(distance < ENEMY.GetSpotRange() && distance >
ENEMY.GetAttackRange()){
            Vector3 followPosition = new Vector3(target.position.x, -
2.35f, target.position.z);
            Vector3 direction = followPosition - transform.position;
            direction = direction.normalized;
            Vector3 velocity = direction * ENEMY.GetMoveSpeed();
            EnemyController.Move(velocity * Time.deltaTime);
        } else if(distance <= ENEMY.GetAttackRange()){
            PlayerScript.Damage(ENEMY.GetDamage());
        }
    }

private void ghostMove(){
        distance = Vector3.Distance(PLAYER.transform.position,
transform.position);
        if(distance < ENEMY.GetSpotRange() && distance >
ENEMY.GetAttackRange() + 0.2f){
            Vector3 followPosition = new Vector3(target.position.x,
transform.position.y, target.position.z);
            Vector3 direction = followPosition - transform.position;
            direction = direction.normalized;
            Vector3 velocity = direction * ENEMY.GetMoveSpeed();
            EnemyController.Move(velocity * Time.deltaTime);
        } else if(distance <= ENEMY.GetAttackRange() - 0.2f){
            Vector3 followPosition = new Vector3(target.position.x,
transform.position.y, target.position.z);
            Vector3 direction = followPosition - transform.position;
            direction = direction.normalized;
            Vector3 velocity = direction * ENEMY.GetMoveSpeed();
            EnemyController.Move(-velocity * Time.deltaTime);
            PlayerScript.Damage(ENEMY.GetDamage());
        }
    }
```

Gore prikazan isječak koda prikazuje funkcije „ghostMove“ i „zombieMove“ koje služe za provjeru okoline zombija ili duha te ovisno o blizini igrača njihovo pomicanje. Varijable udaljenosti, blizine i brzine pomicanja definirane su u klasi tipa neprijatelja te se s obzirom na njih i kalkuliraju. Razlika između kretanja zombija i duha je u tome što će zombi doći na blizinu napada i napadati ako je dovoljno blizu, dok će duh držati određenu udaljenost od igrača i napadati. Time je mnogo teže igraču uhvatiti duha i boriti se s njime, ali mu je lakše pobjeći njima nego zombijima.

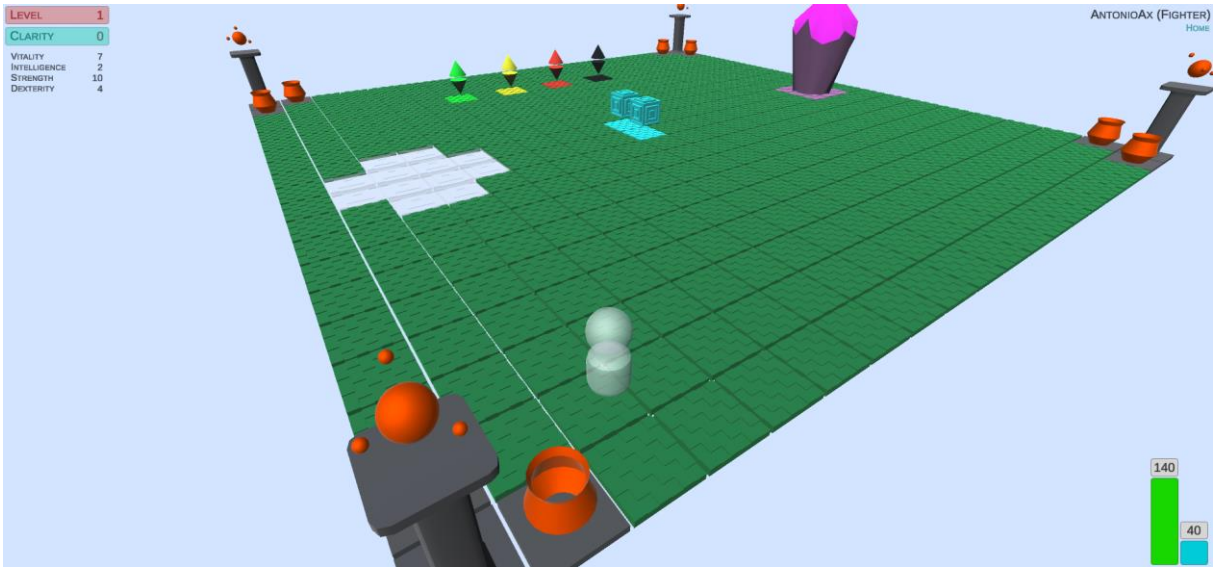
# 5. Konačni rezultat

Kao prikaz konačnog rezultat planiranja, modeliranja i izrade vlastite računalne igre u nastavku je prikazano nekoliko slika izrezanih prilikom igranja. Slike prikazuju konačni izgled glavnog izbornika te redom svaku od zamišljenih platformi tijekom igranja.



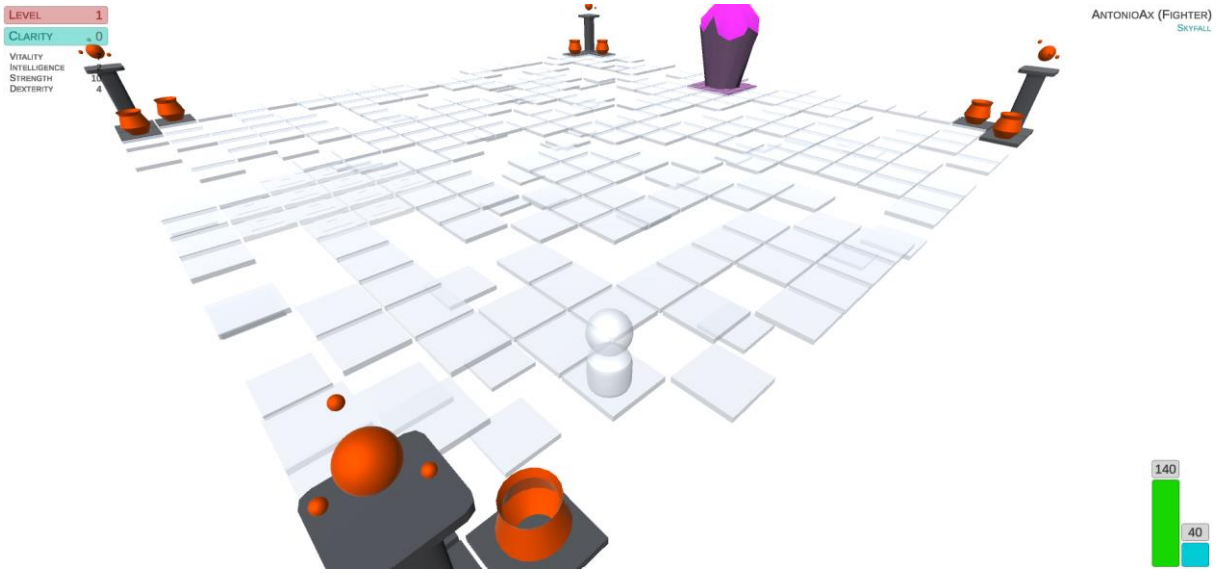

*Slika 16: Glavni izbornik*

*Izvor: autor*



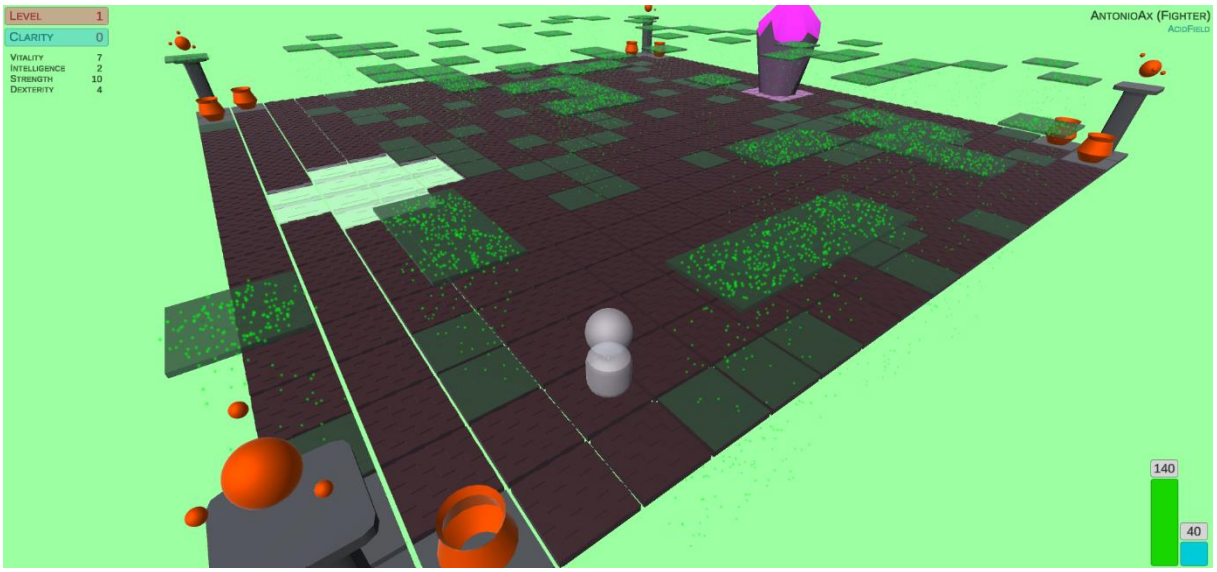

*Slika 17: Početni svijet*

*Izvor: autor*



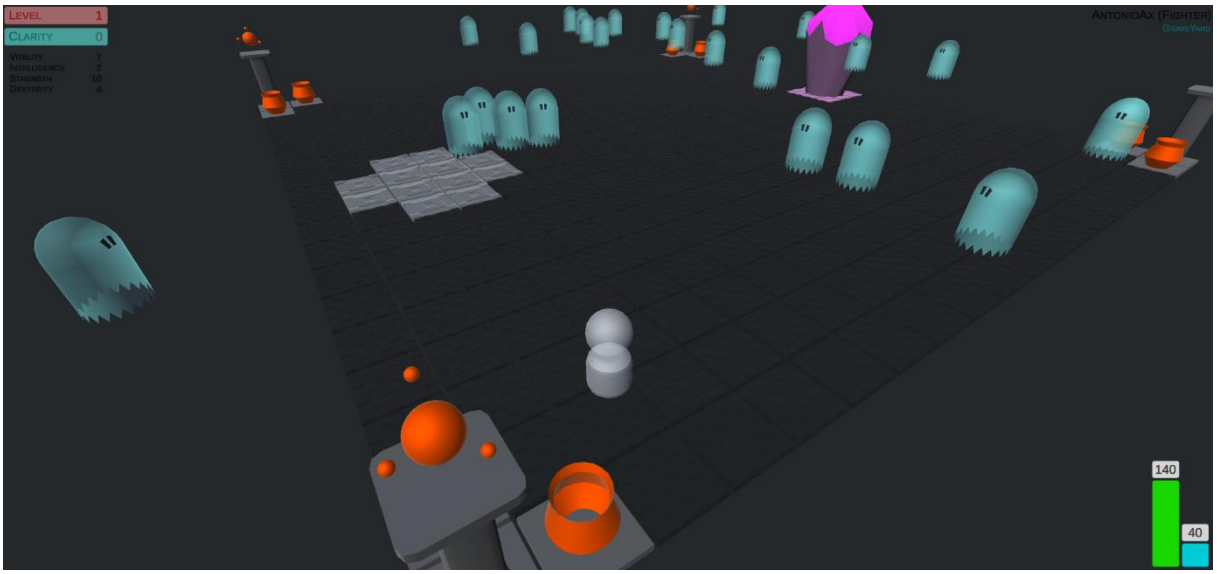

*Slika 18: „Skyfall“ svijet*

*Izvor: autor*



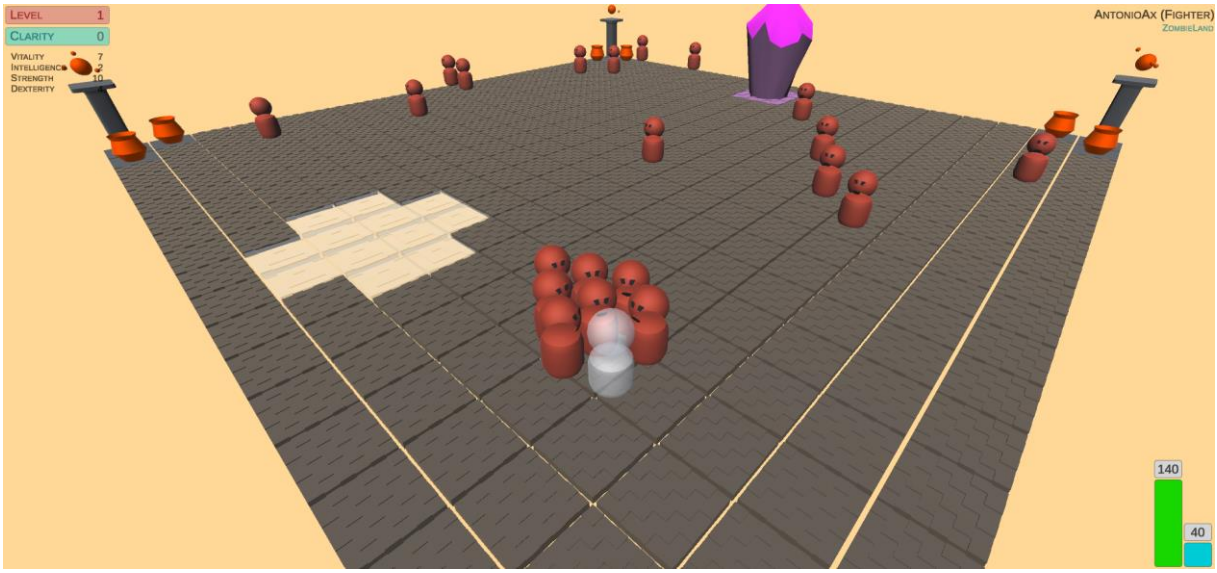

*Slika 19: „Acid-Rain“ svijet*

*Izvor: autor*




*Slika 20: „Ghost“ svijet*

*Izvor: autor*




*Slika 21: „Zombie“ svijet*

*Izvor: autor*

# 6. Zaključak

U ovom radu prošao sam kroz ključne elemente planiranja, modeliranja i izrade računalne igre. Obrazložio sam zbog čega je proces planiranja bitno postaviti kao prvi korak te kako tip, žanr i ideja igre definiraju neke glavne smjernice prema izboru alata i tehnika za uspješnu provedbu sljedećih koraka. Koristeći alat „Autodesk Maya 2023“ objasnio sam način osnovnog 3D modeliranja te praktično modeliranje pokazao na primjeru jednog od neprijatelja u igrici. Razvojnim okruženjem „Unity“ ideju igre sam proveo u djelo tako da sam kreiranim modelima dodijelio C# skripte i prikladne komponente te ih međusobno povezao u cjelinu. Na kraju sam prikazao rezultate isječcima iz igrice i time zaključio temu ovog rada.

Konačna verzija igre „Platform Escape“ funkcionalna je te predstavlja primjer jedne originalno osmišljene igre koja je napravljena upravo prateći korake navedene unutar ovog rada. Glavna ideja i priča igre je ostvarena, zajedno sa osnovnim modelima objekata te mehanikama igre, no to ne mora nužno značiti da je igra konačna, odnosno postoji puno različitih načina mogućih unaprjeđenja.

„Platform Escape“ može se proširiti tako da se dodaju nove vrste platformi s još izazovnijim sustavima i neprijateljima. Vrlo je lako dodati nove tipove igrača, s unikatnim mogućnostima i mehanikama te također nove vrste neprijatelja i različitih atmosfera. Moguće je i poboljšati trenutne modele ili dodati nove detalje unutar igre te ju tako učiniti još ljepšom i originalnijom.

Prije objavljivanja igre na tržišta, potrebno je također provesti razna testiranja, obično provedena od strane krajnjih korisnika tj. igrača, kako bi se utvrdile potencijalne greške ili nedostaci koji se zatim popravljaju. Nakon toga igra se može objaviti te s vremena na vrijeme ažurirati dodavanjem novih funkcionalnosti ili samo eventualnim naknadnim popravcima.

# 7. Literatura

[1] Autodesk Maya 2023, https://help.autodesk.com/view/MAYAUL/2023/ENU/

[2] C#, https://docs.microsoft.com/en-us/dotnet/csharp/

[3] Unity, https://docs.unity.com/

# 8. Popis slika