Igra za više igrača u Unity Engineu

Štrekelj, Juraj

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij računarstvo

IGRA ZA VIŠE IGRAČA U UNITY ENGINEU

Završni rad

Juraj Štrekelj

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 25.09.2017.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Juraj Štrekelj
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3705, 28.09.2017.
OIB studenta:	17040805387
Mentor:	Doc.dr.sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Igra za više igrača u Unity Engineu
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	25.09.2017.
Datum potvrde ocjene Odbora:	16.07.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 11.09.2018.

Ime i prezime studenta:	Juraj Štrekelj
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3705, 28.09.2017.
Ephorus podudaranje [%]:	0

Ovom izjavom izjavljujem da je rad pod nazivom**: Igra za više igrača u Unity Engineu**

izrađen pod vodstvom mentora Doc.dr.sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Zadatak završnog rada je izraditi igru za više igrača s mogućnošću prijelaza iz 2D pogleda u 3D pogled. Igre za više igrača dolaze u dva oblika: lokalno igranje (engl. *local play*) i umreženo igranje (engl. *online play*). Projekt je zamišljen kao online igra mješavine dvaju žanrova: strategije u stvarnom vremenu (engl. *Real-Time Strategy*) i „pucačine“ u trećem licu (engl. *Thrid-Person Shooter*). Partije se igraju jedan-na-jedan, gdje svaki igrač ima kontrolu nad postrojbama svoga tima. Igra započinje u strateškom pogledu iz ptičje perspektive (engl. 2D *top-down view*). Tijekom igre mogu se izrađivati nove postrojbe i može se upravljati postojećim postrojbama i građevinama. Svaki igrač ima također na raspolaganju jednog heroja s kojim može upravljati u trećem licu.

Igra je izrađena u *Unity* razvojnom okruženju, na verziji 5.4.1. koja je bila aktualna verzija u vrijeme započinjanja izrade završnog rada. *Unity* podržava veliki broj platformi (trenutno njih 25), među kojima se ubrajaju: *Mac, Linux, Windows, PlayStation 4* i *Xbox ONE*. Određene radnje objekata se određuju pomoću skripti koje su korisniku ponuđene unutar *Unity* razvojnog okruženja ili koje je korisnik sam napisao. 3D modeli objekata korišteni u igri izrađeni su pomoću *3D Studio Max* (*3DS Max*) programskog paketa. *3DS Max* ima velik broj alata za izradu modela, animaciju i teksturiranje, čime je odličan izbor za izradu ove igre. Određeni elementi su preuzeti sa *Unity*-jevog dućana, *Asset Store*. U *Asset Store*-u se mogu naći razni objekti koji se plaćaju ili su besplatni. Jedini objekt preuzet s *Asset Store*-a implementiran u projektu je *Photon Unity Networking* (PUN). PUN omogućava jednostavnu izradu igre za više igrača. Pomoću PUN API-a (engl. *Application Programming Interface,* API) možemo vrlo jednostavno uz pomoć dokumentacije na njihovoj stranici izraditi igru za više igrača.

1.1. Zadatak završnog rada

U radu je potrebno opisati korištene tehnologije za izradu 2D, ali i 3D dijela igre. Također je potrebno temeljito objasniti načela prelaženja iz 2D u 3D okruženje u smislu grafike, skripti i samoga *gameplay*-ja. U praktičnom dijelu potrebno je napraviti igru za više igrača u *Unity*-ju.

2. UNITY

Unity je višeplatformsko okruženje za razvoj video igara, koje je razvio *Unity Technologies*. *Unity* se često koristi za razvoj računalnih igara i simulacija, koje se mogu pokrenuti na računalu, igraćim konzolama i mobilnim uređajima. *Unity* je prvi put najavljen 2005. godine na *Apple*-ovoj *Worldwide Developer* konferenciji. Gotovo 12 godina nakon konferencije *Unity* se proširio na nevjerojatnih 25 platformi i planira se proširiti na još dodatne dvije platforme. Kroz svoje razvojno razdoblje *Unity* je imao pet glavnih verzija (za vrijeme pisanja rada posljednja stabilna verzija je bila 5.6.2), sa šestom verzijom (imenovana *Unity* 2017) koja se još razvija. Korisnicima su ponuđene četiri tipa za licenci *Unity*-ja: *Personal*, *Plus*, *Pro* i *Enterprise*. Svaka verzija ima različite mogućnosti. *Personal* verzija je besplatna i svakome je dostupna nakon što naprave račun na *Unity*-jevoj stranici. *Personal* verzija ima pristup svim mogućnostima iz *Unity* pokretača (engl. *Unity Engine*) kao i sve ostale verzije, ali korisnici *Personal* i *Plus* verzije nemaju pristup izvornom kodu (engl. *Source Code*) samog pokretača dok *Pro* i *Enterprise* imaju (Tab. 2.1.). Ovakav model pruža vrlo dobar izbor za korisnike ovisno o veličini zamišljenog projekta. Za potrebe ovog završnog rada izabrana je *Personal* verzija.

Tab. 2.1. Tipovi licenca.

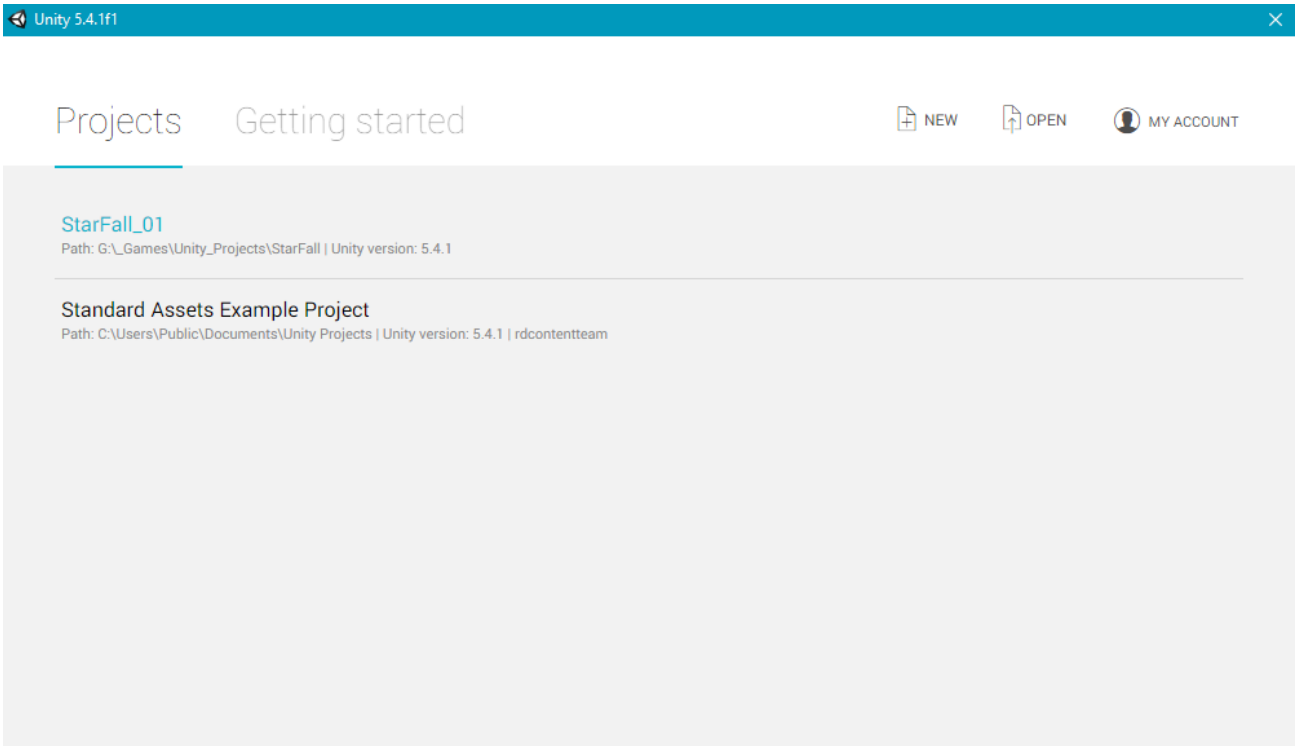
Ime licence	**Ima sve mogućnosti *Unity-a*?**	**Naslovni ekran**	**Kapacitet prihoda**	**Izvorni kod**	**Cijena**
Personal	Da	Napravljeno u *Unity-u*	$100 000	Ne	Besplatno
Plus	Da	Vlastito postavljanje	$200 000	Ne	$35 mjesečno
Pro	Da	Vlastito postavljanje	Beskonačno	Da	$125 mjesečno
Enterprise	Da	Vlastito postavljanje	Beskonačno	Da	Po dogovoru

Značajni uspjeh *Unity*-ja se može pridodati širokom rasponu platformi koje ga podržavaju, ali i tome što se lako nauči zbog količine sadržaja dostupne na internetu te opširnoj dokumentaciji

dostupnoj na web stranici *Unity*-ja. *Unity* je napisan u *C* i *C++* jeziku dok za skriptiranje ponašanja objekata u igri se mogu koristiti *C#*, *UnityScript* (vrlo sličan *JavaScript*-u) i *Boo*. Skriptiranje unutar *Unity*-ja je izrađeno na *Mono*-u, koji je *open-source* implementacija *Microsoft*-ovog *.NET Framework*-a. Skripte koje korisnik napiše ili unaprijed napisane skripte koje dolaze sa *Unity*-jem se nazivaju komponentama. Skripte, odnosno komponente, se postavljaju na objekte u igri pomoću interaktivnog korisničkog sučelja čime se znatno ubrza razvoj igre. *Unity* dolazi s unaprijed definiranim komponentama među kojima su: *Rigidbody*, *Collider*, *Transform* itd. Određene komponente su unaprijed postavljene na svakom objektu koji se doda u igru ili će proširiti mogućnosti tog objekta na kojem su dodane. Komponente mogu dodat mogućnosti kao što su: otkrivanje sudara dvaju objekata, utjecaja sile teže na objekte itd. Skripte u ovom završnom radu su napisane u *C#* programskom jeziku.

Unity podržava sljedeća grafička sučelja za programiranje aplikacija (engl. *graphics* API). Podržava *Direct3D* i *Vulkan* na *Windows* i *Xbox 360* platformi; *OpenGL* na *Mac*, *Linux* i *Windows* platformi; *OpenGL ES* na *Android* i *iOS* platformi; i druge API-je na igraćim konzolama. Unutar *Unity*-ja moguće je izraditi igre s 3D i 2D grafikom. Za 3D igre *Unity* ima podršku za kompresiju tekstura, „*Bump*“ mapiranje, mapiranje refleksija, *parallax* mapiranje, *Screen Space Ambient Occlusion* (*SSAO*), dinamičke sjene pomoću mapa sjena itd. Za 2D igre *Unity* pruža podršku za *sprite*-ove i napredni 2D *renderer*. *Unity* pruža servise korisnicima među kojima su: *Unity Ads, Unity Analytics, Unity Certification* itd. kojima se želi pomoći u razvoju ili marketingu igre.

Prilikom pokretanja *Unity*-a otvara se prozor koji traži da se prijavimo u naš *Unity* račun. Nakon prijave možemo započet novi projekt ili nastaviti raditi na postojećem projektu. Ako kreiramo novi projekt, trebamo navesti ime projekta, mjesto spremanja na tvrdom disku, organizaciji kojoj pripada, radi li se o 3D ili 2D projektu itd. *Unity* na svojoj stranici omogućuje aktivaciju raznih servisa za projekte pod našim vlasništvom, također je moguće dijeliti projekt s drugim korisnicima. U ovom slučaju izabire se projekt „StarFall_01“.

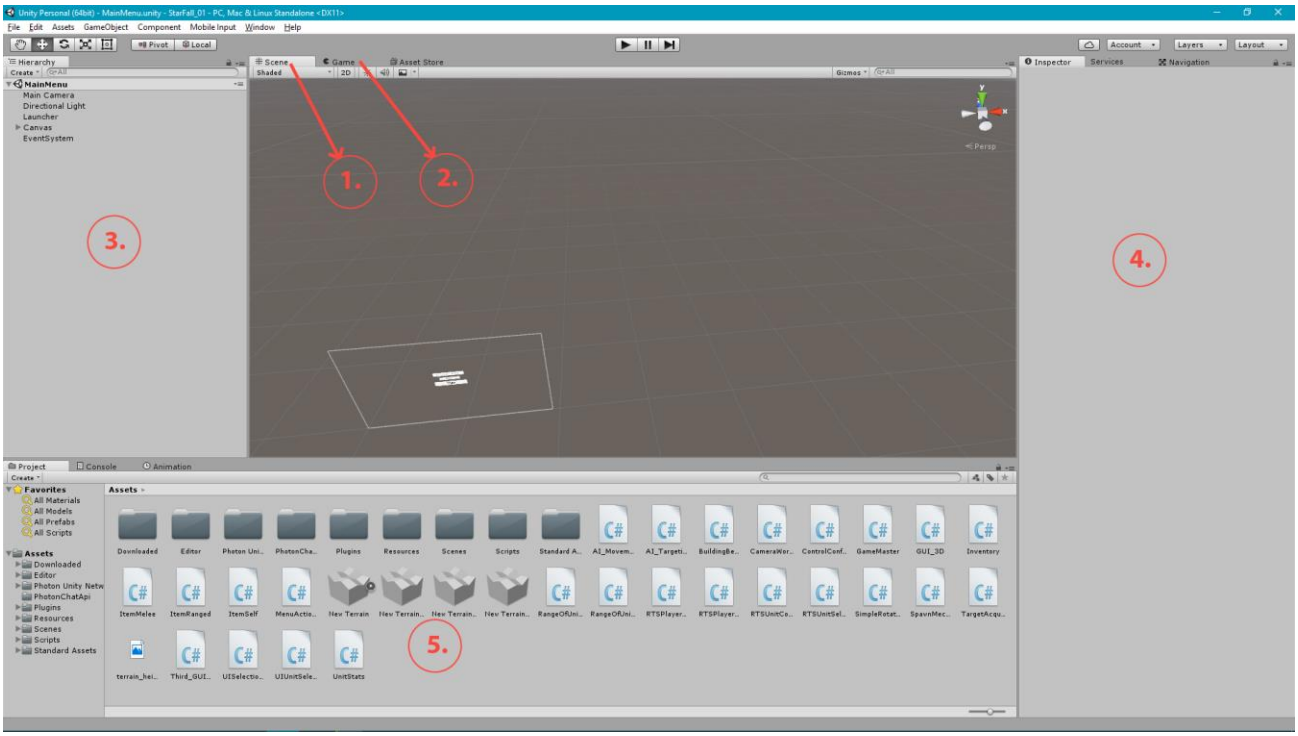


Sl. 2.2. Odabir/izrada projekta unutar Unitya.

Odabirom projekta (Sl. 2.2) otvara se *Unity*-jevo razvojno okruženje (engl. *integrated development environment*, IDE). *Unity* ima mogućnost prilagođavanja sučelja po želji korisnika, također je moguće odabrati jedan od 4 unaprijed određenih rasporeda sučelja: *2 by 3, 4 Split, Tall* i *Wide*.

Sučelje se sastoji od pet pogleda (engl. *Views*) (Sl. 2.3.). Pogledi pružaju mogućnost pregleda određenih komponenata unutar projekta. Scene u *Unity*-ju su ekvivalent razinama unutar igara, pa možemo zaključiti da pogled scene (engl. *Scene View*) (Sl. 2.3. Oznaka 1.) pruža pogled na razinu igre. Preko *Scene View*-a postavljamo objekte kao što su likovi, zgrade, zemljište itd. Objekte je moguće pomicati, rotirati i skalirat unutar *Scene View*-a koristeći alate za transformaciju. Jedna od prednosti *Unity*-ja je mogućnost igranja igre unutar samog razvojnog okruženja, što ubrzava razvoj igre. Pogled igre (engl. *Game View*) (Sl. 2.3. Oznaka 2.) je pogled koji se koristi radi testiranja igre tijekom njenog razvoja. Svijet unutar igre vidimo pomoću kamera koje *render*-iraju scenu i služe kao „oči“ igrača. Sve promjene na sceni tijekom testiranja igre unutar *Game View*-a, su samo privremenog karaktera, odnosno nakon završetka testiranja neće ostati zabilježene. Hijerarhijski pogled (engl. *Hierarchy View*) (Sl. 2.3. Oznaka 3.) prikazuje hijerarhijski odnos svih objekata u sceni. Dvoklikom

na objekt iz hijerarhijskog pogleda prvo će se kamera fokusirati na odabrani objekt, te se zatim popuni inspektor (engl. *Inspector View*) (Sl. 2.3. Oznaka 4.) informacijama o tom objektu. Kao što je rečeno, inspektor prikazuje sve detalje u vezi nekog objekta. Unutar inspektora možemo vidjeti sve komponente koje se nalaze na odabranom objektu, te neke parametre (uglavnom varijable sa vidljivosti *public*) koji su nam dostupni za podešavanje. Pregled projekta (engl. *Project View*) (Sl. 2.3. Oznaka 5.) služi kao upravitelj datoteka projekta. Prikazuje sve datoteke i mape koje su dio aktivnog projekta, te omogućuje organizaciju postojećih i izradu novih datoteka i mapa za projekt. Ako želite raditi neke izmjene na datotekama koje su dio projekta, te izmjene je potrebno raditi unutar *Project View*-a. Imamo dva prozora unutar *Project View*-a, jedan prikazuje hijerarhijsku strukturu datoteka, a drugi sadržaj aktivne mape.



Sl. 2.3. Unity razvojno okruženje s oznakama pogleda. Oznake:
1. *Scene View*; 2. *Game View*; 3. *Hierarchy View*; 4. *Inspector View*; 5. *Project View*

2.1. Photon

Photon je posredni softver (engl. *middleware*) koji omogućuje razvoj skalabilnih više platformskih igara za više igrača u stvarnom vremenu. *Photon* je razvila njemačka tvrtka, *Exit Games*. *Photon* pruža pakete za razvoj programa (engl. *Software Development Kit*, SDK) koji sadrže biblioteke za razne platforme, kao što su *Unity*, *iOS*, *Android*, *Windows* itd. S obzirom na to da se sve biblioteke spajaju na istu poslužiteljsku aplikaciju (engl. *back-end*) koristeći iste komande i logiku, što omogućuje više platformsko igranje. Postoje dva *Photon* servisa: *Photon Server* i *Photon Cloud*. Za potrebe ovog projekta i radi jednostavnosti, koristit će se *Photon Cloud*. Korištenjem *Photon Cloud*-a izbjegavamo pisanje kompleksne serverske logike. *Photon Cloud* je skalabilan, u smislu da se broj istovremeno spojenih igrača mijenja ovisno o potrebnom kapacitetu i to neprimjetno. Garantiran je niski odziv, jer *Photon Cloud* pruža servere u SAD-u, Europi i Aziji. *Photon Cloud* se prodaje kao, softver kao usluga (engl. *software as a service*). Dostupne ponude licenciranja servisa bazirana su na broju istovremeno povezanih igrača i plaćaju se mjesečno. Besplatna ponuda nudi 20 istovremeno povezanih igrača što je dovoljno u svrhu razvoja igre.

Kako bi mogli koristiti *Photon Cloud* u našem projektu, trebamo skinut *Photon Unity Networking* (PUN) sa *Unity*-jevog *Asset Store*-a. PUN se pokreće na *Photon Cloud*-u, te omogućuje uparivanje igrača nasumično ili ovisno o zadanim uvjetima pretraživanja, uz to ima i sve mogućnosti koje su navedene za *Photon Cloud*. Dostupna dokumentacija za PUN omogućava vrlo brzu i laku implementaciju u projekt.

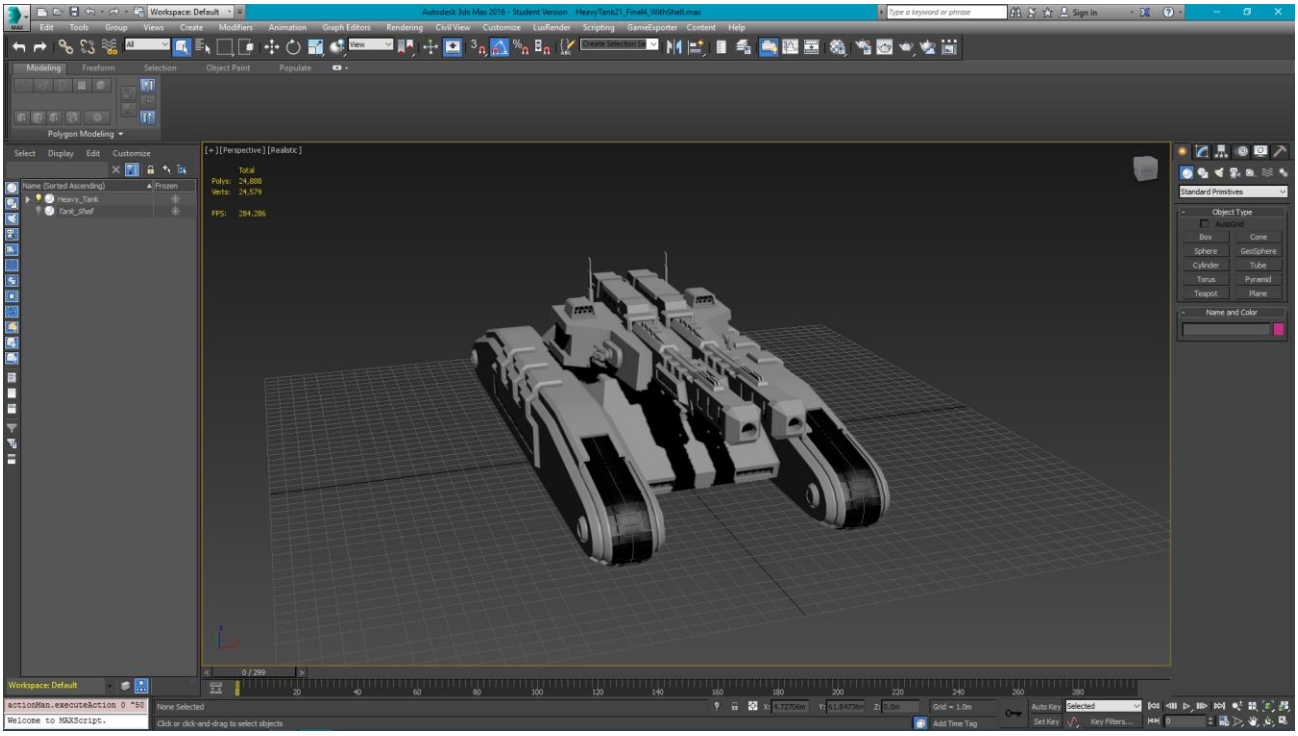
3. RAZVOJ IGRE

3.1. O igri

Igra je zamišljena kao znanstveno fantastična strategija u stvarnom vremenu s mogućnošću igranja i u trećem licu. Mečevi su zamišljeni da se igraju jedan-na-jedan. Svaki igrač ima kontrolu nad postrojbama svoga tima. Igrači započinju u pogledu iz ptičje perspektive, te imaju dostupno na zapovijedanje glavnu bazu, tvornicu za proizvodnju novih postrojbi i 3 tenka. Tijekom igre mogu praviti nove koje će im pomoć da pobjede meč. Svaki igrač na raspolaganju ima još jednu herojsku postrojbu koju koriste u trećem licu. Cilj igre je uništiti neprijateljsku bazu, te ujedno i spriječiti uništenje vlastite baze. Na tržištu postoje igre sa sličnim konceptom, među kojima su: *Armored Core Verdict Day* i *Divinity: Dragon Commander*.

3.2. Izrada elemenata igre

Za izradu elemenata igre, poput tenkova i zgrada, koristio se *3D Studio Max* (Sl. 3.1.) od *Autodesk*-a. *3DS Max* ima širok spektar alata specifično namijenjenih za izradu objekata (3D modela) za igre. *3DS Max* se inače kupuje na pretplatu, ali je moguće nabaviti studentsku verziju kojom se može program koristiti naredne 3 godine od dobivanja studentske verzije. Unutar *3DS Max*-a možemo izraditi kompleksne modele pomoću velikog spektra modifikatora (engl. *modifiers*), moguće je izraditi animacije pomoću trake za animaciju i grafa animacije itd. *3DS Max* ima mogućnost mapiranja UVW mapa koje se koriste prilikom izrade tekstura za modele čime se dodatno može unaprijediti izgled modela. 3DS Max ima alat za postavljanje tekstura koji dolazi u dva izdanja, jedan je za jednostavno postavljanje a drugi za napredno postavljanje tekstura.



Sl. 3.1. 3D Studio Max razvojno okruženje sa tenkom koji je korišten u projektu.

Korisničko sučelje u sredini sadrži pogled scene, uglavnom se koriste četiri pogleda: *Perspective* (pogled u prostoru), *Top* (pogled odozgo), *Left* (pogled s lijeva), *Front* (pogled sprijeda). Na slici 3.1. aktivan je samo pogled u prostoru. Alatna traka se nalazi iznad pogleda scene. Sadržaj alatne trake je moguće mijenjati ovisno o želji korisnika. S lijeva imamo hijerarhijski pogled na sve

objekte u sceni, ako je neki objekt podređen drugom objektu sve promjene na nadređenom objektu, kao što su skaliranje, će imati i utjecaj na podređene objekte. S desna imamo alate za dodavanje objekata u scenu npr. geometrijske oblike, kamere, osvjetljenja itd., modifikatore za oblikovanje modela, alate za animiranje i puno drugih mogućnosti. *3DS Max* ima mogućnost renderiranja scene kao jednu sliku ili niz slika, odnosno video. Modele je moguće uvesti (engl. *import*) iz drugih programa ako su podržanih formata, također je moguće snimiti (engl. *export*) modele u jedan od mnogih podržanih formata, kao što su FBX; OBJ; 3DS; WRL itd.

3.3. Glavni izbornik

Kada korisnik pokrene igru prvo što će vidjeti je glavni izbornik (Sl. 3.2.). Glavni izbornik je scena koja sadrži *Unity*-jevo korisničko sučelje (engl. *User Interface, UI*), objekte i kameru za renderiranje elemenata korisničkog sučelja. Glavni izbornik sadrži polje u koje se unosi ime igrača, izbornik za odabir tima i dugme za igranje odnosno potragu partije. U sceni postoji objekt nazvan „Launcher“ koji služi za postavljanje pojedinih postavki kao što su verzija igre, maksimalni broj igrača po meču, te reference na elemente korisničkog sučelja. Na objektu se nalazi skripta istog imena koja je dodana kao komponenta objekta. Skripta ima metodu „Connect“ povratnog tipa *void* , za spajanje na server. „Play“ dugme ima okidač koji prilikom klika pokreće „Connnect“ metodu na „Launcher“ objektu. Metoda prvo provjerava je li igrač već spojen na *Photon* mrežu (engl. *Photon Network*). Ako je igrač već spojen na server, onda mu *Photon Network* traži nasumičnu partiju, inače spaja igrača na *Photon* mrežu. Kada se igrač spoji na glavni server *Photon Network*-a poziva se funkcija „OnConnectedToMaster“ koja traži novu nasumičnu partiju za igrača.

Sl. 3.2. Izgled glavnog izbornika.

Nakon što se izvrši naredba (3-1) koja traži novu sobu, odnosno meč, poziva se funkcija „OnJoinedRoom“ ako je uspješno pronašao meč, inače se poziva funkcija „OnPhotonRandomJoinFailed“. Funkcije „OnJoinedRoom“, „OnPhotonRandomJoinFailed“ i „OnConnectedToMaster“ su definirane unutar *Photon.PunBehaviour* biblioteke. Ako želimo koristiti naveden funkcije unutar naše skripte, odnosno klase, potrebno je naslijediti *Photon.PunBehaviour* biblioteku. Potom je potrebno prepisat (engl. *override*) funkcije kako bismo implementirali željenu radnju u našoj skripti.

```
PhotonNetwork.JoinRandomRoom();                    (3-1)

PhotonNetwork.LoadLevel(“LevelName”);              (3-2)

PhotonNetwork.CreateRoom(parametri);               (3-3)
```

Pozivom funkcije „OnJoinedRoom“ izvršava se naredba (3-2) koja prvotno zaustavi slanje svih poruka preko mreže kako bi se učitala razina. U slučaju da je postavljena sinkronizacija scene (*automaticallySyncScene*) na *true* onda se učitava razina za sve igrače koji su u istoj prostoriji, odnosno partiji. Pozivom funkcije „OnPhotonRandomJoinFailed“ izvršava se naredba (3-3) koja će napraviti novi meč ovisno o predanim parametrima.

3.4. Glavna razina

Glavna razina je prostor u kojem igrači igraju jedan protiv drugog. Na razini postoji velik broj objekata kao što su zemljište, zgrade, lokacije za stvaranje postrojbi i najbitniji element: „TheGameMaster“ koji prati stanje meča, stvara igrače i njihove postrojbe, te javlja svima pobjednika ako je ispunjen uvjet za pobjedu.

3.4.1. TheGameMaster

Na „TheGameMaster“ objektu je postavljena *GameMaster* skripta koja nasljeđuje *Photon.PunBehaviour* i *IPunObservable* biblioteke. Skripta prilikom učitavanja izvršava naredbu (3-4) koja stvara gotov objekt (engl. *prefab*) na mreži, u ovom slučaju stvara novi objekt igrača svim igračima koji se nalaze u istoj partiji. Nakon stvaranja novog igrača ovisno o odabranom timu u glavnom izborniku, ponovo se poziva naredba (3-4) koja stvara postrojbe za spojenog igrača. Postrojbe koje se stvaraju se dodaju u *C#* listu objekata, „teamAActiveUnits*“* odnosno „teamBActiveUnits*“* ovisno o timu koji je igrač odabrao. Liste su tipa *GameObject,* a to je tip koji je definiran unutar *UnityEngine* biblioteke. Prilikom stvaranja postrojbi stvara se i glavna zgrada: „Headquarters“, čiju referencu spremamo u varijablu tipa *GameObject*: „teamA_HQ“ odnosno „teamB_HQ“ ovisno o odabranom timu igrača.

```
PhotonNetwork.Instantiate(“ObjektIgrača”, pozicija, rotacija, grupa);   (3-4)
```

Pomoću „Update“ funkcije, koja se poziva svaki put kada se *render*-ira nova slika (engl. *frame*), unutar *GameMaster* skripte pratimo stanje varijabli „teamA_HQ“ i „teamB_HQ“. Ako je uništena jedna od glavnih zgrada u igri, jedna od navedenih varijabli će imati referencu na *null* tip podatka, te kada dođe do te pojave znat ćemo da je jedan od igrača pobijedio partiju. Igrače se obavještava o rezultatu partije i na kraju ih se vrati na glavni izbornik. Funkcija „OnPhotonPlayerConnected“ je dio *Photon.PunBehaviour* biblioteke i prima jednu vrijednost tipa *PhotonPlayer*. Kako bi ju koristili potrebno ju je prepisat (engl. *override*). Funkcija izvršava naredbu (3-5) koja predaje vlasništvo nad „TheGameMaster“ objektom igraču koji se posljednji spojio u partiju.

```
photonView.TransferOwnershif(other)                                    (3-5)
```

Kako bismo sinkronizirali objekte na mreži između igrača potrebno je dodati komponentu *PhotonView* na objekt koji se želi sinkronizirati. Kako bismo mogli izravno pristupiti *PhotonView* komponenti na objektu kao što pristupamo u *GameMaster* skripti, potrebno je naslijediti *Photon.PunBehaviour* ili *Photon.MonoBehaviour* biblioteku. Komponenta *PhotonView* omogućava dodjeljivanje vlasništva nad objektima, jedinstveni identifikacijski broj objekta, opcije promatranja i sadrži listu praćenih komponenti. Vlasništvo nad „TheGameMaster“ objektom se prenosi s razlogom dijeljenja podataka o stanju postrojbi i glavnih zgrada igrača. Kako bi ostvarili komunikaciju o stanju navedenih objekata potrebno da skripta nasljeđuje *IPunObservable* biblioteku i ima definiranu funkciju „IPunObservable.OnPhotonSerializeView“ koja prima dvije vrijednosti tipa *PhotonStream* i *PhotonMessage*. Pravo na pisanje u tok (engl. *stream*) imaju samo vlasnici nad objektom. Ujedno je i potrebno dodati komponentu *GameMaster* u listu praćenih komponenti, *PhotonView* komponente. *PhotonStream* sadrži podatke iz toka ili upisuje naše podatke u tok, a podaci koji se šalju su svi podaci koji su dio *PhotonView* komponente.

Za slanje podataka u tok koristimo naredbu (3-6), a za primanje podataka iz toka koristimo naredbu (3-7). Podatak koji se šalje u tok je jedinstveni identifikacijski broj objekta koji je dodijeljen objektu prilikom njegovog stvaranja na mreži. Nakon primanja identifikacijskog broja, tražimo objekt koji imaju primljeni identifikacijski broj i dodajemo u odgovarajuću varijablu. Time se sinkroniziraju vrijednosti između igrača i oba igrača imaju isti prikaz, odnosno stanje na svojoj strani igre. Vrijedi napomenut da se u tok mogu poslati samo određeni tipovi podataka, npr. *int, float, string, Object[], bool* itd.

```
stream.SendNext(podatak);                                    (3-6)
```

```
(ocekivani_tip_podatka) stream.ReceiveNext();                (3-7)
```

3.4.2. RTS_Player

Igrači koji su se spojili u partiju imaju kontrolu nad vlastitim objektom zvanim „RTS_Player“. Navedeni objekt na sebi ima niz komponenti koje nude igraču mogućnost kretanja u prostoru, upravljanje postrojbama, te komponente za prikazivanje i interakciju s korisničkim sučeljem.

Skripta *RTS_PlayerController* prilikom stvaranja objekta igrača, stvara i korisničko sučelje igrača. Unutar „Update“ funkcije skripte, prvo provjeravamo je li objekt pod našim vlasništvom pomoću naredbe (3-8). Provjera je potrebna kako bi izbjegli miješanje inputa između igrača. Npr. nemamo provjeru pomoću naredbe (3-8) i prvi igrač počinje crtati pravokutnik za selekciju postrojbi na svom ekranu. Zbog tog što se ne provjeravamo vlasništvo objekta igrača, dolazi do crtanja pravokutnika i na ekranu drugog igrača. Skripta pruža mogućnost kretanja u prostoru (gore, dolje, lijevo, desno) korištenjem tipki: „W“, „A“, „S“, „D“. Također omogućuje kretanje u prostoru dovođenjem miša na određeni kraj ekrana. Npr. dovedemo li pokazivač na lijevi kraj ekrana kretat ćemo se u lijevo.

```
photonView.isMine                                                    (3-8)
```

Skripta *RTS_UnitControl* omogućava zadavanje naredbi postrojbama koje su pod vlasništvom igrača. Ovisno o odabranoj postrojbi pružaju se odgovarajuće naredbe za naređivanje i mijenja korisničko sučelje za odgovarajuće naredbe. Postrojbama je moguće narediti metu za napadanje, lokaciju prema kojoj da se kreće, patroliranje i zaustavljanje kretnje. Zgradama je moguće postaviti proizvodnju novih jedinica, zaustaviti proizvodnju, nastaviti proizvodnju i postaviti mjesto prema kojem se nove postrojbe trebaju kretati.

Skripta *RTS_UnitSelection* omogućava selekciju postrojbi. Postrojbe je moguće odabrati na 3 načina: jednim klikom odabire se jedna postrojba , povlačenjem miša po ekranu crta se pravokutnik za odabir postrojbi i dvoklikom na neku postrojbu odabiru se sve postrojbe istog tipa koje se nalaze na ekranu. Sve odabrane postrojbe se dodaju u listu tipa *GameObject*. Ako je odabrano više od jedne postrojbe, odabir se sortira po određenim kriterijima kao što su: kojem timu pripada postrojba, ime postrojbe, tip postrojbe itd. Radi jednostavnijeg sortiranja koristi se *LINQ* biblioteka koja dozvoljava vrlo jednostavno sortiranje kompleksnog niza podataka.

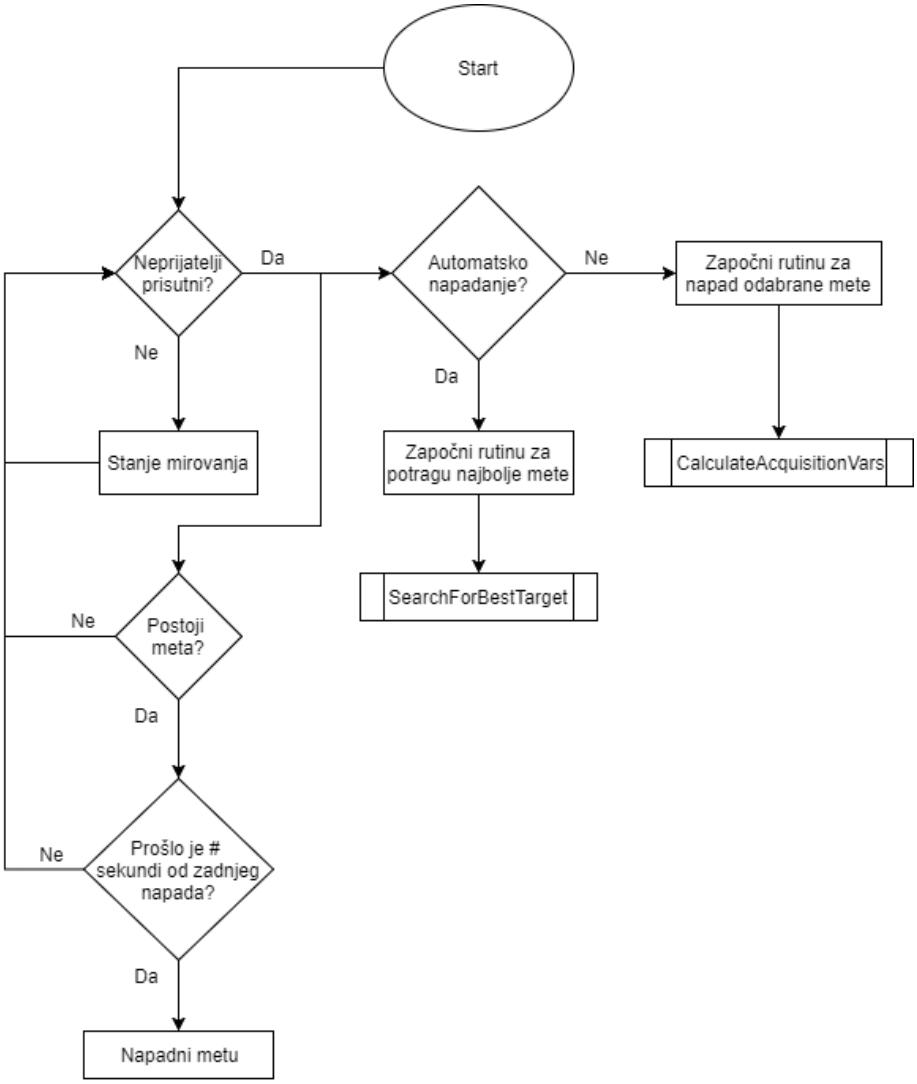
Skripta *UI_UnitSelectionInfo* služi za pravilno prikazivanje odabranih postrojbi na korisničkom sučelju igrača dok skripta *UI_SelectionBox* služi za prikazivanje pravokutnika selekcije.

3.4.3. Postrojbe

U igri postoje dva tipa postrojbi: zgrade (engl. *Building*) i jedinice (engl. *Unit*). Oba tipa postrojbi na svojim glavnim objektima imaju skriptu *UnitStats*. Skripta definira osnovne karakteristike

postrojbe kao npr. koliko životnih bodova ima, koliko je jak štit, koliko štete rade itd. Skripta također ima zadaću uništavanja postrojbe ako životni bodovi postrojbe padnu na nula ili manje od nula. Skripta ima referencu na grafičko korisničko sučelje, koje se nalazi iznad svake postrojbe, u svrhu prikazivanje trenutnog stanja životnih bodova i štita postrojbe igračima.

Zgrade na sebi drže još skriptu *BuildingBehaviour* koja definira moguće radnje zgrade. Skripta omogućava izradu novih postrojbi prilikom primanja naredbe za izradu nove postrojbe od strane igrača. Pri primitku naredbe za izradu nove postrojbe, željena postrojba za izradu se dodaje u niz izgradnje. Unutar *Unity*-jevog i*nspector viewa* moguće je postaviti maksimalni kapacitet niza za izgradnju postrojbi. Unutar skripte se pozivaju animacije, odnosno isječci animacija koji su unaprijed napravljeni unutar *3DS Max* alata. *Unity*-jeva *Animation* klasa pruža mogućnosti za manipuliranje navedenih isječaka. Pomoću navedene klase možemo „stopiti“ razne animacije kako bi dobili fluidan prijelaz iz jednog stanja u drugi. Npr. Prijelaz iz stanja mirovanja heroja u stanje trčanja i obratno.



Sl. 3.3. Dijagram tijeka umjetne inteligencije postrojbi unutar „Update“ funckije *TargetAcquisition* skripte.

Trupe na sebi imaju skriptu *AI_MovementController*, *TargetAcquisition* (Sl. 3.3.) i *RangeOfUnitBasic* ili *RangeOfUnitDrop*. Skripta *RangeOfUnitBasic* se koristi u slučaju kada se projektil kreće pravocrtno u prostoru, a *RangeOfUnitDrop* kada je projektil pod utjecajem sile teže. Skripte *RangeOfUnit* definiraju udaljenost pri kojoj će postrojbe zabilježiti prisutnost neprijateljske postrojbe. Znači prilikom ulaska neprijateljske postrojbe u prostor zapažanja postrojba zabilježi neprijatelja u niz prisutnih neprijateljski postrojbi. Navedeni niz se nalazi u skripti *TargetAcquisition* koja ima zadaću izvršavanja napada na odgovarajuću neprijateljsku postrojbu. Prilikom stvaranja postrojbe skripta *TargetAcquisition*, ovisno o zadanim vrijednostima u *UnitStats* skripti, dodaje

skriptu *RangeOfUnitBasic* ili *RangeOfUnitDrop*. Unutar „Update“ funkcije *TargetAcquisition* skripte definirano je djelovanje postrojbe za tri slučaja, odnosno njezina umjetna inteligencija (*Artificial inteligence, AI*). Prilikom dodavanja prvog neprijatelja u listu mogućih neprijatelja za napadanje, poziva se rutina „SearchForBestTarget“ koja automatski napada najbolju moguću metu u listi mogućih neprijatelja. U slučaju da primi naredbu za napadanje određene neprijateljske postrojbe, započinje se rutina „CalculateAcquisitionVars“, a ako nema moguće mete za napadanje onda se vraća postrojba u stanje mirovanja. Skripta ima zadaću rotiranja kupole i topova prema meti koju napada te ispaljivanje projektila na metu.

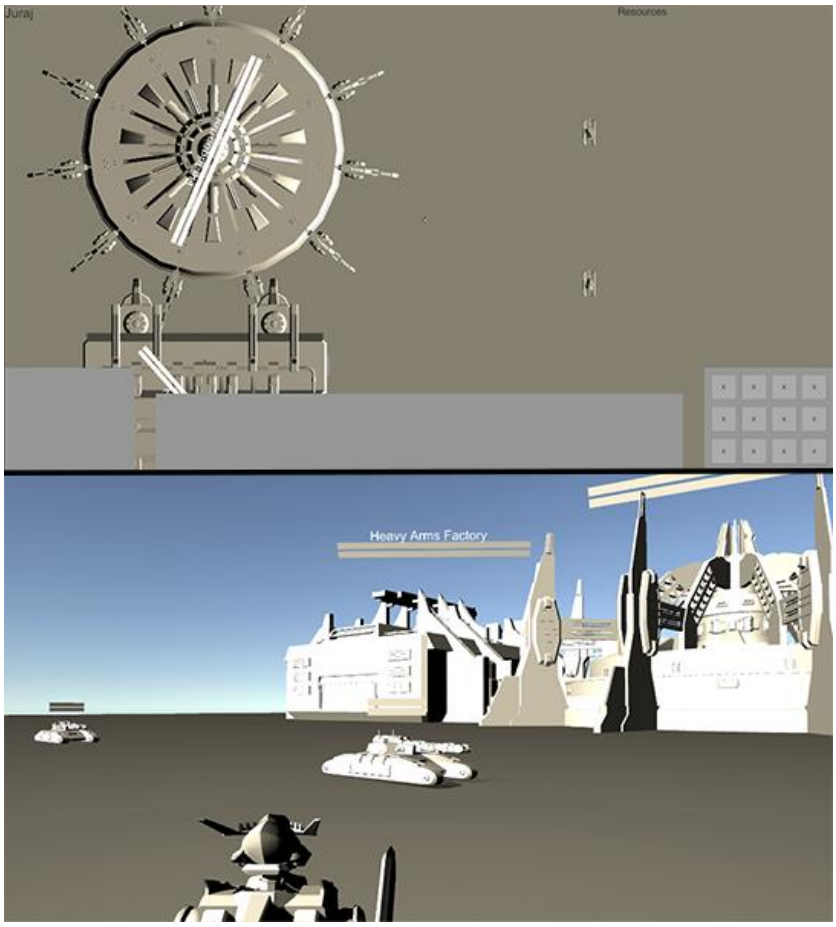
Funkcije „SearchForBestTarget“ i „CalculateAcquisitionVars“. se pozivaju pomoću naredbe (3-9). Naredba StartCoroutine omogućava izvršavanje naredbe na drugoj niti procesora. Navedene funkcije će se izvršiti unutar „Update“ funkcije ovisno o kriterijima koji su prikazani u dijagramu tijeka na slici, Sl. 3.3. Funkcija „Update“ se izvršava pri svakom novom crtanju slike, što ne bi bilo optimalno za kompleksne račune i sortiranja koja se vrše unutar „SearchForBestTarget“ i „CalculateAcquisitionVars“. Iz tog razloga se navedene funkcije vrše na odvojenoj niti, svakih 0.1 sekundu, kako bi optimizirali rad igre, odnosno rad skripte.

```
StartCoroutine("SearchForBestTarget");                    (3-9)
```

Potraga za najboljom metom se odvija unutar *AI_Targeting* skripte. Skripta čuva listu meta formatiranih u vlastito izrađenoj klasi *Targetz*. Klasa *Targetz* čuva reference na metu, moguće točke napada itd. Svaki put kada se izvrši funkcija „SearchForBestTargets“, provjeri se udaljenost između postrojbe i mete, koliko napadača ima meta i je li moguće napasti metu s trenutačne pozicije. Ovisno o rezultatu prijašnje navedenih uvjeta objektu se predloži najbolja moguća meta za napadanje, te se proslijede potrebne vrijednosti rotacije topova i kupole kako bi se izvršio uspješan napad na danu metu. Na sličan način funkcionira „CalculateAcquisitionVars“ funckija, koja ne provjerava čitav niz mogućih meta za napad, već samo metu koju je zadao igrač. Ako je moguće napasti zadanu metu predat će se vrijednosti za rotaciju topova i kupole, a u slučaju da nije moguće onda će se narediti postrojbi da se približi meti kako bi se mogao izvršiti napad na zadanu metu.

4. PERSPEKTIVE IGRAČA

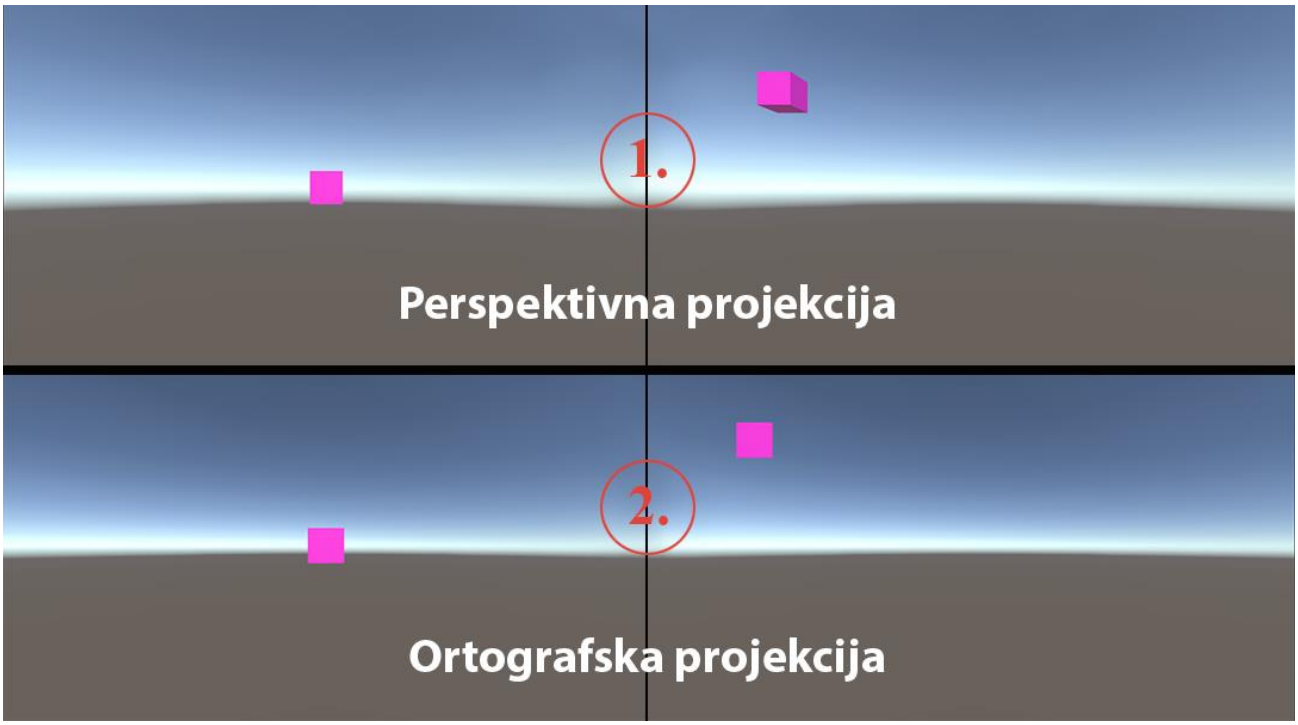
Jedan od zadataka ovoga projekta je omogućiti prebacivanja iz 2D u 3D pogled i obratno. Najjednostavniji način za ostvarivanje prijelaza iz 2D pogleda u 3D pogled i obratno je pomoću promjene projekcije objekata u sceni, tj. igri. Prilikom ulaska igrača u glavni nivo igre on započinje u pogledu ptičje perspektive (Sl. 4.1), to je pogled u kojemu je kamera orijentirana prema površini. Ovakav tip pogleda je često korišten u strateškim i akcijskim igrama. Poznatije igre koje koriste ovakav tip pogleda su *StarCraft*, *Civilization V*, *Bastion* itd. Tijekom igre moguće je promijeniti pogled igrača. Kada igrač prebaci pogled na heroja kamera se pozicionira iznad desnog ramena heroja, takav tip pogled je poznat kao pogled iz trećeg lica (Sl. 4.1).



Sl. 4.1. Gornja slika prikazuje pogled iz ptičje perspektive, a donja slika prikazuje pogled iz trećeg lica nakon što igrač preuzme kontrolu nad svojim herojem.

4.1. 3D projekcija

3D projekcija je matematička transformacija točaka iz trodimenzionalnog prostora na dvodimenzionalnu ravninu, u našem slučaju zaslon računala. Ovaj proces je poznat kao proces *render*-iranja u računalnoj terminologiji. Postoje dva načina projiciranja prostora na plohu, a to su ortografsko i perspektivno projiciranje. Unutar igre se koriste oba načina projiciranja. Ortografsko projiciranje prostora se koristi prilikom igranja u strateškom pogledu (Sl. 4.1. gornja slika), dok se perspektivno projiciranje prostora koristi prilikom kontroliranja heroja (Sl. 4.1. donja slika).



Sl. 4.2. Primjer perspektivne i ortografske projekcije. Oznake: 1. Perspektivna projekcija, 2. Ortografska projekcija.

Ortografsko projiciranje (Sl. 4.2. oznaka 2.) je način projiciranja trodimenzionalnih objekata u dvije dimenzije. Kod ortografskog projiciranja projekcijske zrake su međusobno paralelne i sve točke promatranog predmeta projiciraju se ortogonalno (pod pravim kutom) na projekcijsku ravninu. Ovaj tip projekcije se uglavnom koristi pri izradi tehničkih crteža.

Perspektivno projiciranje (Sl. 4.2. oznaka 1.) je način prikazivanja objekata na plohi slike (na ekranu), stvaranje predodžbe o dubini prostora odnosno način na koji ljudsko oko vidi prostor. Objekti

koji su udaljeniji od oka će biti manja naspram objekata koji su bliži oku. Ljudsko oko u *Unity*-ju je predstavljeno kao glavna kamera (engl. *main camera*) i njoj možemo pristupiti pozivom na varijablu „main“ unutar *Camera* klase *UnityEngine* biblioteke.

Na slici Sl. 4.2. možemo vidjeti primjere ortografskog i perspektivnog projiciranja unutar *Unity*-ja. Sa slike možemo vidjeti da promatrani objekt (kocka) u ortografskom i perspektivnom načinu projiciranja su praktički isti u slučaju kada je objekt centriran. Prilikom pomicanja objekta u gornji lijevi kut može se opaziti razlika u projiciranju objekta. Perspektivna projekcija prikazuje objekt s efektom dubine prostora, dok je ortografska projekcija objekta ostala ista (osim pozicije objekta na ekranu).

4.2. Promjena perspektive

Promjena perspektive se vrši unutar skripte *RTS_PlayerController* koja se spominje u poglavlju 3.4.2. Kako bi igrač promijenio perspektivu potrebno je dodijeliti određenoj tipki (u ovoj skripti to je tipka „K“) funkciju promjenu perspektive. Potrebno je dodati i varijablu tipa *bool* koja će se koristiti kao prekidač za promjenu perspektive. Ako je pritisnuta tipka „K“ i varijabla „controlHero“ je *false*, tada će igrač prijeći iz strateškog pogleda na pogled heroja. Nakon što se izvrši dani uvjet, varijabla „controlHero“ se postavlja na *true* kako bi mogli prijeći na strateški pogled prilikom idućeg pritiska tipke „K“. Prilikom prijelaza na pogled heroja dobro bi bilo sakriti pokazivač, ograničiti prostor kretanja pokazivača i sakriti korisničko sučelje strateškog pogleda, jer nam nije prioritet naređivati postrojbe kada preuzmemo kontrolu nad herojem.

Kako bi to ostvarili potrebno je promijeniti vrijednosti varijabli „visible“ i „lockstate“ iz *Cursor* klase. Varijablu „visible“ postavljamo na *false*, kako bi prikrili pokazivač, a varijablu „lockstate“ postavljamo na vrijednost *CursorLockMode.Locked* (enumerator definiran unutar *UnityEngine* biblioteke), kako bi ograničili kretnje pokazivača na centar ekrana. Korisničko sučelje možemo sakriti korištenjem varijable „playerUI“ koja referencira korisničko sučelje igrača i koristimo metodu (4-1). Pomoću navedene naredbe deaktiviramo objekt u sceni, deaktivirani objekti neće biti iscrtan i skripte koje se nalaze na objektu neće se izvoditi. Promjenom pogleda s heroja na strateški pogled, potrebno je postaviti varijablu „visible“ na *true* i „lockstate“ na *CursorLockMode.Confined*, čime se pokazivač može kretati samo unutar prozora igre, te je potrebno pozvati metodu (4-1) s vrijednosti *true* kako bi prikazali korisničko sučelje strateškog pogleda.

```
playerUI.SetActive(false);                                    (4-1)
```

Promjenu pogleda vršimo koristeći varijablu „cameraWork“, koja referencira komponentu *CameraWorkRTS* na objektu igrača, i pozivom metode „SwitchFollow()“. Skripta CameraWorkRTS sadrži varijable „isFollowingRTS“, koja je tipa *bool*, i varijable „cameraTransform“, „heroTransform“, koje referenciraju *Transform* komponentu glavne kamere u sceni i *Transform* komponentu podobjekta koji služi kao pozicija za postavljanje kamere iznad desnog ramena heroja. Nakon poziva metode „SwitchFollow()“ mijenja se pogled sa strateškog na heroja, i obrnuto. Varijabla „isFollowingRTS“ služi kao prekidač, ovisno o vrijednosti navedene varijable kamera će se adekvatno pozicionirati. Kako bi promijenili način projiciranja iz dvodimenzionalnog ortografskog u trodimenzionalni perspektivni prikaz s dojmom dubine prostora, potrebno je promijeniti vrijednost varijable „orthographic“ na *true*, odnosno *false* ovisno o željenom prikazu. Varijabli „orthographic“ se može pristupiti preko *Camera* klase, te pozivom na „main“ varijablu koja referencira glavnu kameru u sceni.

5. ZAKLJUČAK

U završnom radu je obrađena tema za izradu igre za više igrača u *Unity* razvojnom okruženju i omogućiti prijelaz između 2D i 3D prikaza za igrača. Obrađena je tema kako implementirati *Photon* u *Unity* projekt u svrhu omogućavanja igranja dvaju ili više igrača jedan protiv drugog preko mreže. Objašnjeno je kako se stvaraju igrači i kako se ostvaruje komunikacija između igrača i kako pisati u tok određene podatke radi usklađivanja varijabli za sve igrače koji se nalaze u istom meču. Objašnjene su razne skripte koje se koriste kako bi se ostvario željeni rad postrojbi, promjena perspektive igrača, kontroliranje igrača itd. Dijelom je obrađen i *3DS Max* unutar kojeg su se izradili određeni elementi igre.

Unity se pokazao kao vrlo fleksibilna platforma zbog širokog spektra elemenata dostupnih na njihovom *Asset Store*-u, ali i velikom broju dokumenata dostupnih na internetu. Početnici mogu vrlo lako napraviti igru u *Unity*-ju ako prate dokumentaciju na stranici *Unity Technologies*-a i ako se koriste *Asset Store*-om čime mogu izbjeći pisanje kompleksnih skripti i izradu nekih drugih elemenata igre. Igra nije savršena i svakako je moguće doraditi puno dijelova igre, npr. potrebna je optimizacija koda za pretraživanje mogućih meta za napadanje, nema tekstura na postrojbama i zgradama. Jedan on najvećih problema je što PUN-ova *PhotonView* komponenta pruža maksimalno 1000 jedinstvenih identifikacijskih brojeva, što znači da je maksimalni broj postrojbi u meču 1000. To se može izbjeći implementacijom vlastitog dodjeljivanja identifikacijskih brojeva prilikom stvaranja postrojbi i okretanju više lokalnom pristupu igranja. Igrači bi u tom slučaju slali jedan drugom naredbe koje su zadali svojim postrojbama, ispaljuje li postrojba projektil, stanja izgradnje postrojbi itd., umjesto da se konstanto ažurira položaj postrojbi kao što je napravljeno u ovom projektu.

LITERATURA

[1] Unity (Game Engine) wikipedija, https://en.wikipedia.org/wiki/Unity_(game_engine), pristupljeno 30. lipnja 2017.

[2] Unity internet stranica, https://unity3d.com/

[3] Exit Games wikipedija, https://en.wikipedia.org/wiki/Exit_Games

pristupljeno 30. lipnja 2017.

[4] Photon internet stranica, https://www.photonengine.com/en-US/Photon

[5] E. Lavieri, Getting Started with Unity 5, Packt Publishing, Birmingham, Svibanj 2015.

SAŽETAK

Cilj završnog rada je bio izraditi igru za više igrača u Unity razvojnom okruženju s mogućnošću mijenjanja pogleda iz 2D u 3D tijekom igranja. Za svrhu tog projekta najbolje je odgovarala igra žanra strategije u stvarnom vremenu. Za izradu igre je preporučljivo imati osnovno znanje C# ili JavaScript jezika, jer određeni dijelovi koda zahtijevaju kompleksno upravljanje objektima i varijablama. Za potrebe projekta potrebno je poznavanje i nekog programa za izradu 3D modela: Preporuča se Blender, jer je besplatan i ima gotovo sve mogućnosti (ako ne i više), kao i 3DS Max koji je korišten u ovom projektu. Cilj igre ja da igrači unište bazu neprijateljskog igrača čime pobjeđuju meč.

Ključne riječi: Unity, Blender, 3DS Max, više igrača, C#, JavaScript, 2D, 3D

ABSTRACT

The goal of this finals paper is to develop a multiplayer video game in the Unity game engine and be able to switch between a 2D and 3D view during gameplay. To make this game it's recommended that you have a general knowledge of the C# or JavaScript programming language, because certain parts of the code require complex management of objects and variables. It's best that you also know how to use a 3D modeling software. It's recommended that you use Blender, because it's free and has almost the same capabilities (if not more) as 3DS Max which was used in this project. The goal of the game is for the players to destroy the enemy players base in which case they win the match.

Keywords: Unity, Blender, 3DS Max, multiplayer, C#, JavaScript, 2D, 3D

ŽIVOTOPIS

Juraj Štrekelj je rođen 28. travnja 1995. godine u Osijeku u kojem trenutno i živi. Završio je Osnovnu školu „Retfala“ u Osijeku, te je nakon završetka osnovnoškolskog obrazovanja upisao I. Gimnaziju u Osijeku. Nakon završene gimnazije upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.