SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

PETAR OBROVAC

RAZVOJ 3D RAČUNALNE VIŠEKORISNIČKE IGRE U UMREŽENOM NAČINU RADA

DIPLOMSKI RAD

Dubrovnik, rujan, 2018.

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

RAZVOJ 3D RAČUNALNE VIŠEKORISNIČKE IGRE U UMREŽENOM NAČINU RADA

DIPLOMSKI RAD

Studij: Poslovno računarstvo
Kolegij: Programsko inženjerstvo
Mentor: doc.dr.sc. Krunoslav Žubrinić
Student: Petar Obrovac

Dubrovnik, rujan, 2018.

REPUBLIKA HRVATSKA

SVEUČILIŠTE U DUBROVNIKU

ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

STUDIJ: Poslovno Računarstvo

Sveučilišni diplomski studij

Ur. broj: 62/18-EIR

Dubrovnik, 4. srpnja 2018.

Kolegij: PROGRAMSKO INŽENJERSTVO

Mentor: doc. dr. sc. KRUNOSLAV ŽUBRINIĆ

DIPLOMSKI ZADATAK

Kandidat: **PETAR OBROVAC**

Broj indeksa: 128-SDS/PR

Naslov zadatka: **RAZVOJ 3D RAČUNALNE VIŠEKORISNIČKE IGRE U UMREŽENOM NAČINU RADA**

DEVELOPMENT OF 3D COMPUTER MULTIPLAYER NETWORKED GAME

Opis zadatka: Opisati specifičnosti 3D računalnih višekorisničkih igara koje se igraju u umreženom načinu rada. Proučiti dostupna okruženja za razvoj 3D računalnih višekorisničkih igara u umreženom načinu rada. Opisati specifičnosti razvoja 3D računalne višekorisničke igre u umreženom načinu rada. Osmisliti, dokumentirati i praktično realizirati 3D računalnu višekorisničku igru koju u lokalnoj mreži može igrati više igrača na mobilnim uređajima.

Zadatak je uručen kandidatu 4. srpnja 2018.
Rok za predaju diplomskog rada je do 21. rujna 2018.

Mentor:

doc. dr. sc. Krunoslav Žubrinić

Pročelnica Odjela:

izv. prof. dr. sc. Marija Mirošević

Sažetak

Unity je višeplatformski (engl. *cross-platform*) grafički pogon razvijen od strane *Unity Technologies* za izradu 2D i 3D video igara. Pruža jednostavno *drag-and-drop* okruženje za kreiranje igara, u kojem je moguće napraviti jednostavnu igru uz minimalno programiranje. U ovom radu je prikazan proces izrade jednostavne višekorisničke igre. Objašnjeni su napredni *Unity* pojmovi, mrežne komponente i cijeli proces izrade jednostavne višekorisničke 3D igre.

Ključne riječi**:** Višekorisnička igra, Unity3D, Unity predlošci, 3D objekti, Unity Skripte, Unity komponente

Abstract

Unity is a cross-platform game engine developed by Unity Technologies for developing 2D and 3D games. The Unity Editor provides a drag and drop environment for creating games and it's possible to create a game in Unity almost without writing any code. This paper presents the process of making a simple multiplayer game. It explains the advanced concepts of Unity, network components and process needed for creating simple multiplayer 3D game.

Keywords**:** Multiplayer game, Unity3D, Unity prefabs, 3D objects, Unity scripts, Unity components

SADRŽAJ:

1. UVOD 1
2. Višekorisnička igra 2
2.1. Povijest 2
2.2. Usporedba višekorisničkih i jednokorisničkih igara 3
2.3. Arhitektura 4
2.4. On-line višekorisničke igre 5
2.4.1. Mogući problemi 5
2.5. Off-line višekorisničke igre 6
2.5.1. Neumreženo igranje 6
2.5.2. Umreženo igranje 8
2.6. Mobilne višekorisničke igre 10
3. Rješenja za razvoj višekorisničke igre u *Unity* sustavu 13
3.1. Unity networking 13
3.1.1. Unity HLAPI 13
3.1.2. Unity LLAPI 16
3.1.3. Internet Services 17
3.2. Photon Unity Networking 17
4. Razvoj igre 18
4.1. Okruženja za razvoj igre 19
4.1.1. Unity 20
4.2. Optimizacija igara 21
4.3. Unity programiranje 21
4.3.1. Komponente za višekorisničku igru 21
4.3.2. Sinkronizacija 24
4.3.3. Važniji koncepti programiranja koristeći C# skripte 26
4.4. Materijali, alati za sjenčanje i teksture 30
4.5. Svjetla 31
4.5.1. Vrste svjetla 31
4.5.2. Osvjetljenje 34
4.6. Modeliranje scene i objekata 35
4.6.1. Instalacija Blendera 36
4.6.2. Korisničko sučelje 37
4.6.3. 3D modeliranje 38
4.6.4. Kako dalje 41
5. Izrada igre 43
5.1. Trgovina elementima 43

5.2. Battle Cars 44
5.3. Izrada elemenata igre 45
5.3.1. Predložak za oružje 45
5.3.2. Predlošci za PowerUp 47
5.3.3. PowerUp generator 49
5.3.4. Kreiranje igrača 50
5.3.5. Kreiranje borbene arene 53
5.3.6. Glavni izbornik 55
5.4. Razvoj igre za različite platforme 61
5.4.1. Razvoj na osobna računala 62
5.4.2. Razvoj na mobilne platforme - Android 63
6. Korisnički priručnik 65
6.1. Instalacija igre 65
6.1.1. Instalacija igre na mobilne uređaje 65
6.1.2. Instalacija igre na desktop računala 65
6.2. Igranje 66
6.2.1. Battle Arena 68
6.2.2. Battle Race 68
7. ZAKLJUČAK 70
8. LITERATURA 72

1. UVOD

Računalne igre danas zauzimaju velik udio tržišta računalnog softvera. Razvoj grafičkih kartica i hardvera općenito omogućio je izradu vrlo opširnih, detaljnih i vizualno impresivnih igara koje korisnik može igrati na svom osobnom računalu. Iako postoje mnoge igre namijenjene jednom igraču (engl. *singleplayer*), danas sve veći dio tržišta zauzimaju i višekorisničke (engl. *multiplayer*) igre. Bilo koja igra koja omogućuje istovremeno igranje više od jednog igrača smatra se višekorisničkom igrom. U početku, višekorisničke igre su postojale samo kao dodatak igrama za jednog igrača. Zatim je došlo do razvoja prvih pravih višekorisničkih igara preko umreženih računala. Daljnjim razvojem tehnologija došlo je i do razvoja višekorisničkih igara. Razvojem Interneta osobito popularne su postale višekorisničke mrežne igre (engl. *on-line multiplayer*). Prema podacima preuzetih sa službene stranice tvrtke *Blizzard Entertainment* [1], *World of Warcraft* je jedna od najpopularnijih mrežnih igara svih vremena. Preko 100 milijuna ljudi iz 244 različite države je igralo igru, a trenutno ima oko 8 milijuna stalnih pretplatnika.

U ovome radu sam prikazao kompletan proces razvoja višekorisničke igre koristeći *Unity* razvojno okruženje. Rad ne sadrži osnove već opisuje naprednije stvari te je poželjno imati određeno predznanje u radu s *Unity* razvojnim okruženjem. Poznavanje *Unity* korisničkog sučelja te osnovnih pojmova uvelike olakšava razumijevanje ovoga rada. U drugom poglavlju je ukratko prikazan povijesni razvoj višekorisničkih igara, te su opisane specifičnosti 3D računalnih višekorisničkih igara koje se igraju u umreženom načinu rada. U trećem poglavlju su opisana moguća rješenja za razvoj višekorisničke igre kroz *Unity* razvojno okruženje. U četvrtom poglavlju sam analizirao dostupna okruženja za razvoj 3D računalnih igara s naglaskom na *Unity* okruženje koje je i detaljnije obrađeno. Osnovna arhitektura, modeliranje scene i grafičkih objekata te pristup razvoju takvih igara su također opisani u četvrtom poglavlju. Opis praktične realizacije 3D višekorisničke igre u umreženom načinu rada je tema petog poglavlja, a način instalacije i upute za korištenje realizirane igre navedeni su u šestom poglavlju. U zaključnom sedmom poglavlju dat je osvrt na realizaciju kao i mogućnosti buduće nadogradnje.

2. Višekorisnička igra

Od pojave prvih višekorisničkih igara mijenjali su se načini na koje se mogu igrati višekorisničke igre. Danas se višekorisničke igre najčešće igraju preko Interneta ali također se mogu igrati i preko lokalne mreže (LAN, WiFi) ili čak na istoj konzoli. *Split screen* višekorisničko igranje je popularno kod konzola, ali obično dopušta samo između dva i četiri igrača.

2.1. Povijest

Neke od prvih igara u dva igrača su bile *Tennis for two* (sportska igra razvijena 1958, koja simulira tenis) i *Pong* (sportska igra razvijena 1972, koja simulira stolni tenis) [2]. *Empire* je bila prva umrežena višekorisnička igra, razvijena za *PLATO* sustav 1973. godine [2]. Igra je podržavala do 32 igrača istovremeno, podijeljenih u dva tima, a cilj igre bio je osvajanje galaksije. Zatim dolazi do pojave igara koje su u osnovi igre za jednog igrača a dojam višekorisničke igre se ostvaruje tako da igrači naizmjenično igraju igru te pokušavaju ostvariti bolji rezultat. Tijekom godina, višekorisničke igre su sve više dobivale na popularnosti. Detaljan opis razvoja višekorisničkih igara kroz povijest je izvan opsega ovoga rada. Prema mojoj slobodnoj procjeni, neke od važnijih godina u daljnjem razvoju višekorisničkih igara su:

- 1971 ARPANET (*Advanced Research Projects Agency Network*), početak implementacije budućeg internetskog protokola.
- 1972 *eSport*, prvo službeno eSport natjecanje.
- Igre za 2 igrača koji naizmjenično igraju igru:
    - 1978 *Space Invaders;*
    - 1979 *Space Asteroids;*
    - 1980 *Pac Man.*
- 1983 Internet, 1.1.1983. ARPANET koristi TCP/IP protokol, početak modernog Interneta.
- 1987 *Midi Maze*, igra s kojom se *Atari* pridružuje višekorisničkom okruženju. Konzola *Atari ST* dopušta spajanje do 16 konzola. Prvi put se kao koncept igranja pojavio „*death match*“.
- 1993 prvi LAN party.
- 1993 *Doom*, smatran kao jedan od pionira pucačina u prvom licu (engl. *first-person shooter*). *Doom* je podržavao LAN igranje do 4 igrača.
- 1994 *Warcraft*, dva igrača igraju preko modema ili lokalnih mreža.

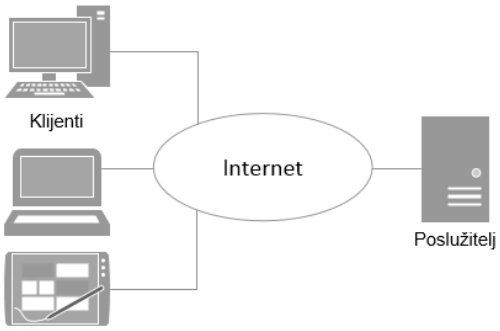
- 1997 *Golden Eye*, višekorisnički način igranja omogućuje igranje dva, tri ili četiri igrača koji se međusobno bore u *split screen* načinu igranja.
- 1997 *Ultima Online*, razvoj masivne višekorisničke igre.
- 1999 *EverQuest*, 3D masivna višekorisnička igra.
- 2000 *Counter Strike*, početak serijala višekorisničke pucačine iz prvog lica.
- 2001 pojava *Xbox*-a na tržištu.
- 2002 *Battlefield 1942.*
- 2004 *World of Warcraft,* igra je do danas stekla kultni status.
- 2006 *Nintendo Wii*, upotreba igračih palica osjetljivih na pokrete.
- 2011 *Minecraft.*
- 2014 *PlayStation 4* u prodaji.
- 2016 *Pokemon Go*, igra proširene stvarnosti (engl. *augmented reality*) za mobilne uređaje.
- 2017 *Fortnite*, višekorisnička igra koju trenutno igra preko 125 milijuna ljudi.

2.2. Usporedba višekorisničkih i jednokorisničkih igara

Prva i osnovna razlika između jednokorisničkih i višekorisničkih igara je broj igrača koji igraju igru. Jednokorisnička igra je namijenjena igri samo jednog igrača, dok je višekorisnička namijenjena igri više igrača. Jednokorisnička igra obično ima razrađenu cijelu priču, te se točno zna koji je krajnji cilj igre. Igrač slijedi priču istražujući, otkrivajući tragove, prelazeći nivoe, rješavajući zagonetke, uništavajući neprijatelje i slično. U slučaju borbe, igrač se bori protiv neprijatelja upravljanih od strane umjetne inteligencije. Ponašanje neprijatelja je relativno očekivano, baš kao i pozicija u nivou. Igrač zna kada i gdje može očekivati neprijatelja. Kod višekorisničkih igara priča nije toliko važna, a igra se obično svodi na neku vrstu natjecanja između igrača. Zbog toga je izrada jednokorisničke igre znatno teži zadatak. Višekorisničke igre se najčešće svode na kreiranje npr. neke vrste borbene arene gdje se više igrača natječe da bi porazilo sve druge igrače ili skupilo što više bodova. Takve arene obično imaju razne dodatke za oružje ili zdravlje te na taj način potiču igrače da aktivnije sudjeluju u igri. Za razliku od jednokorisničke igre i protivnika koji koriste umjetnu inteligenciju, višekorisnička igra je čista suprotnost jer se ne može predvidjeti ponašanje svakog pojedinca. Pojedinac može biti amater ali i pravi profesionalac. Višekorisničkim igranjem se ostvaruje komunikacija i interakcija s ostalim igračima. Stvaraju se klanovi, savezi ili timovi te se na taj način stvara i osjećaj zajednice.

2.3. Arhitektura

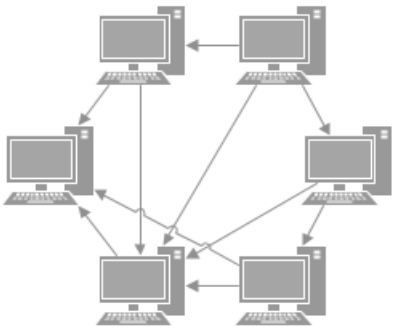
Višekorisničke igre su najčešće realizirane putem klijentsko – poslužiteljske (engl. *client - server*) arhitekture prikazane na slici 1, ili "svaki sa svakim" (engl. *peer-to-peer* - P2P) arhitekture prikazane na slici 2.



Slika 1. Klijentsko - poslužiteljska arhitektura

Klijentsko-poslužiteljska arhitektura je arhitektura gdje su klijent i poslužitelj odvojeni jedan od drugoga, a koja se sastoji od centraliziranog poslužitelja s više klijenata (računala ili neki drugi uređaji korisnika) koji su povezani s poslužiteljem. Poslužitelj je zadužen za optimalno upravljanje zajedničkim resursima, kontrolu pristupa te sigurnost podataka, a klijenti vrše upravljanje korisničkim sučeljem i izvršavaju dio logike. Pregledavanje internetske stranica je primjer klijentsko-poslužiteljske arhitekture. Korisničko računalo i web preglednik imaju ulogu klijenta i oni šalju zahtjeve, dok su računalo i baza podataka koji čine web stranicu poslužitelj koji poslužuje, odnosno odgovara na zahtjeve klijenta. Slična stvar je i kod *on-line* igara. Igrači sa svojom igraćom konzolom su klijenti, a *game server* je poslužitelj. Glavna karakteristika ove arhitekture je da se većina programskog koda igre izvodi na poslužitelju, dok klijent kontrolira samo manji dio koda. Komunikacija uključuje klijenta koji šalje zahtjev poslužitelju, poslužitelj taj zahtjev potvrđuje ili ne, a zatim poslužitelj prosljeđuje zahtjev svim ostalim klijentima.

P2P arhitektura je bila popularan izbor za realizaciju višekorisničkih igara tijekom 90-ih godina 20. stoljeća.



Slika 2. P2P arhitektura

Svi čvorovi (engl. *peer*) u P2P modelu su hijerarhijski jednaki, istodobno su i klijenti i poslužitelji te dijele podatke između sebe. Na slici 2 je prikazana P2P arhitektura gdje se vidi povezanost svih čvorova. Svaki čvor šalje informacije svim drugim čvorovima . Istovremeno svaki čvor prima informacije od svih povezanih čvorova.

2.4. On-line višekorisničke igre

On-line igre su igre koje se igraju preko Interneta. U samim počecima Interneta, uređaji su se spajali na Internet pomoću 14.4K modema koristeći običnu telefonsku liniju. U to vrijeme se zbog male brzine svega nekoliko igrača moglo povezati i igrati višekorisničku igru poput *Doom* ili *Duke Nukem 3D*. Nove tehnologije i širokopojasne veze pomogle su u populariziranju internetskih igara. Mnoge igre imaju mogućnost igranja *on-line* što omogućuje igračima da igraju igru protiv ili u suradnji s igračima širom svijeta. Današnje *on-line* višekorisničke igre mogu uključivati čitav virtualni svijet u kojemu se nalazi velik broj igrača istovremeno. Broj igrača se može kretati od svega nekoliko igrača pa sve do više stotina ili čak tisuća igrača. Takve igre se zovu masivne višekorisničke *on-line* igre (engl. *Massively multiplayer online games* - MMOG) i dijele se na:

- MMORPG (engl. *Massively multiplayer on-line role-playing game*) – u kojoj igrač preuzima ulogu virtualnog lika iz igre,
- MMORTS (engl. *Massively multiplayer on-line real-time strategy*) – strategija u realnom vremenu,
- MMOFPS (engl. *Massively multiplayer on-line first-person shooter*) – pucačina iz prvog lica,
- MMOSG (engl. *Massively multiplayer on-line social game*) – igra namijenjena druženju odnosno socijalnim aktivnostima.

World of Warcraft je najpopularnija MMOG igra s preko 8 milijuna mjesečnih pretplatnika širom svijeta [1].

2.4.1. Mogući problemi

Za što kvalitetnije igraće iskustvo kod igranja igara na Internetu, potrebno je da veza bude što bolja i brža. Brza internetska veza ne oslanja se samo na dobru brzinu preuzimanja (engl. *download*) i učitavanje (engl. *upload*), već i na vrijeme odziva. Vrijeme odziva, ili reakcijsko vrijeme, naziva se *ping*. *Ping* je mjera brzine veze ili, točnije, latencije veze. Ako

ping iznosi 100ms (milisekundi), to je vrijeme koje je potrebno za računalo da odgovori na zahtjev drugog računala. Stoga, poželjno je da *ping* vrijeme bude što kraće ili može doći do primjetnog i neželjenog kašnjenja u igri (engl. *lag*). U slučaju dužeg *ping* vremena, prvo i najjednostavnije rješenje je zatvaranje svih ostalih programa i otvorenih ekrana na računalu, tako da u pozadini ne bude aktivnih preuzimanja koja bi mogla utjecati na *ping*. Također, ako je na istu mrežu spojeno više uređaja koji aktivno upotrebljavaju internetsku vezu (npr. gledanje video sadržaja visoke razlučivosti (engl. *high density*)), to će biti i veći *ping*. Ako je i dalje *ping* vrijeme dugo, može se kontaktirati pružatelja internetskih usluga da dodatno provjeri vezu. Igranje preko Interneta pruža i određeni stupanj anonimnosti što također može negativno utjecati na kvalitetu igre.

2.5. Off-line višekorisničke igre

Off-line višekorisničke igre obično zahtijevaju od igrača da dijele resurse jednog igraćeg sustava ili koriste mrežnu tehnologiju kako bi se povezali i zajedno igrali preko veće udaljenosti. Dijeljenje jednog sustava se još naziva i neumreženo (engl. *non networked*) igranje, a povezano igranje se još naziva i umreženo (engl. *networked*) igranje.

2.5.1. Neumreženo igranje

U ovu kategoriju spadaju one igre gdje više igrača igra istu igru preko jednog uređaja. Na ovaj način se preko jedne igraće konzole i jednog primjerka igre ostvaruje višekorisnička igra.

Split screen

U *split screen* načinu igranja zaslon je podijeljen na dva ili više jednakih dijelova, obično dva do četiri. Kroz povijest, ovakav način igranja je bio popularan na konzolama koje nisu imale pristup Internetu ili lokalnoj mreži.

Pitstop II [3] je prva 3D igra koja je omogućila istovremeno igranje dva igrača preko *split screena*. Ekran je podijeljen u dva dijela. Gornji dio je za prvog igrača, a donji dio za drugoga. Cilj igre je pobijediti u utrci.

Slika 3. *Pitstop II* [3]

Igru *Pitstop II* je izdala izdavačka kuća *Epyx* za *Commodore 64* i *Atari* konzole davne 1984. godine. Iako ovo nije najpopularniji višekorisnički način igranja, *split screen* igre se i dalje razvijaju. Tako osim natjecanja između dva igrača, postoji i mogućnost suradnje u igri (engl. *cooperative games*) dva ili više igrača . Popularan naziv za ovakvu vrstu igre je i kauč suradnja (engl. *couch co-op*). Takav način omogućuje igračima da igraju zajedno kao tim, obično protiv jednog ili više protivnika. Jedan od primjera je igra *Star Wars: Battlefront* od izdavača *EA* [4]. U toj igri je jedan od načina *split screen* gdje se još može odabrati žele li dva igrača surađivati u misijama protiv zajedničkog neprijatelja ili se žele međusobno boriti.

Slika 4. *Star Wars Battlefront* [4]

Hotseat

Hotseat način igranja se odnosi na igre gdje dva ili više igrača na istom uređaju igraju igru na poteze (engl. *turn-based*). Prvi igrač odigra svoj potez, zatim je red na drugog igrača te se tako naizmjence igrači izmjenjuju. Najjednostavniji primjer igre na poteze je šah. Kroz povijest, jedne od popularnijih *hotseat* igara su *Heroes of Might and Magic* serijal [5] izdavačke kuće *Ubisoft* i serijal *Sid Meier's Civilization* .

Slika 5. *Heroes of Might and Magic* [5]

2.5.2. Umreženo igranje

Prije nego što je Internet postao popularan i pristupačan, višekorisničke igre su koristile razne oblike lokalnih mreža za prijenos informacija između igrača.

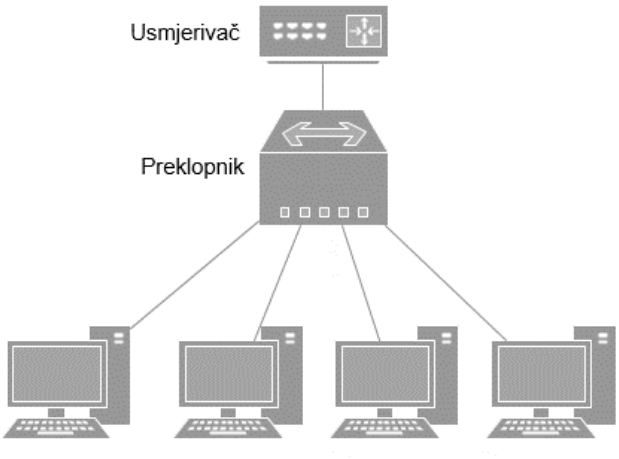
Arhitektura

Računalna mreža je digitalna mreža koja omogućuje uređajima dijeljenje resursa. Računala međusobno razmjenjuju podatke preko podatkovne veze. Podatkovna veza može biti realizirana putem žice ili optičkog kabela, a može biti realizirana i bežičnom vezom kao što su WiFi ili Bluetooth.

LAN

Lokalna mreža (engl. *local area network* - LAN) je mreža koja spaja uređaje na ograničenom području koristeći *Ethernet* tehnologiju. *Ethernet* predstavlja skup mrežnih

računalnih tehnologija primijenjenih unutar lokalnih mreža. Uređaji su spojeni žicom, a podaci se prenose električnim signalima. Dva uređaja se mogu spojiti izravno preko uvrnute parice (engl. *twisted pair*), a za spajanje više uređaja potrebno je koristiti neki od uređaja za umrežavanje kao što su preklopnik (engl. *switch*) ili usmjerivač (engl. *router*), kao što je prikazano na slici 6.



Slika 6. LAN topologija

Prilikom kreiranja višekorisničke igre preko LAN mreže, jedno računalo obično djeluje kao domaćin (engl. *host*) igre, a druga se računala onda povezuju s tim računalom. Kao domaćin se obično odabire najjače računalo jer se na takav način ostvaruje i najbolja veza. Igranje igara preko LAN mreže gotovo eliminira kašnjenje u igri koje je veliki problem kod online višekorisničkih igara. Popularan način igranja višekorisničkih igara je LAN okupljanje (engl. *LAN party*), na kojem bi igrači umrežili svoja računala i na taj način igrali višekorisničke igre. LAN može biti realiziran i preko P2P mreže.

WLAN

Bežična mreža (engl. *wireless network*) je lokalna mreža koja spaja uređaje bežično putem radio valova kao što je prikazano na slici 7. WLAN omogućuje korisnicima da se kreću oko područja pokrivenosti uz održavanje mrežne veze što je posebno važno za mobilne uređaje. Za izradu WLAN-a je potrebno imati pristupnu točku (engl. *access point* - AP) i jednog ili više klijenata. AP povezuje više klijenata u zajedničku grupu i služi za povezivanje sa žičanom mrežom ili s drugim bežičnim mrežama. Za povezivanje s drugim mrežama koristi se bežični usmjerivač (engl. *wireless router*) koji u sebi objedinjuje pristupnu točku i mrežni usmjerivač. Prednosti radio valova su:

- Mogu prelaziti velike udaljenosti

- Prodiru kroz zgrade i objekte
- Šire se u svim smjerovima od izvora

Neki od nedostataka su:

- Ovakva veza je sporija od žičane veze.
- Miješanje signala s ostalim elektroničkim uređajima.
- Dodatno zračenje.

Slika 7. WLAN mreža

Najveća prednost WiFi je prenosivost. Osobe koje koriste prijenosna računala ili ručne uređaje poput pametnih telefona i PDA uređaja mogu se prebaciti s jedne WiFi mreže na drugu jednostavnim odabirom mreže preko sučelja uređaja. Da bi bežična veza bila stabilna, potrebno je da se uređaj nalazi u blizini izvora signala jer je uobičajen radijus pokrivenosti signalom oko 10-20 metara. WiFi signali gube jačinu dok se udaljuju od antene, zbog čega se kvaliteta veze smanjuje dok se računalo ili uređaj nalazi dalje od izvora. Aplikacije za upravljanje WiFi vezama na računalima i drugim uređajima često imaju razinu za ocjenjivanje snage veze: odlična (engl. *excellent*), dobra (engl. *good*), loša (engl. *poor*).

2.6. Mobilne višekorisničke igre

Mobilne igre su igre koje se igraju na mobilnim ili pametnim telefonima koristeći tehnologije prisutne na samom uređaju. U industriji video igara već danas su mobilne igre zaslužne za najveći dio prihoda. Na slici 8 su prikazani prihodi na tržištu video igara po segmentima.

Slika 8. Tržište video igara – prihodi po segmentu [6]

U višekorisničkom okruženju, igrač surađuje ili se natječe s drugim igračima koji igraju istu igru na svom mobilnom uređaju dok su povezani putem mreže. Mreža može biti Internet ili lokalna mreže putem WiFi. Kod *on-line* mobilnog igranja postoje razne komunikacijske tehnologije:

- *Bluetooth*,
- *General Packet Radio Service* (GPRS),
- *Third generation* (3G),
- *Universal Mobile Telecommunications Services* (UMTS),
- *Wide Code Division Multiple Access* (WCDMA),
- *High-Speed Downlink Packet Access* (HSDPA).

Mnoge mobilne igre distribuiraju se besplatno krajnjem korisniku, ali sadržavaju plaćeno oglašavanje (engl. *paid advertising*) ili slijede *freemium* model, u kojem je osnovna igra besplatna, ali dodatne značajke igre moraju se kupiti zasebno.

Najveće ograničenje za *on-line* igranje je kašnjenje. Dok se kod PC *on-line* igranja kašnjenje mjeri u prihvatljivih nekoliko desetaka milisekundi, kod mobilnog igranja se može raditi i preko nekoliko stotina milisekundi, a u nekim slučajevima i čak nekoliko sekundi. Ako se igra online višekorisnička igra na poteze to i nije neki veliki problem ali ako se radi o igri u stvarnom vremenu onda je igra jednostavno neigriva. Zbog toga je glavni izazov kod izrade mobilne višekorisničke igre rješavanje problema kašnjenja. Uz visokopropusnu mrežu, važna stavka kod

izbjegavanja problema kašnjenja je i optimizacija same igre. Prije same izrade igre za mobilne platforme, potrebno je voditi računa o sljedećim stvarima:

- memorijska ograničenja, desktop računala su puno snažnija od mobitela pa treba voditi računa o optimizaciji
- kontrole korištene u igri, mobiteli su osjetljivi na dodir, nema miša i kursorskih tipki za upravljanje likovima u igri
- veličina zaslona mobitela je puno manja od monitora
- puno vrsta različitih mobilnih uređaja, različitih veličina i rezolucija

O optimizaciji je potrebno voditi računa od samoga početka izrade igre jer su mobilni uređaji znatno slabiji od računala. Uštede se mogu pronaći na više mjesta:

- Uštede u dizajnu igre
    - Što manji nivo detalja 3D objekata (engl. *Level Of Detail* - LOD)
    - Što je manje moguće objekata u jednom trenutku u sceni
    - Koristiti *bake* osvjetljenje umjesto osvjetljenja u stvarnom vremenu
    - Izbjegavati korištenje *Mesh* sudarača
    - Ograničiti upotrebu čestica
- Uštede kod programiranja
    - Koristiti *object pooling* umjesto *Instantiate* i *Destroy* funkcija
    - Ograničiti upotrebu skupih kalkulacija koje se izvršavaju svakog okvira

3. Rješenja za razvoj višekorisničke igre u *Unity* sustavu

Igre postaju sve bolje i kompleksnije, a mogućnost višekorisničkog igranja ih čini i mnogo zabavnijim. Višekorisničko igranje je moguće preko internetske veze i preko lokalne mreže. Igranje preko mobilnih uređaja je kompatibilno s *desktop* računalima te *iOS* i *Android* uređajima tako da igrači mogu igrati istu igru preko različitih platformi. Višekorisničko igranje kroz *Unity* može biti realizirano na više načina. *Unity* [7] dolazi sa svojim sustavom za umrežavanje, a postoje i gotova rješenja razvijena od treće strane (engl. *third-party*). Najpopularnije takvo rješenje je *Photon* [8], a neki od poznatijih su još i *uLink* [9], *Forge* [10], *PlayFab* [11] i *DarkRift* [12].

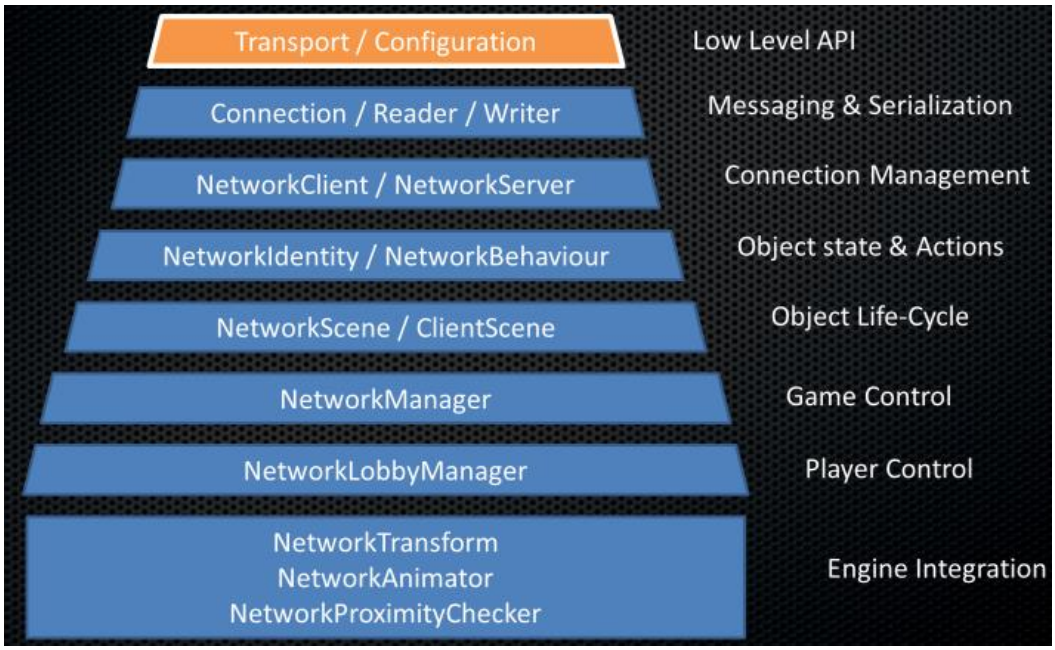
U ovom radu sam se odlučio za izradu višekorisničke igre koristeći *Unity* sustav za umrežavanje preko lokalne Wi-Fi mreže s posebnim naglaskom na igranje preko mobilnih uređaja.

3.1. Unity networking

Od *Unity* inačice 5.1 dostupna je nova značajka (engl. *feature*) za izradu umreženih igara, *Unity networking – UNet*. *UNet* je osmišljen kao zamjena za stari mrežni sustav koji je radio preko *NetworkView* komponente, a nudi skup prilagodljivih komponenti za kreiranje mrežne igre u stvarnom vremenu kroz *Unity*. *Unity* sustav za umrežavanje sadrži API niske razine (engl. *low level API*, LLAPI) i API visoke razine (engl. *high level API*, HLAPI). LLAPI je namijenjen izradi igara sa specifičnim potrebama koje ne zadovoljava *Unity* implementacija i naprednijim korisnicima koji žele izraditi vlastitu mrežnu infrastrukturu. Kako za potrebe ovog rada korištenje LLAPI nije bilo potrebno, daljnji rad fokusirat će se na HLAPI rješenju.

3.1.1. Unity HLAPI

HLAPI je sustav za dodavanje višekorisničkih mogućnosti igri. Oblikovan je u više razina na principu stoga u kojem svaki nivo dodaje neku novu funkcionalnost. Osnovu čini transportni nivo smješten na dnu stoga. HLAPI nivoi su prikazani na slici 9. Transportni sloj podržava bilo koju vrstu mrežne topologije, a HLAPI poslužitelj je autoritativan što znači da sva komunikacija prolazi preko poslužitelja, dok poslužitelj ne smije izravno utjecati na druge klijente bez prethodnog zahtjeva od strane klijenta za nekom promjenom.



Slika 9. HLAPI nivoi [13]

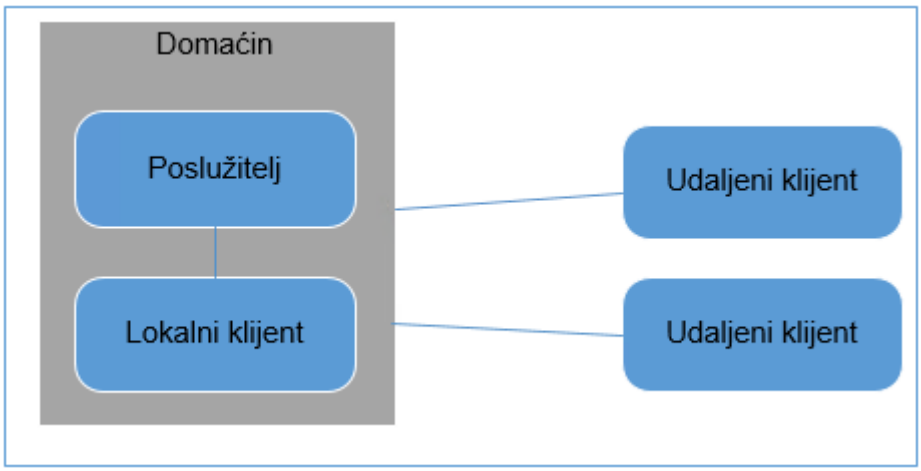
HLAPI sustav uključuje jedan poslužitelj i više klijenata:

- Poslužitelj predstavlja instancu igre na koju se svi igrači koji žele igrati igru spajaju. Poslužitelj pored toga upravlja i raznim drugim aspektima igre poput praćenja rezultata te slanje podataka o igri svim klijentima.
- Klijenti su instance igre koje se spajaju na poslužitelj s drugih uređaja, bilo preko lokalne mreže ili preko Interneta.

Poslužitelj može biti:

- Dedicirani poslužitelj (engl. *Dedicated server*) koji predstavlja instancu igre koja se ponaša samo kao poslužitelj.
- Domaćin poslužitelj (engl. *Host server*). Kada ne postoji dedicirani poslužitelj jedan od klijenata preuzima ulogu poslužitelja te postaje domaćin poslužitelj. Domaćin poslužitelj stvara jednu instancu igre koja se onda ponaša i kao klijent i kao poslužitelj.

Na slici 10 su prikazana tri igrača u višekorisničkoj igri. U igri se jedan klijent ponaša i kao poslužitelj te se takav igrač naziva lokalni klijent (engl. *local client*). Lokalni klijent se spaja na domaćin poslužitelj koji se nalazi na istom uređaju. Druga dva igrača su udaljeni klijenti (engl. *remote clients*) koji se nalaze na drugim uređajima te su spojeni na isti domaćin poslužitelj.



Slika 10. Realizacija klijentsko - poslužiteljskog modela u HLAPI sustavu

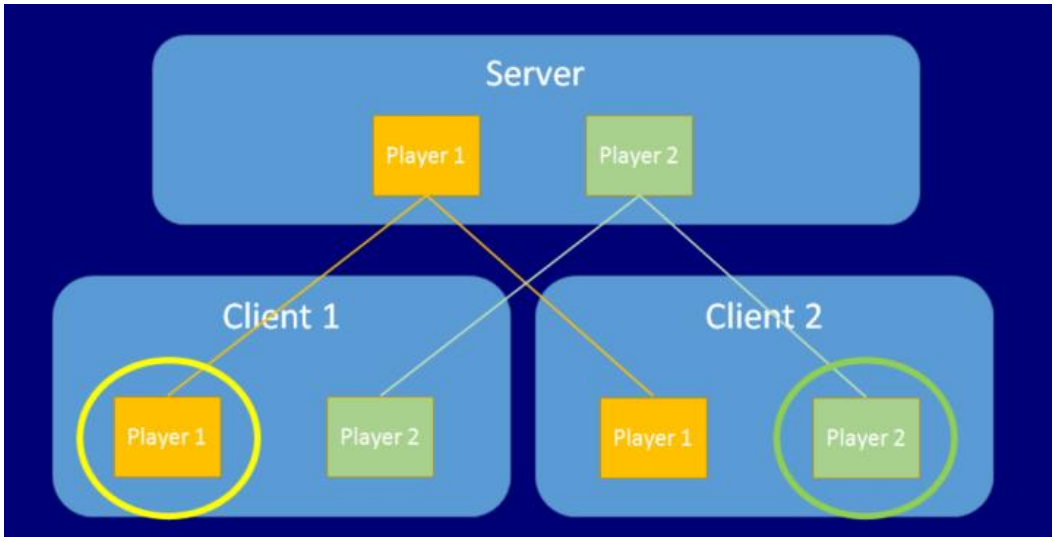
Sva se komunikacija između umreženih računala odvija preko poslužitelja. Poslužitelj može biti i domaćin - kombinacija klijenta i poslužitelja. Takav se klijent vodi kao lokalni klijent te dijeli proces s poslužiteljem. Zbog toga se poruke između poslužitelja i lokalnog klijenta ne razmjenjuju preko Interneta već pomoću redova poruka (engl. *message queue*). Treba napomenuti da je ovo interna razlika pa se HLAPI programski kod piše neovisno o vrsti klijenta. Ovo omogućuje jednostavniji razvoj igara koje nude opcije samostalnog i mrežnog igranja jer ne treba pisati odvojene skripte za svaki način rada.

Mrežni GameObject

Mrežni *GameObject* je onaj *GameObject* koji je upravljan i sinkroniziran preko *Unet* sustava. Upotrebom sinkroniziranih mrežnih *GameObjecata* se postiže višekorisničko igranje. Svi igrači istovremeno se vide međusobno te vide sve akcije i čuju sve zvukove i to sa svoje lokalne točke gledišta. Višekorisnička igra obično sadrži scenu koja sadrži mrežne *GameObjecte* i obične nemrežne *GameObjecte*. Mrežni objekti su svi oni koji se kreću tijekom igre ili na neki način mijenjaju, npr. igrači kojima se mijenja položaj u igri, zdravlje i municija. Sve takve objekte treba sinkronizirati. S druge strane, nemrežni objekti su oni objekti koji se ne kreću u igri te im se nikakve promjene ne događaju, ili se radi o objektima koje nije potrebno sinkronizirati, npr. razne prepreke, elementi okoliša i slično. Sinkronizacija se odrađuje preko *Unity* mrežnih komponenti koje ću opisati u poglavlju 4. Ukratko, svi mrežni objekti moraju sadržavati određene mrežne komponente.

Lokalni GameObject

Unity HLAPI sustav na različiti način tretira *GameObject* koji pripada igraču od onih koji mu ne pripadaju. Kada se novi igrač pridružuje igri tada *GameObject* toga igrača postaje lokalni igrač (engl. *local player*) na njegovom klijentu. *Unity* svakom igraču dodjeljuje jedan takav *GameObject*. Na taj način svaki igrač upravlja samo svojim *GameObjectom* i ne može se dogoditi situacija da jedan igrač izvršava naredbe drugoga igrača. Na slici 11 su prikazana dva klijenta, svaki sa svojim lokalnim igračem.



Slika 11. Lokalni igrač [13]

Oba igrača se nalaze na poslužitelju koji sinkronizira oba igrača tako da su vidljiva na oba klijenta. Međutim, na klijentu 1 kao lokalni igrač je označen samo igrač 1, a na klijentu 2 kao lokalni igrač je označen igrač 2. Višekorisnička podrška i sinkronizacija su ostvarene preko mrežnih komponenti koje su opisane u 4.3 poglavlju. Za lokalnog igrača je zadužena posebna klasa *NetworkBehaviour*, a općenito za podršku višekorisničke igre i kreiranje samoga igrača je zadužena komponenta *NetworkManager*.

3.1.2. Unity LLAPI

Za razliku od HLAPI koji je dizajniran tako da podržava minimum funkcionalnosti potrebnih za ostvarivanje višekorisničkog okruženja, LLAPI dopušta maksimalnu fleksibilnost i kompletnu kontrolu. LLAPI je pogodan za naprednije višekorisničke igre poput masivnih višekorisničkih igara u kojima sudjeluje veliki broj igrača. Općenito, LLAPI je pogodan za složenije mrežne igre, a HLAPPI za jednostavnije izvedbe. HLAPI i LLAPI nisu međusobno isključivi te se oba mogu istovremeno koristiti i kombinirati.

3.1.3. Internet Services

Unity omogućava komunikaciju između igara preko Interneta koristeći internetske usluge (engl. *Internet services*). Internetska usluga omogućuje korisnicima objavljivanje i oglašavanje igara, nudi popis dostupnih igara te pridruživanje postojećim igrama. Da bi koristili internetsku uslugu potrebno je prvo registrirati projekt preko Window->Unity Services ekrana ili izravno pristupajući web preglednikom adresi https://developer.cloud.unity3d.com/landing/ .

Kod *UNet* internetske usluge mrežni promet prolazi kroz relejni poslužitelj (engl. *relay server*) koji se nalazi u oblaku umjesto da ide izravno između klijenata. Kod besplatne verzije *Unityja*, postoji ograničenje od 20 istovremenih igrača (engl. *concurrent users* - CCU) u višekorisničkoj igri preko Interneta ali samo tijekom razvoja igre. Jednom kada prestane razvoj te igra krene u komercijalne svrhe, ta internetska usluga više nije besplatna.

3.2. Photon Unity Networking

Photon je višekorisničko okruženje za razvoj igre u stvarnom vremenu, masovne višekorisničke igre, *chat* i glasovni *chat* na raznim platformama, uključujući mobilne, PC i konzole. Razvijen je od strane *Exit Games* kompanije, utemeljene 2003. Jedan od proizvoda kojega su razvili je i *Photon Unity Networking* (PUN). PUN je *Unity* paket za podršku izrade i objave višekorisničkih igara. PUN podržava razvoj na svim platformama koje podržava i *Unity*, a postoje dvije verzije:

- PUN *Free*, besplatna verzija paketa s raznim demoima, gotovim raznim skriptama i dokumentacijom. Verzija podržava do 20 konkurentnih korisnika.
- PUN *Plus*, sve kao i kod besplatne verzije, a podržano je do 100 konkurentnih korisnika za *Photon Cloud*. Cijena je oko 90 eura za 60 mjeseci korištenja.

PUN zahtijeva *Unity* verziju 5.0 ili više, bez obzira radi li se o besplatnoj ili plaćenoj *Unity* verziji. Sve igre ili aplikacije razvijene preko PUN-a pokreću se preko centra za pohranu (engl. *hosting centre*) *Photon Cloud*. *Photon Cloud* centri za pohranu nalaze se u SAD-u, Europi, Aziji (Singapur, Japanu) i Australiji te pružaju nisko kašnjenje za igre diljem svijeta.

Instalacija PUN-a je dostupna preko sljedećih linkova:

- Službena *Photon* stranica: https://www.photonengine.com/en/pun
- *Unity* trgovina: https://assetstore.unity.com/packages/tools/network/photon-pun-12080

4. Razvoj igre

Razvoj igre obuhvaća kompletan proces izrade igre od samoga početka pa sve do završetka. Nije jednostavno definirati što bi bio početak, a što završetak procesa. Početak je obično ideja, najčešće u obliku koncepta. Završetak je izrađena igra, spremna za igranje. Za izradu jedne igre potrebna su znanja iz različitih područja. Razvoj igara može trajati i po nekoliko godina, a razvojni tim može biti nezavisan (engl. *indie developer*) ili komercijalan. Nezavisna igra je igra koja je razvijena bez financijske podrške izdavača i u manjim timovima, ili od strane samostalnih programera U takvom okruženju često pojedinac preuzima sve uloge razvojnog tima na sebe. Nezavisne igre bilježe porast popularnosti u drugoj polovici 2000-ih, prvenstveno zbog novih metoda distribucije na mreži i razvojnih alata [14].

Razvojni tim može da se sastoji od sljedećih uloga [14]:

- dizajner (engl. *designer*), zadužen za dizajn igre, igrivost, pravila i strukturu igre;
- umjetnik (engl. *artist*), zadužen za kreiranje 2D ili 3D sadržaja, tekstura, okruženja;
- programer, zadužen za programiranje programskog koda igre;
- tester, zadužen za testiranje kvalitete igre;
- stručnjak za zvuk, osobe zadužene za zvučne efekte, pozadinsku muziku i slično.

Početak svake igre je ideja. Ideja se može opisati kao koncept, slika ili na bilo koji drugi kreativan način. Ideje se prikupljaju i analiziraju. Loše ideje se odbacuju, a dobre prihvaćaju. Pretvaranje ideje u igru je ono što rade dizajneri igara. Oni konceptualnim elementima daju oblik. Kada dizajner ima dobru predodžbu ideje obično se kreira dokument dizajna igre (engl. *Game Design Document - GDD*). GDD pomaže dizajneru kako bi zapisao što želi napraviti i pokazuje drugim članovima tima što točno dizajner želi dizajnirati, kako bi svi bili usklađeni. Za manje igre to može biti običan list papira s jednostavnim opisom igre. GDD također može sadržavati i sve alate koji će se koristiti pri izradi igre. Koristeći alate definirane u GDD, umjetnici kreiraju sadržaj, a programeri pišu programski kod.

Prije same izrade igre treba imati ideju o kakvoj će se igri uopće raditi. Je li 2D ili 3D? Koji je žanr? Radnja? Koja je ciljana skupina ljudi za koju se igra razvija? Možda najvažnija stvar je da igra bude zanimljiva i igriva tako da privuče ljude na igranje. Razmišlja se i o platformi na kojoj će se igra razvijati i pokretati.

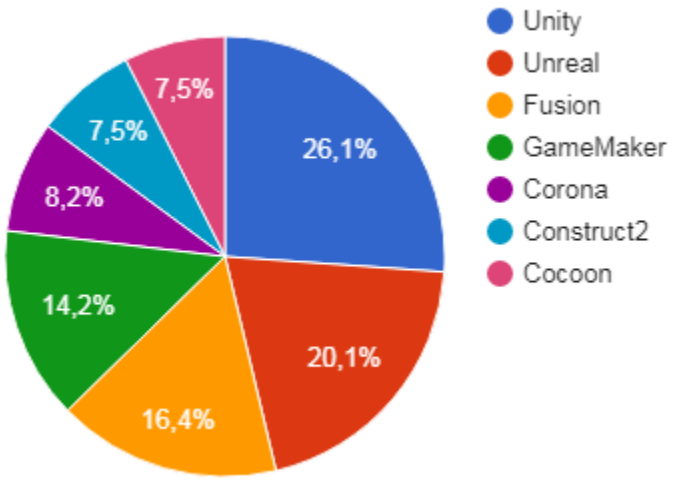
Takve stvari su dio koncepta igre (engl. *Game concept*). Koncept igre svom najjednostavnijem obliku je vizija igre koja se stavlja na papir na jednostavan i razumljiv način tako da svi koji su uključeni u igru znaju točno o čemu se radi i što će sve biti potrebno da bi se igra napravila. To znači da će izdavač, investitori, umjetnički timovi, programeri, pa čak i

trgovci znati što će se od njih očekivati i koji će biti njihov doprinos razvoju igre. Koncept igre trebao bi slijediti faze kreiranja igre, od priče do umjetnosti, pa čak i kako zaraditi novac igrom. Prva faza koncepta je sama ideja. Ukratko se opiše radnja ili priča oko koje se igra događa. Ideja se zatim širi, priča se sve više razvija, opisuju se radnja i likovi uključeni u igru.

4.1. Okruženja za razvoj igre

Mnoge igre se temelje na komercijalnim, a u zadnje vrijeme i besplatnim okruženjima za razvoj igre. Neka od okruženja su: *Unreal* [15], *Unity* [7], *CryEngine* [16], *Construct* [17], *PlayCanvas* [18], *Godot* [19], *GameMaker* [20], a postoje još i mnoga druga okruženja. Da bi se odabralo ispravno okruženje, prvo treba znati koje ideje o samoj igri imamo. Koje platforme planiramo ciljati? Koji su podržani programski jezici? Kakvi su nam financijski uvjeti za izradu igre? Da li se igra radi za komercijalne svrhe? Ima sigurno još mnogo pitanja koja bi se mogla postaviti. Nakon što odgovorimo na postavljena pitanja tek onda bi trebalo početi analizirati dostupna okruženja.

Moja igra će biti višekorisnička 3D igra utrkivanja automobila. Osim na stolnim računalima, planiram igru igrati i preko mobilnih uređaja s Android operacijskim sustavom. Na slici 11 je prikazana popularnost različitih okruženja za razvoj mobilne igre u 2018. godini. Od programskih jezika upoznat sam s jezikom C# jer ga u svakodnevnom poslu koristim te mi je jedini logični izbor za programiranje pošto nemam potrebe za učenjem novih programskih jezika. Nisam siguran da li će mi biti potrebna baza podataka, u slučaju potrebe preferiram *MySql* ili *MS Sql Server* rješenja. Trenutno nemam u planu komercijalizaciju igre, bilo prodajom ili plaćenim oglasima, ali dobro je znati da postoji i ta mogućnost.



Slika 12. Popularnost okruženja za razvoj na mobilne uređaje [21]

Nakon kratke analize došao sam do zaključka da je za mene i razvoj moje igre najpogodnije *Unity* okruženje. Osim što je besplatno, podržava i niz platformi, višekorisničko igranje te čak i mogućnost zarade prodajom završene igre ili reklamiranjem i plaćenim oglasima. Zaključio sam i da je *Unreal Engine* okruženje bolja opcija za igre koje trebaju grafički intenzivnu izvedbu i za tvrtke koje imaju vrlo iskusne razvojne timove. *Godot* je također odlična opcija ali u nedostatku vremena nisam imao vremena za kvalitetniju analizu okruženja.

4.1.1. Unity

Unity je višeplatformski *game engine* razvijen za izradu 2D i 3D video igara. Razvijen je od strane *Unity Technologies* 2004. godine s ciljem razvoja pogona prihvatljive cijene za uporabu šire javnosti. Od 2009. godine postoji potpuno besplatna verzija pogona dostupna na: http://unity3d.com/ . Za razvoj igara podržane su platforme *MS Windows 7 SP1+*, 8, 10; *macOS*. Jednom kada se igra napravi, može se izvoditi na preko 20 različitih platformi. Trenutačno podržane platforme su *Android, Android TV, Facebook Gameroom, Fire OS, Gear VR, Google Cardboard, Google Daydream, HTC Vive, iOS, Linux, macOS, Microsoft HoloLens, Nintendo 3DS family, Nintendo Switch, Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOS, WebGL, Wii U, MS Windows, MS Windows Phone, MS Windows Store* i *Xbox One*. *Unity* je do nedavno podržavao i svoj *Unity Web Player* koji je omogućavao razvoj igre u web okruženju. *Unity Web Player* je dodatak pregledniku koji je bio podržan samo u sustavima *MS Windows* i *OS X*, a koji je izašao iz upotrebe u korist *WebGL*-a koji je također podržan samo u *MS Windows* sustavima (preglednik *Internet Explorer 11*) i *OS X* (preglednik *Safari*).

Unity pruža dobru polaznu točku u razvoju igara. Besplatna verzija aplikacije omogućuje pojedincima da eksperimentiraju, uče i razvijaju igre. Ako se želi zaraditi na prodaji igara tada također postoji besplatna verzija sve dok zarada ne prelazi 100.000,00 USD godišnje. *Unity* raspolaže i velikom korisničkom bazom, a aktivna korisnička zajednica omogućuje svima - od početnika do profesionalaca da dobiju odgovore i razmjenjuju informacije. *Unity* je stoga savršen izbor za manje tvrtke koje se bave razvojem igara, kao i pojedince koji imaju želju napraviti vlastitu igru.

Unity podržava programske jezike *Boo* (podržan do verzije *Unity* 5), *UnityScript* (podržan do Verzije 2017.1) i *C#*. U ovome projektu ću se koristiti isključivo programskim jezikom *C#*, i verzijom *Unity* 2017.1.1f1. Stvar je osobnog ukusa s kojim programskim jezikom će se raditi, a unutar projekta se mogu koristiti oba programska jezika, svaki unutar svoje skripte. Također, valja istaknuti činjenicu da je *C#* najpopularniji podržani programski jezik te da nije baš

izgledno da će izaći iz upotrebe kao *Boo* i *UnityScript* jezici. Kao razvojno okruženje se može koristiti *MonoDevelop* koji dolazi zajedno sa *Unity* okruženjem ali samo do verzije 2018.1. Od te verzije pa dalje, isporučivat će se *Microsoft Visual Studio 2017 Community*, a kojega ću tijekom rada i koristiti. Postoje i alternativna rješenja:

- *Visual Studio Code,*
- *JetBrains Rider.*

4.2. Optimizacija igara

Optimizaciju možemo pratiti preko *Unity Profiler* ekrana dostupnog preko glavnog izbornika: Window --> Profiler. Preko *Profilera* imamo uvid u potrošeno vrijeme u raznim dijelovima igre. Tako možemo vidjeti koliko vremena tijekom izvođenja igre se troši na vizualni prikaz, animaciju ili logiku same igre. Oni dijelovi igre na koje se troši najviše vremena su dijelovi kojima treba posvetiti pažnju jer su to dijelovi koje je potrebno najviše optimizirati. Analizom i usporedbom rezultata prije i nakon optimiziranja lako možemo vidjeti eventualna poboljšanja ali i pogoršanja jer teško da će svaki pokušaj optimizacije biti uspješan.

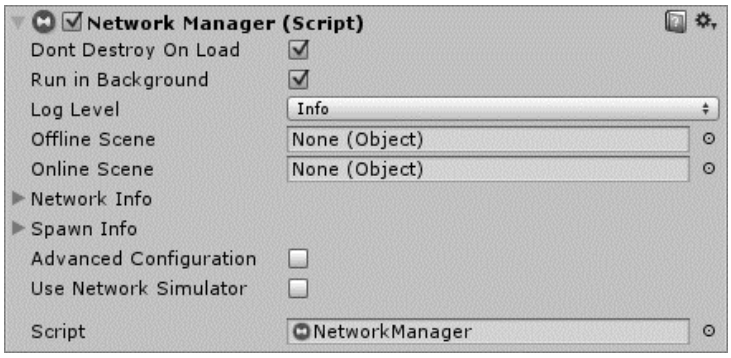
4.3. Unity programiranje

4.3.1. Komponente za višekorisničku igru

Podrška za kreiranje višekorisničke igre kroz *Unity* je podržana s nekoliko komponenti. U ovom radu ću ukratko opisati samo one komponente koje sam koristio prilikom izrade igre.

NetworkManager

Network Manager je glavna komponenta koja kontrolira višekorisničku igru te upravlja mrežnim stanjima igre. Višekorisnička igra mora imati ovu komponentu, a najčešće se implementira u glavnom izborniku. *NetworkManager* komponenta prikazana je na slici 13.



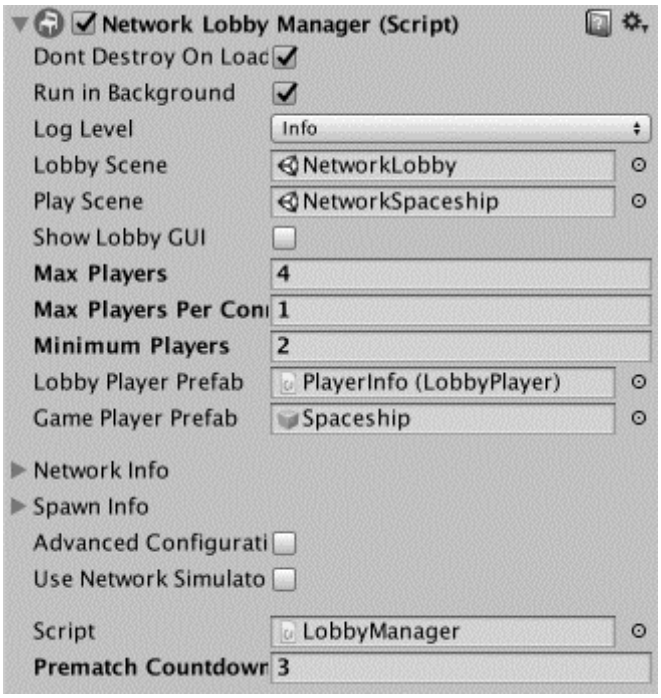
Slika 13. *NetworkManager* komponenta

Također, ova komponenta obuhvaća dosta jednostavnih elemenata i opcija kojima se može pristupiti i preko skripti te se komponenta može nadograđivati. Značajnije varijable i funkcije koje sadrži su:

- funkcije *StartHost(), StartClient()* i *StartServer()* koje se pozivaju ovisno o tome želi li korisnik započeti igru kao domaćin, klijent ili običan poslužitelj;
- varijable *networkAddress* i *networkPort*, tj. IP adresa i port koji će se koristiti za uspostavu veze;
- varijable *offlineScene* i *onlineScene* koje sadrže ime scene u kojoj se korisnik nalazi prije uspostave veze i scene u koju se prebacuje nakon poziva jedne od gornje navedenih funkcija. Prva scena obično sadrži izbornik, a druga je obično scena u kojoj započinje igra. Ako dođe do raskida veze, igra se vraća u prvu scenu.
- varijabla *playerPrefab* koja sadrži referencu na objekt koji u igri predstavlja igrača. Taj će se objekt instancirati za svakog klijenta koji se spoji.

NetworkLobbyManager

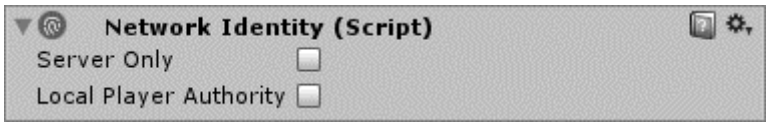
NetworkLobbyManager je posebni tip *NetworkManager* klase koji omogućuje predsoblje prije početka igre. Predsoblje je posebna scena u kojoj igrači čekaju početak igre. Čekanje traje sve dok se ne prijavi dovoljan broj igrača za igranje. Koristi se kada se želi osigurati da svi igrači u istom trenutku započnu s igrom. Komponenta je prikazana na slici 14.



Slika 14. *NetworkLobbyManager* komponenta

Network Identity

Ova komponenta kontrolira jedinstveni identitet *GameObjecta* na mreži. Dodjelom ove komponente *GameObjectu*, mrežni sustav postaje svjestan tog *GameObjecta* te mu dodjeljuje jedinstveni identifikacijski broj *NetworkInstanceId*. Svaki *GameObject* kojim upravlja igrač mora imati ovu komponentu koja je prikazana na slici 15.



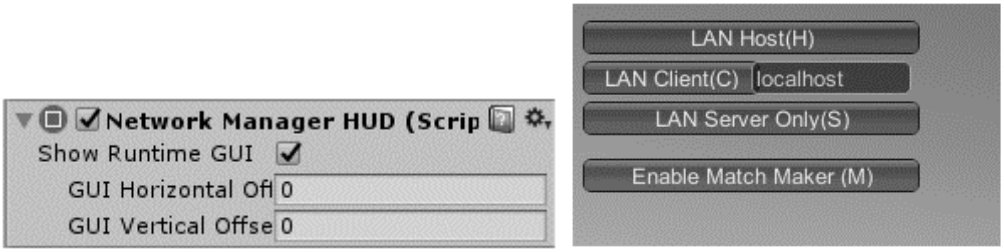
Slika 15. *NetworkIdentity* komponenta

Komponenta se sastoji od svega dva svojstva od kojih se odabire samo jedno:

- *Server Only*, *GameObject* se instancira samo na poslužitelju, ne i na klijentu
- *Local Player Authority*, kontrola nad *GameObjectu* se predaje klijentu koji je vlasnik tog *GameObjecta* te igrač dobiva autoritet nad tim *GameObjectom*

Network Manager HUD

NetworkManagerHUD komponenta omogućuje brzi i laki način dodavanja osnovnih mrežnih funkcija. Sastoji se od nekoliko jednostavnih tipki koje se prikazuju tijekom igre kao što je prikazano na slici 16. Ova komponente je namijenjena za sami početak izrade igre te bi u kasnijoj fazi razvoja ipak trebalo zamijeniti ovu komponentu sa prikladnijom verzijom.



Slika 16. *NetworkManageHUD* komponenta

Network Transform

NetworkTransform komponenta sinkronizira kretanje i rotaciju *GameObjecta* preko mreže ali samo onih *GameObjecata* koji su instancirani na poslužitelju (engl. *spawned*). Komponenta kontrolira autoritet *GameObjecta* definiranog preko komponente *NetworkIdentity* tako da svaki *GameObject* kojem želimo dodijeliti ovu komponentu mora imati i *NetworkIdentity* komponentu. Objekti s lokalnim autoritetom sinkroniziraju poziciju od klijenta

prema poslužitelju te preko poslužitelja svim ostalim lokalnim klijentima. *GameObjecti* sa poslužiteljskim autoritetom sinkroniziraju svoju poziciju od poslužitelja prema klijentima.

NetworkStartPosition

NetworkStartPosition se upotrebljava prilikom kreiranja objekata na sceni koji predstavljaju igrače. Komponenta definira poziciju i rotaciju na kojoj će se pojaviti novokreirani igrač. Obično se na sceni postave prazni *GameObjecti*, na mjestima gdje želimo da se igrači mogu pojaviti, a kojima se onda dodjeljuje ova komponenta.

4.3.2. Sinkronizacija

Dok se za kreiranje višekorisničke igre koriste mrežne komponente, za sinkronizaciju se koriste posebni atributi za koje je zadužen programer. Sinkronizacija predstavlja realan i istinit prikaz objekata na sceni za sve korisnike, tj. klijente koji sudjeluju u višekorisničkoj igri. U mojoj igri sinkronizacija znači da položaj svakoga igrača u igri mora biti isti na svim klijentima (za što je zadužena *NetworkTransform* komponenta), a ispaljivanje metaka i evidencija pogodaka svakoga igrača mora biti vidljiva svim igračima. Sinkronizacija se najlakše odrađuje preko poslužitelja, što znači da je za sve promjene u igri zadužen isključivo poslužitelj. Ako dođe po pogotka u igri, poslužitelj treba voditi računa o evidenciji štete te tu informaciju onda proslijediti svim klijentima. Ista stvar je i kod kreiranja *PowerUp* elemenata. Igrač nema nikakve veze s nastankom *PowerUp* elemenata već poslužitelj o tome vodi brigu. Sinkronizacija se odrađuje atributima, a u radu ću opisati samo one atribute koje sam koristio prilikom izrade igre.

SyncVar atribut

SyncVar je atribut koji se može dodijeliti varijablama definiranim u klasama koje nasljeđuju *NetworkBehaviour* klasu. Takve varijable će imati svoje vrijednosti sinkronizirane od poslužitelja prema svim spojenim klijentima koji sudjeluju u igri. Samo jednostavniji tipovi varijabli mogu biti označeni sa *SyncVar* atributom:

- Osnovni tipovi (*byte*, *int*, *float*, *string*, *UInt64*, itd.),
- Ugrađeni (engl. *Built-in*) *Unity* matematički tipovi (*Vector3*, *Quaternion*, itd.),
- *Struct* tip podatka koji sadržava samo dopuštene tipove.

Deklariranje *SyncVar* varijabli može se vidjeti u sljedećem kodu.

```
public class PlayerCar : NetworkBehaviour
{
      [SyncVar]
      public int health = 100;

      [SyncVar]
      public float energy = 100;
}
```

Command atribut

Atribut se dodjeljuje metodama *NetworkBehaviour* klase što omogućuje klijentu da pošalje naredbu poslužitelju za izvođenjem metode na serveru. Argumenti metode su serijalizirani preko mreže, što znači da će se metoda na poslužitelju izvoditi s istim vrijednostima kao što su i na klijentu. Metode kojima se dodjeljuje ova atribut moraju u nazivu imati prefiks „Cmd“ i ne mogu biti *static* metode. Primjer definiranja metode sa *Command* atributom može se vidjeti u sljedećem kodu.

```
public class Player : NetworkBehaviour
{
      int moveX = 0;
      int moveY = 0;
      void Update()
      {
            if (!isLocalPlayer)
                  return;
            CmdMove(moveX, moveY);
      }
      [Command]
      public void CmdMove(int x, int y)
      {
            moveX = x;
            moveY = y;
            isDirty = true;
      }
}
```

ClientRpc atribut

Atribut se dodjeljuje metodama *NetworkBehaviour* klase što omogućuje poslužitelju da pošalje naredbu klijentima za izvođenjem metode.

Argumenti metode su serijalizirani preko mreže, što znači da će se metoda na klijentu izvoditi s istim vrijednostima kao što su i na poslužitelju. Metode kojima se dodjeljuje ova atribut moraju u nazivu imati prefiks „Rpc“ i ne mogu biti *static* metode. Primjer definiranja metode sa *ClientRpc* atributom može se vidjeti u sljedećem kodu.

```
public class Example : NetworkBehaviour
{
      int counter;
      [ClientRpc]
      public void RpcDoMagic(int extra)
      {
            Debug.Log("Magic = " + (123 + extra));
      }

      void Update()
      {
            counter += 1;
            if (counter % 100 == 0 && NetworkServer.active)
            {
                  RpcDoMagic(counter);
            }
      }
}
```

4.3.3. Važniji koncepti programiranja koristeći C# skripte

NetworkBehaviour

NetworkBehaviour je osnovna (engl. *base*) klasa koja je ujedno i ekstenzija (engl. *extension*) *MonoBehaviour* klase te ima pristup svim funkcijama definiranim u *MonoBehaviour* klasi. *NetworkBehaviour* klasa je dizajnirana na način da radi samo s objektima koji sadržavaju *NetworkIdentity* komponentu te koriste HLAPI funkcije kao što su *Commands*, *ClientRPCs*, *SyncEvents* i *SyncVars*.

NetworkBehaviour se sastoji od sljedećih značajki:

- Sinkroniziranih varijabli,
- Poziva preko mreže (engl. *Network callbacks*,)
- Poslužiteljskih i klijentskih funkcija,
- Slanja naredbi *Commands*,
- Pozivanja klijentskih RPC funkcija,
- Mrežnih događaja (engl. *Networked Events*).

Svojstva *NetworkBehaviour* klase koja sam koristio tijekom izrade igre:

- *isLocalPlayer*, radi li se o lokalnom igraču?
- *isServer*, radi li se o poslužitelju?
- *isClient*, radi li se o klijentu?

Preko *NetworkBehaviour* klase korisniku je omogućena sinkronizacija varijabli od servera prema klijentima. Kako je HLAPI *server authorative*, upotreba *Commands* funkcije je način na koji klijenti zahtijevaju nešto od poslužitelja. RPC pozivi se upotrebljavaju od strane poslužitelja da odrađuju akcije na klijentima.

PlayerPrefs

Klasa *PlayerPrefs* pohranjuje i pristupa igračevim postavkama. Radi na način da zapisuje i čita vrijednosti u datoteku. Npr. da bi se pohranila neka brojčana vrijednost (tip podatka *int*):

```
PlayerPrefs.SetInt("Score", currentScore);
```

Za pročitati vrijednost upotrebljava se sljedeća funkcija:

```
currentScore = PlayerPrefs.GetInt("Score");
```

Osim *int* tipa podatka, koriste se još i *SetString()* i *SetFloat()* metode za pohranu vrijednosti te metode *GetString()* i *GetFloat()* za dohvat vrijednosti. Jedna od *korisnijih PlayerPref* metoda je i *HasKey()* koja vraća *true* ili *false*, ovisno o tome da li podatak već postoji definiran.

Singleton

Svaka igra se obično sastoji od početnog izbornika, izbora nivoa igranja, prelaska nivoa, skupljanja bodova i sličnih elemenata. Na neki način treba pamtiti neke od tih elemenata za cijelo vrijeme trajanja igre te ih prenositi između raznih *Unity* scena odnosno nivoa igre. Npr. ako na početnom izborniku postoji mogućnost odabira vozila kojim se želi voziti trka, podatak o izabranom vozilo se mora prenositi kroz sve nivoe igre, isto kao i prikupljeni bodovi (npr. broj skupljenih novčića). Te vrijednosti se mogu pamtiti preko globalnih varijabli, a globalne varijable se mogu realizirati upotrebom *singletona*. *Singleton* je dizajnerski obrazac (engl. *design pattern*) koji ograničava instanciranje klase na samo jedan objekt. *Singleton* sakriva konstruktor klase na način da se deklarira kao privatan i na taj način onemogućava instanciranje objekta izvan klase. Upotrebljava se u slučajevima kada je samo jedan objekt potreban npr. za koordinaciju svih akcija kroz cijelu aplikaciju što je i slučaj u mojoj igri. Česta upotreba

singletona je u svrhu globalnih varijabli. U tom slučaju se vrijednosti dostupne kroz cijelu igru bez obzira s koje scene se varijablama pristupa. Jedan od uobičajenih načina implementacije *singletona* prikazan je u sljedećem programskom kodu:

```
public sealed class Singleton
{
      private static Singleton instance = new Singleton();
      private Singleton() {}

      public static Singleton Instance
      {
            get
                  return instance;
      }
}
```

Object pooling

Instantiate() i *Destroy()* su metode koje se koriste za kreiranje i uništavanje objekata tijekom igre. Svako kreiranje i uništavanje objekata zahtijeva neko minimalno procesorsko vrijeme. Kada se ne kreira puno objekata, potrošnja vremena i procesora je zanemariva. Međutim, u slučajevima kada se u kratko vrijeme kreira i uništava jako puno objekata može doći do značajnijeg zauzimanja procesora i većih kašnjenja u igri. Također, *Unity* upotrebljava *Garbage Collection* da oslobodi prostor u memoriji koji se više ne upotrebljava. Često pozivanje *Destroy()* metode okida *Garbage Collection* te na taj način dodatno opterećuje procesor i usporava igru. Primjer je ispaljivanje metaka u igri. Jedno vozilo može ispaliti jako puno metaka u kratkom vremenu, a igru istovremeno može igrati više igrača s više vozila koja istovremeno mogu ispaljivati metke. Jedan od načina kako možemo igru optimizirati je korištenje *object poolinga*. To je postupak u kojemu se unaprijed rezervira memorija za metke na način da ih se u samom početku igre instancira kao neaktivne objekte koristeći metodu, te se tijekom igre koriste isključivo tako instancirani metci. Umjesto stvaranja novih objekata i uništavanja starih tijekom igranja, igra ponovno koristi objekte iz "bazena". Dohvaća se prvi neaktivni metak, aktivira se, odradi se ispaljivanje, a umjesto uništavanja se opet deaktivira. Primjer realizacije *object poolinga* prikazan je u sljedećem kodu. U *Start* metodi se instancira određeni broj objekata koji se zatim zapišu u listu objekata *pooledObjects*. Lista objekata je „bazen“ iz koje se dohvaćaju neaktivni objekti.

```
public List<GameObject> pooledObjects;
public GameObject objectToPool;
public int amountToPool;
void Start()
{
      pooledObjects = new List<GameObject>();
      for (int i = 0; i < amountToPool; i++)
      {
            GameObject obj = (GameObject)Instantiate(objectToPool);
            obj.SetActive(false);
            pooledObjects.Add(obj);
      }
}
```

Korutine

Korutine (engl. *Coroutines*) su funkcije koje imaju mogućnost pauziranja svojega izvođenja. Tijekom tog pauziranja program se nastavlja dalje izvoditi sve dok ne prođe vrijeme pauze nakon čega se nastavlja izvođenje korutine. Korutina je deklarirana povratnim tipom *IEnumerator* te s *yield return* naredbom koja mora biti uključena negdje unutar funkcije. Naredba *yield return* je linija u kojoj se izvršavanje pauzira, a pauziranje traje sve dok se uvjeti ne zadovolje. Uvjeti ovise o načinu na koji je *yield* definiran:

- *yield return null* – izvođenje se nastavlja u okviru nakon što se izvedu sve *Update* funkcije;
- *yield return new WaitForSeconds(t)* – izvođenje se nastavlja nakon isteka *t* sekundi;
- *yield return new WaitForEndOfFrame()* – izvođenje se nastavlja nakon što su sve kamere i grafička korisnička sučelja renderirana;
- *yield return new WaitForFixedUpdate()* – izvođenje se nastavlja nakon poziva svih *FixedUpdates* funkcija;
- *yield return new WWW(url)* – izvođenje se nastavlja nakon uspješnog ili neuspješnog preuzimanja web sadržaja.

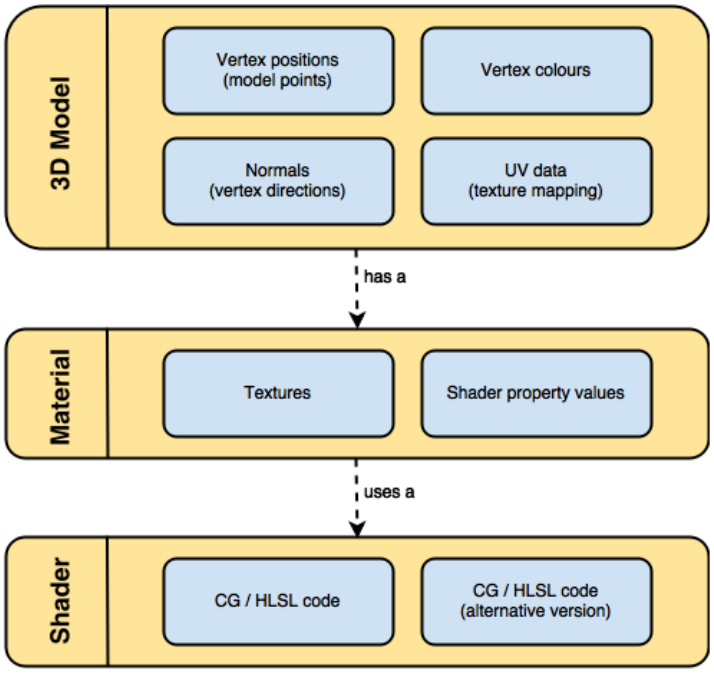
Da bi se korutina pokrenula treba ju pozvati preko *StartCoroutine* funkcije. Može se pozvati na dva načina: preko tipa podatka *string*: *StartCoroutine("myCoroutine")* ili preko tipa podatka *IEnumerator*: *StartCoroutine(myCoroutine()).*

Za *string* tip pozivanja je moguća upotreba samo korutina koje nemaju parametara te se preporučuje upotreba preko tipa pozivanja *IEnumerator*.

```
void Start()
{
      StartCoroutine("MyCoroutine"); // string based
      StartCoroutine(MyCoroutine()); // IEnumerator based
      StartCoroutine(MySecondCoroutine(0.5f); //IEnumerator based sa
parametrom
}

IEnumerator MyCoroutine()
{
      Debug.Log("Hello world!");
      yield return null;
}

IEnumerator MySecondCoroutine(float waittime)
{
      yield return new WaitForSeconds(waittime);
      Debug.Log("Hello world");
}
```

4.4. Materijali, alati za sjenčanje i teksture

Za realističan prikaz objekata u sceni, *Unity* upotrebljava materijale (engl. *materials*), alate za sjenčanje (engl. *shaders*) i teksture (engl. *textures*) koji su međusobno ovisni, što je prikazano na slici 17.



Slika 17. Veza između materijala, alata za sjenčanje i tekstura [13]

Materijali definiraju kako površina modela treba izgledati u ovisnosti o upotrjebljenim teksturama, pozicioniranju tekstura i nijansama boje. Dostupne opcije za materijale ovise o upotrjebljenom alatu za sjenčanje.

Alati za sjenčanje su skripte koje sadrže matematičke izračune i algoritme za izračunavanje boje svakog piksela temeljene na osvjetljenju i postavkama materijala.

Teksture su *bitmap* slike. Materijal može sadržavati reference na teksture, tako da alat za sjenčanje koji se upotrebljava može koristiti teksture prilikom izračunavanja površinske boje *GameObjecta*. Uz temeljnu boju (engl. *Albedo*) površine *GameObjecta*, teksture mogu predstavljati mnoge druge aspekte površine materijala kao što su njegova refleksija ili hrapavost.

Unity sadrži više različitih alata za sjenčanje. Za većinu renderiranja kao što su prikazivanje likova, okoliša, tvrde i mekane površine, najbolji izbor je korištenje *Standard* alata za sjenčanje.

Novi materijal se kreira preko opcije: Assets->Create->Material iz glavnog izbornika, a *Standard* je ujedno i *default* alat za sjenčanje prilikom izrade novih materijala.

Standard je vrlo prilagodljiv alat za sjenčanje preko kojega se mnoge vrste površina mogu prikazati na realističan način. Postoje i alati posebno pogodni za mobilne uređaje. Međutim, kako sam tijekom izrade igre svakako ograničio upotrebu svjetla jer svako osvjetljenje zahtjeva dodatno procesorsko opterećenje, kroz izradu igre koristio sam *Standard* alat za sjenčanje.

4.5. Svjetla

Svjetla su važan element svake scene. Svaka scena može imati više izvora svjetlosti, a svi izvori svjetlosti značajno utječu na konačan izgled scene. Potrebno je malo prakse i iskustva u radu sa svjetlima, a najbolji način za učenje je eksperimentiranje. Važno je i ne pretjerivati jer svjetla mogu značajno utjecati na performanse same igre.

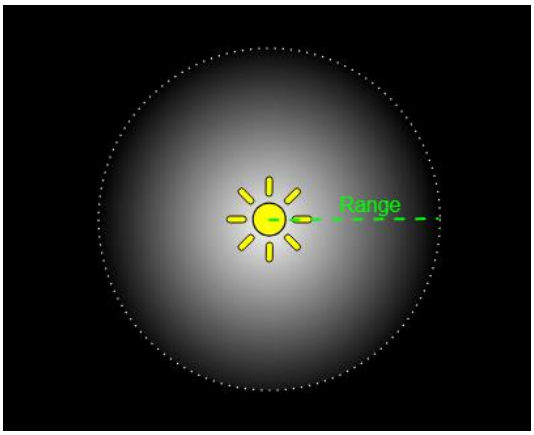
Svjetla se na scenu dodaju preko izbornika GameObject->Light, a na podizborniku se prikazuju dostupne vrste svjetla. Novostvorenim svjetlom možemo upravljati baš kao i bilo kojim drugim *GameObjectom*.

4.5.1. Vrste svjetla

Na scenu možemo dodavati različite vrste izvora svjetlosti, ovisno o potrebama.

Točkasta svjetla

Točkasto svjetlo (engl. *point light*) ima izvor svjetlosti u jednoj točki u prostoru i iz te točke šalje svjetlo na sve strane jednako. Na slici 18 je prikazan izvor točkastog svjetla. Intenzitet se smanjuje s obzirom na udaljenost od izvora, a pada na nulu na udaljenosti definiranoj u postavkama.

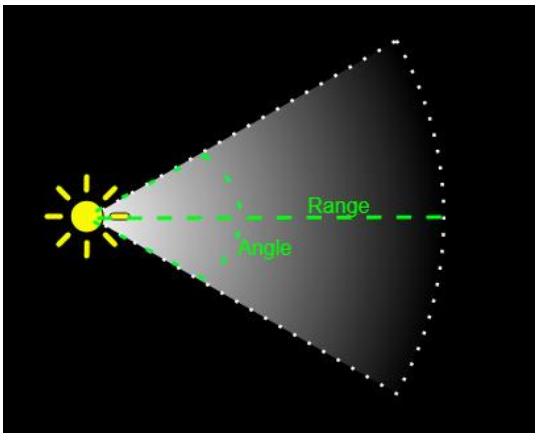


Slika 18. Točkasto svjetlo [13]

Ova vrsta izvora svjetlosti korisna je za simuliranje svjetlosti lampe, baklje ili sličnih vrsta svjetlosti u sceni. Također se mogu koristiti i kod kreiranja iskrenja ili eksplozija. Mogu se koristiti tako da na tren osvijetle scenu prilikom eksplozija ili udara groma.

Reflektori

Reflektori (engl. *spot lights*) također imaju definiranu lokaciju u prostoru i domet svjetlosti. Za razliku od točkastog svjetla, reflektori su ograničeni na jedan kut što rezultira stožastim područjem osvjetljenja, što je prikazano na slici 19.

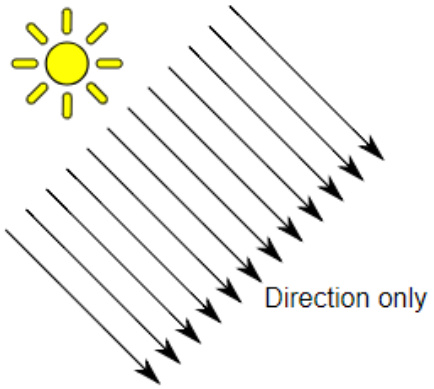


Slika 19. Reflektor [13]

Reflektori se generalno koriste kao umjetni izvori svjetlosti npr. za ručne svjetiljke, svjetla na autu ili žmigavce.

Usmjerena svjetla

Usmjerena svjetla (engl. *directional lights*) se najčešće koriste za stvaranje efekta sunčeve svjetlosti. Na sceni se ponašaju kao udaljeni izvori svjetlosti koji istim intenzitetom obasjavaju sve objekte na sceni bez obzira na udaljenost objekata od izvora svjetlosti. Također, svi objekti na sceni se obasjavaju uvijek iz istog smjera. Usmjereno svjetlo je prikazano na slici 20.



Slika 20. Usmjereno svjetlo [13]

Osim kao izvor sunčeve svjetlosti, usmjereno svjetlo se može koristiti i za simulaciju mjesečeve svjetlosti ili bilo kojeg drugog velikog izvora svjetlosti koji se nalazi daleko od objekata u sceni.

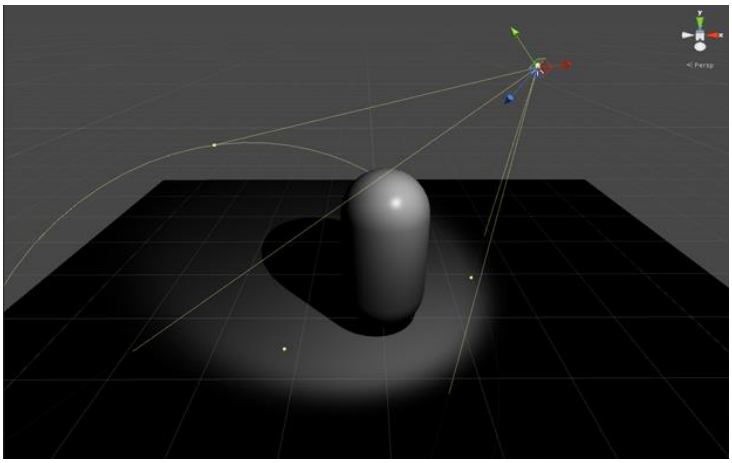
4.5.2. Osvjetljenje

Simulacija realnog osvjetljenja scene je zahtijevan proces koji zahtijeva mnoštvo točnih izračuna. Zbog toga igre upotrebljavaju niz pristupa za obradu tih izračuna prije samog igranja, odnosno već prilikom izrade igre. Na taj način se za vrijeme igranja troši manje resursa što je pogotovo važno za igre koje se igraju na mobilnim uređajima.

Općenito govoreći, osvjetljenje u *Unityju* se na neki način može realizirati kao osvjetljenje u stvarnom vremenu (engl. *realtime*) ili unaprijed izračunato (engl. *precomputed*) osvjetljenje. Obje tehnike mogu se kombinirati kako bi se stvorilo željeno osvjetljenje scene i objekata u sceni.

Osvjetljenje u stvarnom vremenu

Inicijalno, svjetla u *Unityju* (točkasta, reflektori, usmjerena) se realiziraju u stvarnom vremenu. To znači da se osvjetljenje scene osvježava i izračunava prilikom svakog okvira. Kako se svjetla i objekti kreću unutar scene tako se i parametri osvjetljenja ponovno preračunavaju, a scena osvježava. Ovo je najosnovniji način osvjetljenja scene te je pogodan za osvjetljavanje likova u igri ili bilo kojeg pokretnog objekta u sceni.

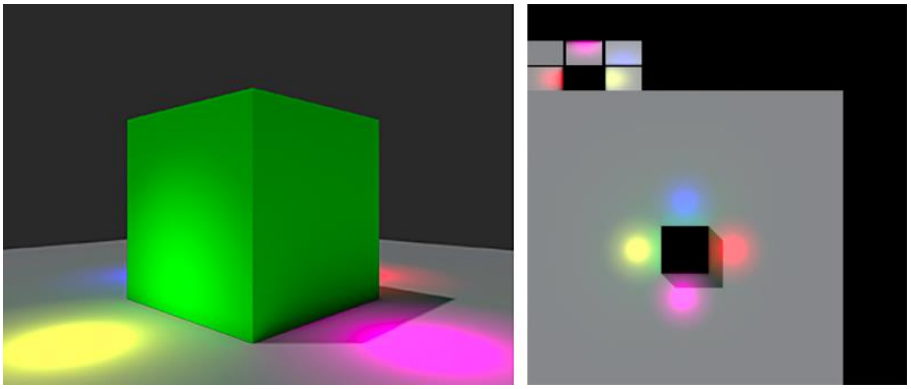


Slika 21. Osvjetljenje u stvarnom vremenu [13]

Na slici 21 se vidi da je sjena osvijetljenog objekta potpuno crna zato što ne postoji refleksija svjetlosti. Da bi se stvorila realističnija scena, moguće je koristiti i druge tehnike osvjetljavanja primjerice globalno osvjetljenje (engl. *global illumination*, GI).

Baked GI Osvjetljenje

Baking predstavlja proces u kojemu se efekti osvjetljenja statičkih objekata na sceni izračunavaju, a rezultat zapisuje u teksture koje se oblažu preko scene. Ovakva vrsta osvjetljenja je pogodna za sve objekte na sceni koji ne mijenjaju položaj tijekom igre te se na taj način vrši značajna ušteda potrošnje resursa.



Slika 22. *GI* osvjetljenje sa teksturom [13]

Na slici 22 lijevo je prikazana scena osvijetljena svjetlima, a na slici desno je prikazana tekstura koja nastaje u *baking* procesu. Tekstura sadrži sve svjetlosne efekte uključujući i sjenu.

4.6. Modeliranje scene i objekata

Za modeliranje 3D objekata potreban je neki od alata za 3D modeliranje. Postoji jako puno alata za 3D modeliranje, neki su besplatni, a neki se moraju platiti. Neki alati poput *AutoCAD* su profesionalni alati namijenjeni uglavnom arhitekturi i građevinarstvu te nisu pogodni za izradu objekata koje ću koristiti u igri. Za razliku od nekih drugih industrija, gdje je za modele nužno da budu što detaljniji i precizniji, jedna od glavnih značajki kod 3D modeliranja za video igre je kako zadržati dovoljno detalja u modelu, a u isto vrijeme smanjiti broj poligona od kojih je model napravljen kako igra ne bi bila preopterećena i spora. Istražujući alate za modeliranje, došao sam do kratkog popisa pogodnih alata za izradu 3D objekata koje ću koristi u igri:

- *Maya*
- *Blender*
- *Cinema 4D*
- *3D Studio Max.*

Maya [22] je moćan alat koji se jako puno koristi u modernoj filmskoj industriji, ali i u arhitekturi i industriji video igara. Razvijen je od strane *Autodesk Inc.* te se radi o komercijalnom alatu što znači da nije besplatan za upotrebu. Dostupan je za *MS Windows*,

macOS te *Linux* operacijske sustave. Veliki filmski hitovi poput Avatara, Gospodara prstenova i Spider-Mana nastali su uz pomoć ovog alata.

Blender [23] je besplatni alat otvorenog koda za 3D modeliranje, animaciju, vizualne učinke, video uređivanje, a moguće je čak i video igre kreirati. Dostupan je kao instalacijski programski paket za *MS Windows* (verzija *Vista* i više), *macOS* (*OSX* 10.6 i više) i *Gnu/Linux* operacijske sustave. Za korisnike koji još uvijek koriste *MS Windows XP* dostupna je starija verzija alata. Razvojnim programerima i svima koji žele doprinijeti razvoju alata dostupan je izvorni kod. *Blender* je razvila nizozemska tvrtka *NeoGeo* u vlasništvu Ton Roosendaala, no u konačnici *Blender* grafička aplikacija rezultat je rada *Blender* fondacije, neovisne i neprofitne javne organizacije koja želi razvijati i promicati 3D tehnologiju na međunarodnom nivou s *Blenderom* kao osnovnim alatom.

3D Studio Max [24] je najpopularniji program za 3D modeliranje. Uz filmsku industriju i industriju video igara, često se koristi i u arhitekturi te dizajnu interijera. Radi se o komercijalnom i jako skupom alatu razvijenom od strane *Autodesk Inc*., a dostupan je samo u *MS Windows* operacijskom sustavu.

Cinema 4D [25] je komercijalni alat primarno osmišljen kao alat za filmsku produkciju, no najčešće se koristi u TV produkciji. Razvijen je od strane *MAXON Computer GmbH*, a dostupan je za *MS Windows, macOS* te *AmigaOS* operacijske sustave.

Većina alata ima dosta gotovo identičnih mogućnosti. Svi nabrojani alati se mogu koristiti za modeliranje, animiranje i teksturiranje, a većina razlika se odnosi uglavnom na korisničko sučelje i jednostavnost korištenja. Svi alati također imaju i podršku za FBX format. FBX format se koristi za pohranu 3D modela te je pogodan za izvoz 3D modela u druge alate uključujući i *Unity*. Iako do sada nisam imao iskustva s 3D modeliranjem, u ovome radu odlučio sam se za *Blender*. Kao početnika me najviše privuklo što iako se radi o besplatnom alatu, *Blender* je profesionalan alat s odličnom dokumentacijom.

4.6.1. Instalacija Blendera

Zbog prirode alata treba obratiti pažnju na zahtjeve računalne opreme (engl. *hardware requirements*). Blender.org preporuča 3 kategorije zahtjeva: minimalna, preporučena i optimalna [23].

- Minimalna (osnovna upotreba) računalna oprema:

  32-bit dvojezgreni procesor 2Ghz sa SSE2 podrškom

2 GB RAM

OpenGL 2.1 kompatibilna grafička kartica s minimalno 512 MB RAM

- Preporučena računalna oprema:

  64-bit četverojezgreni procesor

  8 GB RAM

  OpenGL 3.2 kompatibilna grafička kartica s minimalno 2 GB RAM

- Optimalna računalna oprema

  64-bit osmojezgreni procesor

  16 GB RAM

  Dual OpenGL 3.2 kompatibilna grafička kartica s minimalno 4 GB RAM

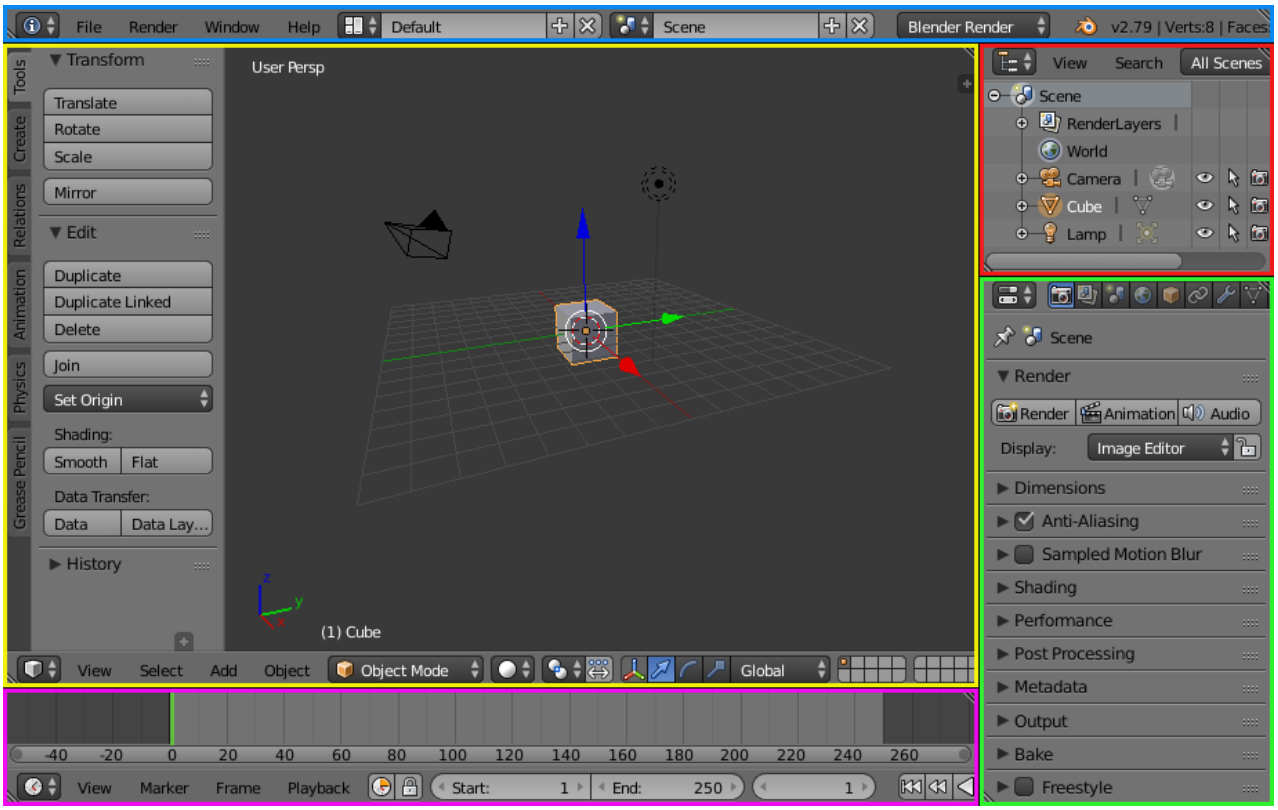
Instalacija *Blendera* se obavlja sa www.blender.org/download web stranice koja je prikazana na slici 23. Kod downloada treba odabrati instalacijski paket pogodan za operacijski sustav koji koristimo.

Slika 23. Odabir instalacije [23]

Trenutna verzija *Blendera* u kojoj ću izraditi sve modele je 2.79b, a već krajem 2018. godine se očekuje i nova verzija 2.80.

4.6.2. Korisničko sučelje

Sučelje je podijeljeno na nekoliko dijelova (upravljačkih prozora) i može biti različitog izgleda ovisno o tome što se trenutno radi u programu. Veličina i položaj svakog dijela može se prilagođavati lijevim klikom miša na graničnu crtu i njenim povlačenjem. Izgled sučelja vidljiv je na slici 24.



Slika 24. *Blender* sučelje

Osnovni elementi početnog ekrana su:

- *Info*, na vrhu ekrana, na slici obrubljen plavom bojom,
- *3D View*, središnji dio ekrana, na slici obrubljen žutom bojom,
- *Timeline*, na dnu ekrana, na slici obrubljen ljubičastom bojom,
- *Outliner*, u gornjem desnom dijelu ekrana, na slici obrubljen crvenom bojom,
- *Properties*, u donjem desnom dijelu ekrana, na slici obrubljen zelenom bojom.

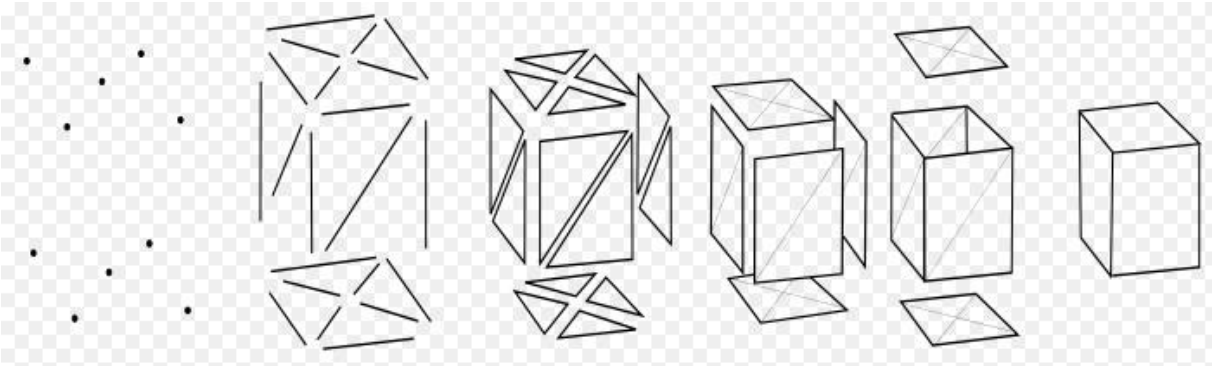
Prema potrebi, sučelje se može prilagoditi pomoću postavki zaslona (engl. *Screen Layouts*). Željene postavke se zatim mogu spremiti za kasniju upotrebu. Prilagodba postavki zaslona se nalazi na izborniku: Window ▸ Screen ▸ Areas ▸ Editors ▸ Regions ▸ (Tabs) ▸ Panels ▸ Controls

Prečaci uvelike olakšavaju rad u Blenderu, a prečaci koji su meni bili od najveće koristi nalazi se na popisu prečaca CommonBlenderShortcuts .

4.6.3. 3D modeliranje

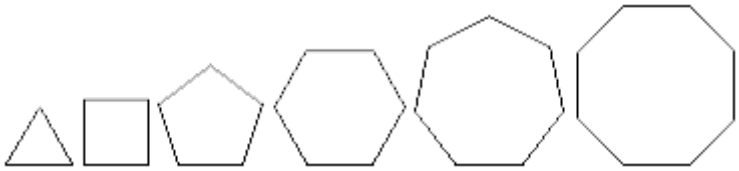
Objekti u stvarnom svijetu definirani su s tri dimenzije: visinom, širinom i dužinom. Dimenzija je karakteristika prostora koja identificira objekte i njihov položaj u prostoru. Cilj 3D modeliranja je stvoriti 3D model koji predstavlja trodimenzionalnu simulaciju stvarnog

objekta. Prepoznatljiv je po svojem obliku koji je definiran vrhovima (engl. *vertex*). Vrhovi se povezuju bridovima (engl. *edge*), a tri ili više povezana ruba čine lice stranice (engl. *face*). Primjer simulacije stvarnog objekta prikazan je na slici 25.



Slika 25. Simulacija stvarnog objekta [26]

Najjednostavniji poligon je trokut i upravo on je u najčešćoj upotrebi u 3D grafici gdje su sklopovlja stvorena tako da mogu prikazati milijune trokuta u samo jednoj sekundi. Svaki od poligona koji nije trokut je rastavljiv na dva ili više trokuta. To uvelike doprinosi 3D modeliranju jer svaki složeniji 3D model se može rastaviti na veliki broj trokuta. Prikaz takvog rastavljanja vidljiv je na slici 26.



Slika 26. Poligoni [26]

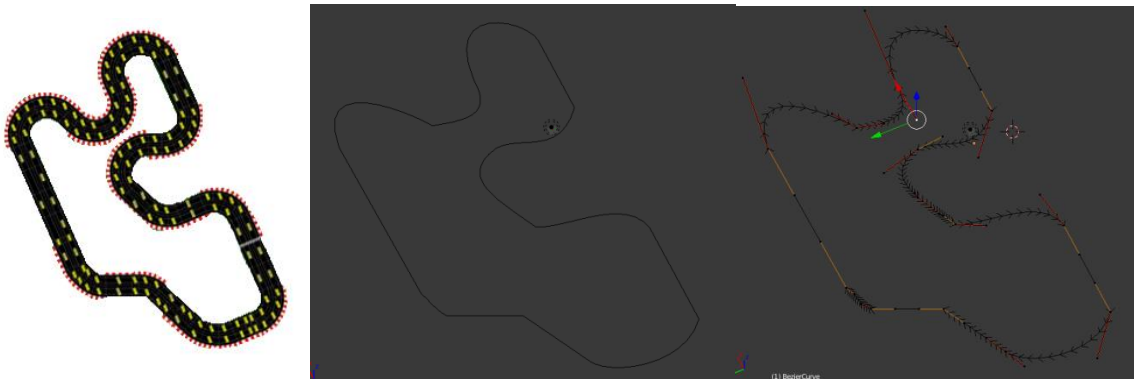
Modeliranje staze

Postoji više tehnika za modeliranje 3D objekta. Jedna od tehnika je modeliranje pomoću kocke (engl. *box modelling*) gdje se model počinje modelirati od obične kocke te se pomoću alata za poligonalno modeliranje izradi gotovi model. Druga tehnika je modeliranje pomoću krivulja (engl. *spline modeling*), što podrazumijeva korištenje krivulja u prostoru pomoću kojih se izrađuju površine, a okolina je izrađena pomoću proceduralnog modeliranja.

Izrada 3D modela staze

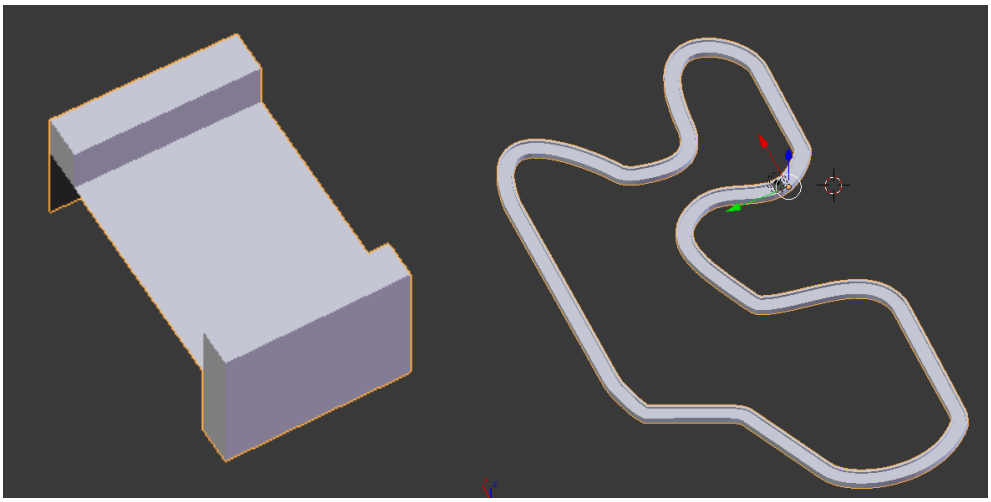
Model 3D staze sam izradio koristeći krivulje. Na slici 27 je prikazan oblik staze koju želim modelirati te krivulja koja prati oblik staze. Krivulja se kreira preko glavnog izbornika Add→ Curve → Bezier. Nakon kreiranja krivulje, potrebno je odabrati *Edit mode* tako da

možemo raditi promjene na krivulji. Krivulja se sastoji od dvije točke, početne i završne. Da bi kreirali oblik staze potrebno je dodati još točaka, a to se radi tako da se odabere bilo koja točka na krivulji te se pritisne prečac E – *Extrude*. Nova točka se zatim pozicionira na željeno mjesto, a postupak se ponavlja sve dok krivuljom ne nacrtamo kompletnu stazu.



Slika 27. Staza i krivulja

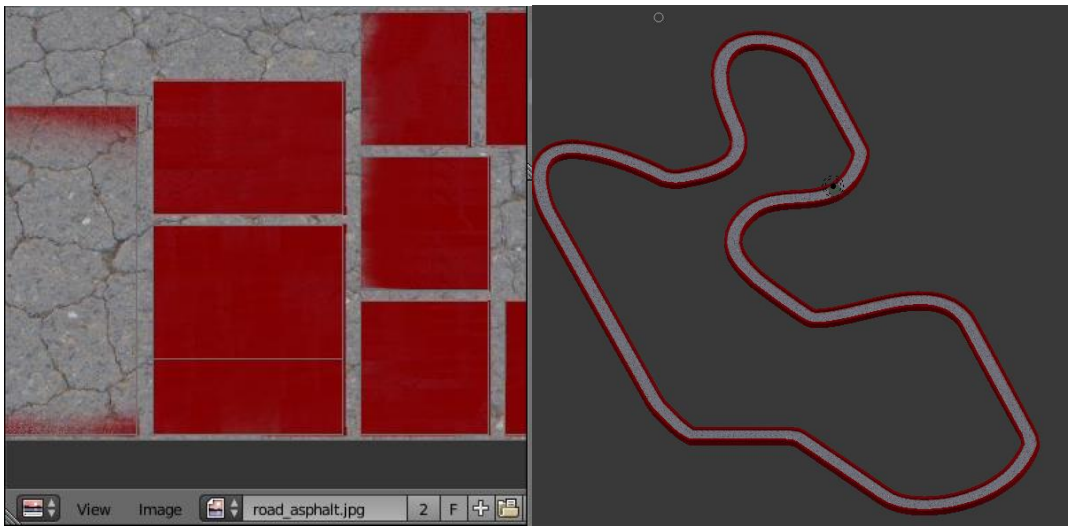
Nakon definiranja oblika staze, treba modelirati osnovni 3D element koji će se povezati na krivulju te multiplicirati tako da pokrije cijelu krivulju. To se radi na način da se osnovnom elementu dodijeli *Array modifier* te se odabere opcija *Fit Curve* tako da se veličina polja poklapa sa veličinom krivulje. Da se uskladi i oblik sa krivuljom potrebno je još dodati i *Curve modifier*. Rezultat je prikazan na slici 28.



Slika 28. 3D model staze

UV mapiranje

Nakon modeliranja slijedi teksturiranje. Teksturiranje je proces kojim se modelima dodjeljuju detalji, boje i razni efekti poput hrapavosti, sjaja, refleksije i slično. Da bi se tekstura primijenila na 3D model koristi se UV mapiranje (engl. *mapping*). UV mapiranje je proces u kojemu se od 3D modela kreira UV mapa. UV mapa je prikaz razmotanog (engl. *unwrap*) 3D modela u 2D koordinatnom prostoru gdje se osi označavaju slovima U i V, a na kojoj se crtaju željene teksture. Za trkaću stazu je prirodno da je napravljena od asfalta tako da ću kao materijal upotrijebiti teksturu asfalta, a ogradu staze ću jednostavno obojiti crvenom bojom. Rezultat UV mapiranja je prikazan na slici 29. Dodavanje površinskih detalja objektima pridonosi realnosti tih objekata i cijele scene.



Slika 29. UV mapiranje staze

4.6.4. Kako dalje

U ovome potpoglavlju sam prikazao samo osnove Blendera. Izradio sam samo model staze, a kroz *Blender* se može daleko više toga napraviti. *Blender* ima široki spektar mogućnosti zbog kojih ga se može usporediti s mnogim komercijalnim 3D softverskim rješenjima. Na njegovu razvoju radi zajednica stručnjaka i korisnika koji pišu i izdaju priručnike i vodiče za rad u *Blenderu*, poznatija kao *BlenderArtists*. Na njihovim se stranicama http://blenderartists.org/forum/ mogu pronaći korisni savjeti i upute, a cijela zajednica djeluje kao podrška svim korisnicima *Blendera*.

Na Internetu postoji mnogo kvalitetnih *Blender* sadržaja, a neki koji su meni bili najzanimljiviji su:

- Blender manual: Editors — Blender Manual
- Blender Udemy tečaj: https://www.udemy.com/blendertutorial/learn/v4/content

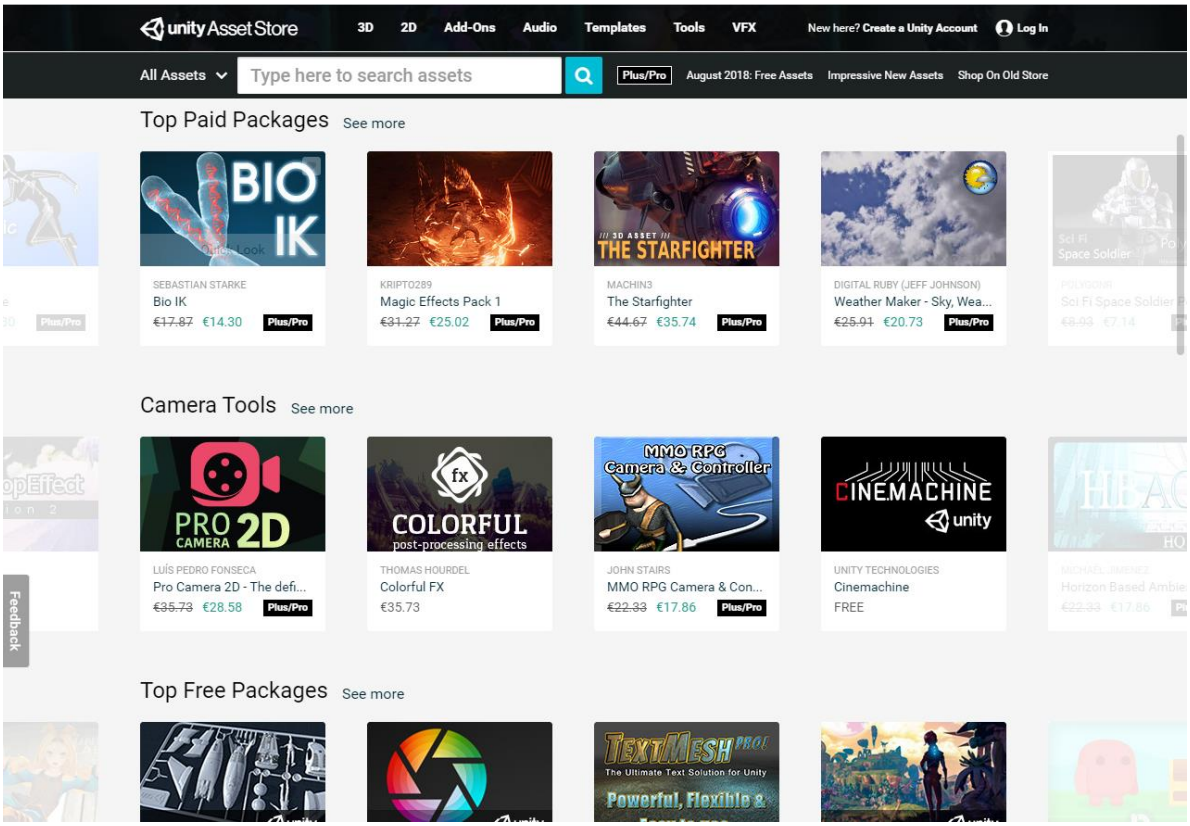
YouTube Blender kanali:

- Blender: https://www.youtube.com/channel/UCSMOQeBJ2RAnuFungnQOxLg
- Blender Guru: https://www.youtube.com/channel/UCOKHwx1VCdgnxwbjyb9Iu1g
- Darrin Lile: https://www.youtube.com/channel/UCyu4kn_ROhA34jug2g_AwJA
- Grant Abbitt: https://www.youtube.com/channel/UCZFUrFoqvqlN8seaAeEwjlw
- CG Masters: https://www.youtube.com/user/blengine

5. Izrada igre

5.1. Trgovina elementima

Prije početka izrade same igre potrebno je imati instalirane neke standardne predloške i efekte. Trgovina elementima (engl. *Asset Store*) omogućuje prodaju i kupovinu 3D modela, dodataka, predložaka, skripti i mnogih drugih sadržaja. Nalazi se na adresi: https://www.assetstore.unity3d.com/en/, a početna stranica izgleda kao na slici 30. Sadržaju trgovine se može pristupiti i kroz *Unity* uređivač, što je puno bolja opcija ako se želi nešto dohvatiti iz trgovine. Potrebno je samo otvoriti prozor *Asset Store* tako da se na glavnom izborniku odabere Window--> Asset Store.



Slika 30. Trgovina elementima [27]

Preko trgovine elemenata treba preuzeti besplatne elemente:

- *Standard Assets*, dostupne na linku https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351 . treba preuzeti ako standardni elementi već nisu instalirani prilikom instalacije *Unity* okruženja. Sadrži elemente, predloške te skripte koje uključuju *Sky Car*, okoliš, kontrole za mobilno upravljanje.
- *Unity Particle Pack*, dostupan na linku

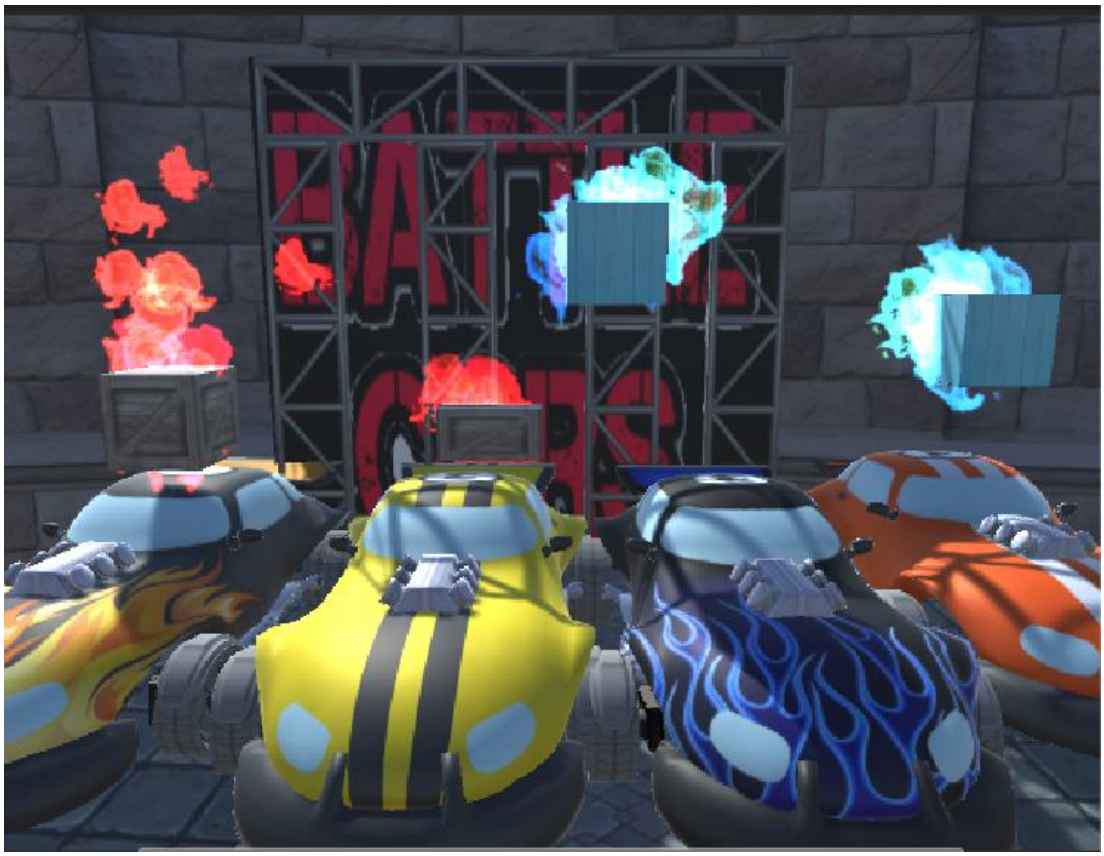
https://assetstore.unity.com/packages/essentials/asset-packs/unity-particle-pack-73777 sadrži razne efekte koje možemo upotrijebiti u igri.

5.2. Battle Cars

Battle Cars je jednostavna višekorisnička igra koja podržava istovremeno igranje do 4 igrača. Igra je dizajnirana isključivo za igranje preko lokalne LAN mreže koja može biti realizirana preko fizički povezanih uređaja (pogodno za stolne računala) ili preko WiFi signala (pogodno za mobilne uređaje). U igri igrač upravlja posebno dizajniranim vozilom koje ima mogućnost ispaljivanja metaka i evidencije zdravlja. Kao protivnik će poslužiti drugi igrači koji također imaju mogućnost ispaljivanja metaka. Igra omogućuje izbor 4 različita vozila, a svako vozilo ima drugačije karakteristike. Karakteristike vozila su:

- Brzina vozila (engl. *Speed*),
- Broj metaka (engl. *Ammo*),
- Brzina ispaljivanja metaka (engl. *Fire rate*).

Dio igre je realiziran koristeći standardne predloške i postojeće 3D modele, a neke predloške sam napravio kroz *Unity* te 3D modele koristeći *Blender*. Za pucanje i pretrpljenu štetu sam napisao skripte u programskom jeziku C#.



Slika 31. Battle Cars

5.3. Izrada elemenata igre

5.3.1. Predložak za oružje

Da bi igračima omogućili pucanje potrebno je prvo napraviti predložak za oružje. Taj predložak se zatim dodaje na predložak auta i na taj način se autima omogućuje pucanje. Osim predloška za oružje, potrebno je kreirati i predložak za metak. Predložak za metak je zapravo jednostavan model cilindra koji mora imati i komponentu *Rigidbody* te skriptu *Bullet.cs* koja je zadužena za detekciju pogotka. Važnije metode skripte su *OnCollisionEnter()* koja provjerava da li je metak pogodio igrača ili ne. U slučaju pogotka igrača, pogođenom igraču se nanosi šteta pozivom metode *Damage()*. Pošto je metak predložak koji nije statičan u igri te mora biti sinkroniziran za sve igrače, potrebno je predlošku dodijeliti *NetworkIdentity* i *NetworkTransform* komponentu.

```
private void OnCollisionEnter(Collision collision)
{
     CheckCollision(collision);
     Explode();
}

private void CheckCollision(Collision collision)
{
     if (collision.gameObject.tag == "Player")
     {
          CarPlayer player =
               collision.collider.GetComponentInParent(
                    typeof(CarPlayer)) as CarPlayer;
          if (player != null)
               player.Damage(m_damage, m_owner);
     }
}
```

Predložak za oružje je također jednostavni model cilindra. Predlošku je dodijeljena skripta *WeaponShoot.cs* koja je zadužena za pucanje, tj. za ispaljivanje metka. Da bi pucanje predloška bilo moguće, predlošku se preko *WeaponShoot.cs* skripte dodjeljuje predložak metka, položaj ispaljivanja metka te početna količina metaka i brzina pucanja. Pošto se predložak za oružje samo dodjeljuje predlošku auta te zapravo samo prikazuje oružje, nije potrebno dodjeljivati mrežne komponente. Ali isto tako, pošto je oružje predložak koji će imati svako vozilo koje sudjeluje u igri, potrebno je sinkronizirati pucanje. Prva kontrola se radi preko *isLocalPlayer* člana *NetworkBehaviour* klase.

isLocalPlayer – vrijednost je *true* ako se radi o objektu koji je kontroliran lokalnim uređajem. Na ovaj način se izbjegava da lokalni igrač puca ne samo sa svoga vozila već i sa svih ostalih vozila koji sudjeluju u igri.

```
public void Shoot()
{
     if (!isLocalPlayer)
          return;
     if (m_shootsLeft <= 0)
          return;
     if (m_reloading == true)
          return;
     CmdShoot();
     m_shootsLeft--;
     UpdateAmmoBar(m_shootsLeft);
}
```

```
[Command]
void CmdShoot()
{
     Bullet bullet = null;
     Rigidbody rbody = Instantiate(m_bulletPrefab,
          m_bulletPosition.position,
          m_bulletPosition.rotation) as Rigidbody;
     bullet = rbody.gameObject.GetComponent<Bullet>();
     if (rbody != null)
          rbody.velocity = bullet.m_speed *
               m_bulletPosition.transform.forward;
     bullet.m_owner = GetComponent<CarPlayer>();
     NetworkServer.Spawn(rbody.gameObject);
}
```

Pucanje je jedna od akcija za koju je puno bolje da se izvodi na poslužitelju, tj. da poslužitelj preuzme kontrolu nad ispaljenim mecima. Pucanjem se instancira predložak metka koji treba biti vidljiv svim igračima. To se postiže atributom [*Command*] te naredbom *NetworkServer.Spawn().*

NetworkServer.Spawn() metoda kreira novi objekt na svim klijentima koji sudjeluju u igri.

[Command] atribut omogućuje klijentu da pošalje naredbu za kreiranjem metka poslužitelju, a poslužitelj zatim kreira metak vidljiv svim igračima.

Nakon svakog ispaljenog metka odrađuje se i ažuriranje vizualnog prikaza preostalih metaka preko klizača (engl. *Sliders*) koji se nalazi na vrhu ekrana u scenama borbene arene i utrke. Ista

stvar vrijedi i za prikaz preostalog zdravlja koje se izračunava u *Damage()* metodi. Vizualni prikazi zdravlja i preostalih metaka su vidljivi samo lokalnom igraču tako da nije potrebna sinkronizacija između klijenata.

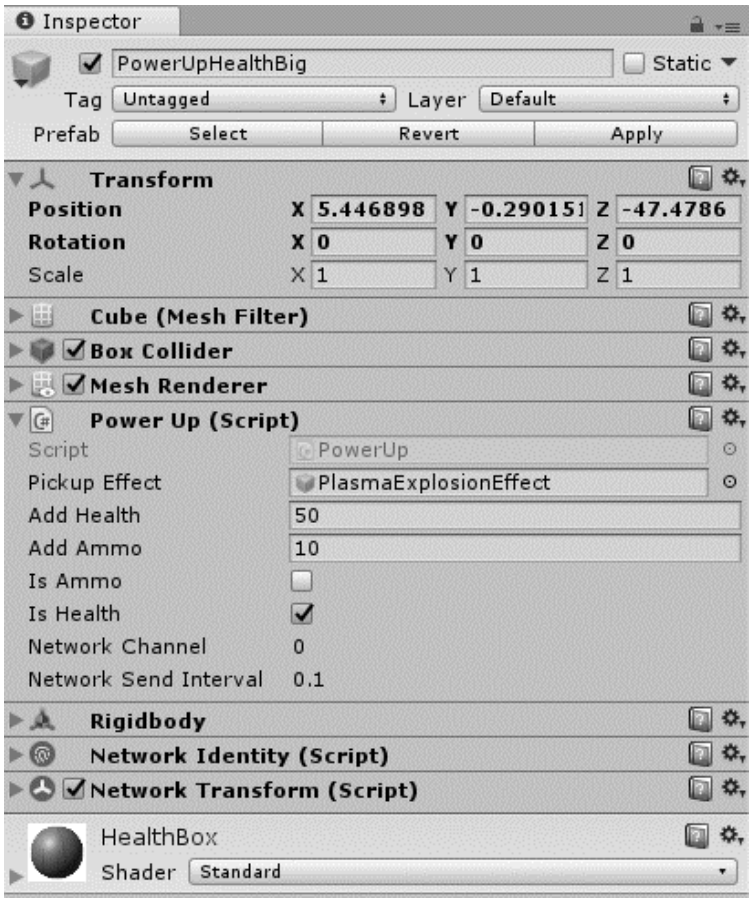
```
public void UpdateAmmoBar(float ammo)
{
      if(m_ammoSlider == null)
            m_ammoSlider = GameObject.Find("Ammo
                  Bar").GetComponent<Slider>();
      m_ammoSlider.value = ammo;
}
```

5.3.2. Predlošci za PowerUp

PowerUp predlošci su predlošci preko kojih će igrač stjecati razne prednosti tijekom igre. Odlučio sam se za izradu predložaka koji će igraču dodavati zdravlje i metke. Za zdravlje i metke sam kreirao po dva predloška:

- *PowerUpHealth*, dodaje igraču 20 jedinica zdravlja.
- *PowerUpHealthBig*, dodaje igraču 50 jedinica zdravlja.
- *PowerUpAmmo*, dodaje igraču 10 metaka.
- *PowerUpAmmoBig*, dodaje igraču 30 metaka.

Za kreiranje predložaka zdravlja i metaka koristio sam običan model 3D kocke na koju sam dodao materijal drvene kutije. Nakon toga sam dodao efekt eksplozije plazme (*PlasmaExplosionEffect* iz *Unity Particle* paketa) koji će se odrađivati kada igrač pokupi predložak. Predlošcima sam zatim dodijelio skriptu preko koje će se igračima omogućiti skupljanje *PowerUp* objekata tijekom igre. Naziv skripte je *PowerUp.cs*, a na slici 32 su prikazana svojstva *PowerUpHealthBig* predloška. Pošto se radi o predlošcima koji aktivno sudjeluju u igri, treba im dodijeliti i komponente za višekorisničko igranje *NetworkIdentity* i *NetworkTransform*.



Slika 32. *PowerUpHealthBig* svojstva

Osim svojstava koje definira skripta *PowerUp.cs*, ona je zadužena i za interakciju predložaka s igračima. Detekcija kupljenja predloška je realizirana preko *OnCollisionEnter* metode koja prvo provjerava adi li se o interakciji s igračem (tag == „Player“). Kupljenjem predloška, igraču se dodaje zdravlje ili meci, te se ažuriraju klizači koji prikazuju preostalo zdravlje i metke. Dohvat igrača koji je pokupio *PowerUp* je realiziran naredbama:

```
CarPlayer carPlayer =
     player.collider.GetComponentInParent(typeof(CarPlayer)) as
          CarPlayer;

WeaponShoot playerShoot =
     carPlayer.GetComponentInChildren(typeof(WeaponShoot)) as
          WeaponShoot;
```

Objekt *carPlayer* sadrži podatke o početnom i trenutnom zdravlju, a objekt *playerShoot* sadrži podatke o početnom stanju metaka i preostalim metcima.

```
if (isHealth) {
     carPlayer.m_currentHealth += addHealth;
     if (carPlayer.m_currentHealth > carPlayer.m_maxHealth)
```

```
            carPlayer.m_currentHealth = carPlayer.m_maxHealth;
      carPlayer.UpdateHealthBar(carPlayer.m_currentHealth);
}

if (isAmmo) {
      playerShoot.m_shootsLeft += addAmmo;
      if (playerShoot.m_shootsLeft > playerShoot.m_bulletShoots)
            playerShoot.m_shootsLeft = playerShoot.m_bulletShoots;
      playerShoot.UpdateAmmoBar(playerShoot.m_shootsLeft);
}
```

Na slici 33 su prikazani gotovi predlošci za zdravlje i metke. Radi lakšeg razlikovanja, predlošci za zdravlje su prikazane plavim plamenom, a za metke crvenim.

Slika 33. *PowerUp* predlošci

5.3.3. PowerUp generator

Kreiranje *PowerUp* predložaka tijekom igre je riješeno preko *GameObjecta* naziva *PowerUpGenerator* kojem sam dodao skriptu *PowerUpGenerator.cs*. Ta skripta slučajnim odabirom kreira jedan od četiri predloška. Skripta ima i mogućnost odabira frekvencije kreiranja predložaka kao i vrijeme trajanja predloška. Početna vrijednost je kreiranje predloška svakih 10 sekundi i samouništenja nakon 30 sekundi u slučaju da niti jedan igrač ne pokupi predložak.

```
public class PowerUpGenerator : NetworkBehaviour
{
      public GameObject health;
      public GameObject healthBig;
      public GameObject ammo;
      public GameObject ammoBig;
      public float spawnRate = 10f;
      public float spawnDestroy = 30f;
      float nextSpawn = 0f;
      int whatToSpawn;
```

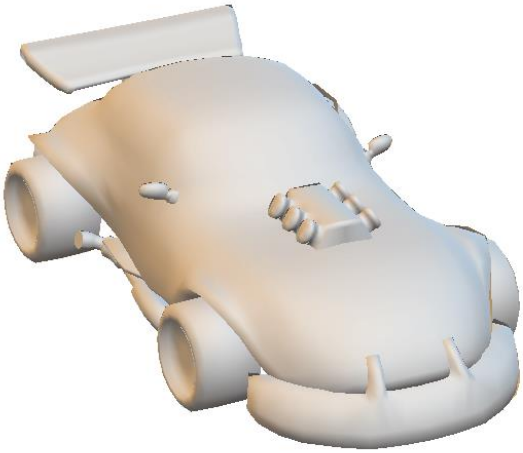
```
    void Update()
    {
        if (Time.time > nextSpawn)
        {
            whatToSpawn = Random.Range(1, 5);
            switch (whatToSpawn)
            {
            case 1:
                CmdCreate(health);
                break;
            case 2:
                CmdCreate(healthBig);
                break;
            case 3:
                CmdCreate(ammo);
                break;
            case 4:
                CmdCreate(ammoBig);
                break;
            }
            nextSpawn = Time.time + spawnRate;
        }
    }

     [Command]
    void CmdCreate(GameObject powerUp)
    {
        GameObject pow = Instantiate(powerUp, transform.position,
            Quaternion.identity);
        pow.GetComponent<Rigidbody>().AddForce(Random.Range(
            -20, 20), Random.Range(-20, 20), Random.Range(
            -20, 20), ForceMode.Impulse);
        NetworkServer.Spawn(pow);
        Destroy(pow, spawnDestroy);
    }
}
```

5.3.4. Kreiranje igrača

Glavni igrač u igri je automobil. Osim što auto mora biti u voznom stanju, zbog borbe mora imati i dodatne borbene funkcionalnost. Mora biti u stanju pucati iz oružja, zadavati štetu drugim igračima te isto tako i primati štetu kada je pogođeno od strane drugih igrača. Za izradu auta sam koristio *Unity* standardni predložak *Sky Car*. Radi se o potpuno funkcionalnom autu futurističkog izgleda prikazanog na slici 34.

Slika 34. *Unity Sky Car* predložak

Za potrebe moje igre, *Sky Car* predlošku ću napraviti neke izmjene. Prvo ću promijeniti oblik auta. To ću napraviti tako da ću u *Unity* učitati model 3D auta prikazanog na slici 35 kojega ću onda jednostavno dodati na *Sky Car* predložak.



Slika 35. 3D model auta

Sky Car predložak sadrži 3D model *CarBody* koji predstavlja tijelo auta. *CarBody* model ću izbrisati i na njegovo mjesto postaviti 3D model moga auta. Nakon toga ću napraviti predložak za novi auto tako da s *drag-and-drop* metodom dovučem auto sa scene u Prefab direktorij i predlošku promijenim naziv u *Multiplayer1*. Pošto se radi o igri trke i borbe s autima, uvijek je dobro imati mogućnost izbora između više auta. *Multiplayer1* predložak ću kopirati u ukupno 4 predloška koristeći prečac *Control+d*, te ću predloške nazvati *Multiplayer2*, *Multiplayer3* i *Multiplayer4*. Da bi auti izgledali ipak malo realnije i da bi se razlikovali, dodat ću i teksture. Svaki auto će imati svoju teksturu koje su prikazane na slici 36.

Slika 36. Teksture za predloške auta

Koristeći *drag-and-drop* metodu, teksture dodjeljujemo pripadajućim predlošcima nakon čega predlošci dobivaju konačan izgled prikazan na slici 37.

Slika 37. Auti

Sada imamo 4 auto predloška kojima treba dodati borbenu funkcionalnost te podršku za višekorisničko igranje. Borbena funkcionalnost je ugrađena preko novog predložaka za oružje, a višekorisnička podrška preko *Unity* mrežnih komponenti. Potrebno je dodati mrežne

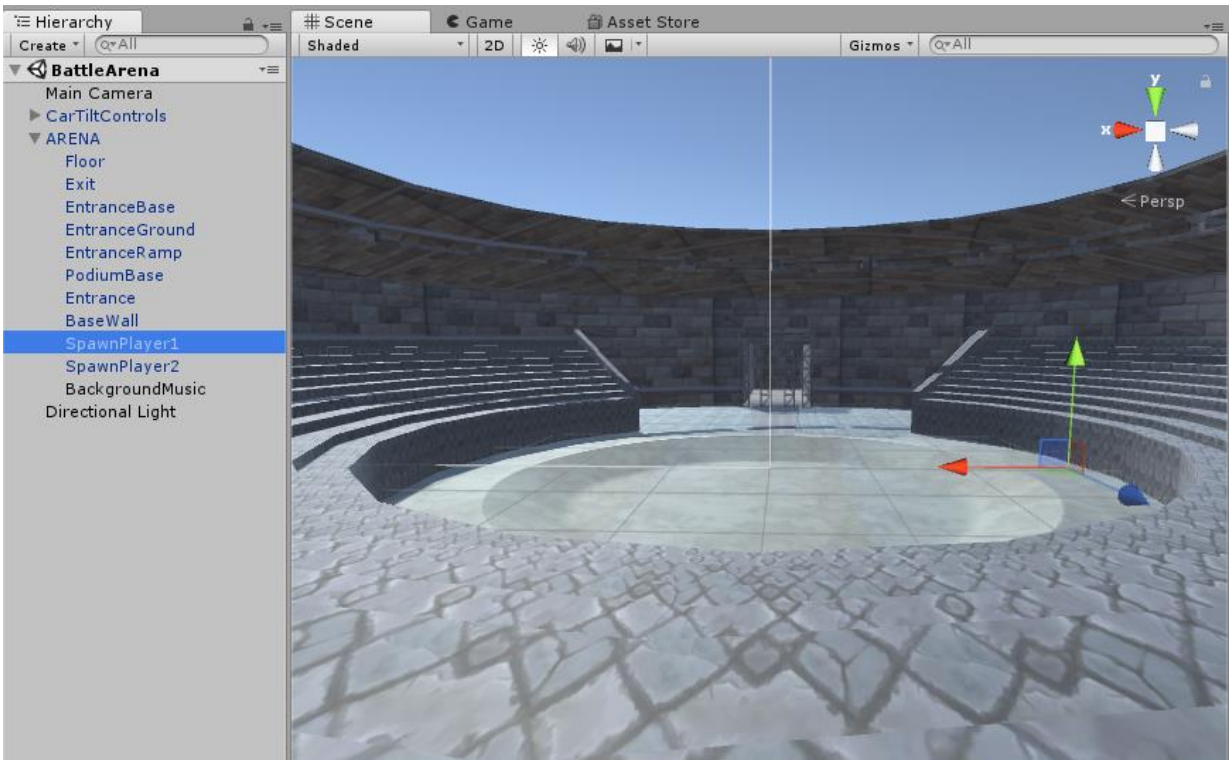
komponente *NetworkIdentity* s odabranom opcijom *Local Player Authoroty* i *NetworkTransform* komponentu koju je potrebno malo doraditi. Zbog brzine vozila, kao i zbog brzine promjene smjera (odnosno rotacije) vozila, potrebno je malo pojačati interpolaciju pokreta i rotacije. Na ovaj način će se osigurati da kretanje protivničkih igrača bude glatko prikazano, bez trzanja i preskakanja. Na kraju, svakom vozilu je potrebno dodijeliti skriptu *PlayerCar.cs*. Preko te skripte se definira pucanje i zdravlje igrača.

5.3.5. Kreiranje borbene arene

Borbena arena je scena u kojoj se odvijaju borbe između igrača. Kao borbenu arenu sam koristio besplatnu postojeću scenu preuzetu sa http://devassets.com/assets/medieval-arena/ .

Slika 38. Borbena arena [28]

Pošto mi nisu potrebni svi detalji arene, dosta elemenata sam izbrisao iz scene i 3D modela arene. Na ovaj način sam i optimizirao izvođenje igre jer je sa stajališta resursa uvijek bolje imati što manje elemenata u sceni. Na slici 38 je prikazana početna verzija arene, a na slici 39 je prikazana arena nakon dorada.



Slika 39. Borbena arena

U arenu sam dodao i četiri prazna *GameObjecta* koje sam nazvao *SpawnPlayer1*, *SpawnPlayer2*, *SpawnPlayer3* i *SpawnPlayer4* te sam im dodijelio mrežnu komponentu *NetworkStartPosition* koja definira poziciju i rotaciju na kojoj se mogu pojaviti igrači. Dodao sam i jedan *GameObject* za pozadinsku muziku kojega sam nazvao *BackgroundMusic* i dodijelio komponentu *AudioSource*. Kao *AudioClip* sam upotrijebio besplatnu muziku preuzetu sa *Unity asset stora*, dostupnu na linku:

https://assetstore.unity.com/packages/audio/music/rock/heavy-pack-free-115854

Da bi arena bila atraktivnija za igru, dodao sam *Physic Material* na zidove arene. *Physic Material* se koristi za podešavanje trenja i odskakanja sudaraćih objekata, a kreira se preko glavnog izbornika odabirom Assets → Create → Physic Material. Nakon što se kreira *Physic Material*, potrebno ga je *drag and drop* metodom dovući na željene sudarače. Sada će se auto uvijek lagano odbijati prilikom sudaranja sa zidovima arene.

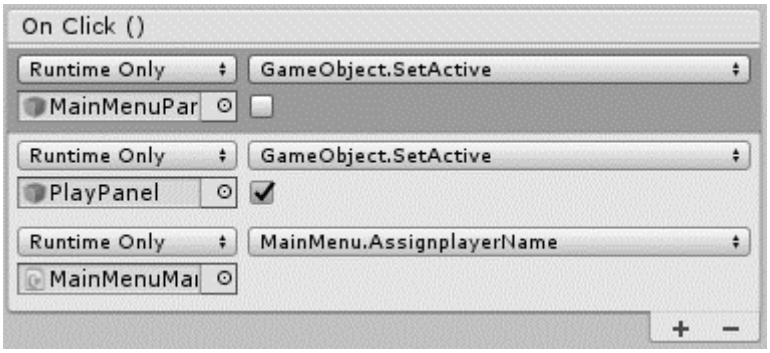
Nakon svega, koristeći *drag-and-drop* metodu arenu je potrebno povući u *Prefab* direktorij i napraviti predložak.

5.3.6. Glavni izbornik

Glavni izbornik je početna scena igre koju sam nazvao *MainMenu*, a preko koje ću povezati cijelu igru. Sastoji od nekoliko podizbornika:

- *Play*, otvara sučelje za kreiranje igre,
- *Settings*, jednostavne općenite postavke,
- *Credits*, prikazuje se ekran sa zahvalama,
- *Quit*, izlaz iz igre.

Svi izbornici i podizbornici su realizirani preko panela. Prijelazom između panela, panelima se postavlja vidljivost. Kada se otvori sučelje za izbor igre preko tipke *Play*, sakriva se panel glavnog izbornika, a prikazuje panel odabira igre. Zatim se klikom na tipku *Back*, sakriva panel za odabir igre, a opet se prikazuje panel glavnog izbornika. Realizacija je prikazana na slici 40.



Slika 40. Vidljivost panela

Kroz izbornike i podizbornike korištene su sljedeće *PlayerPrefs* vrijednosti:

- PlayerPrefs.SetString("PlayerName", m_name);
  Naziv igrača koji se upisuje u polje *PlayerName*. Vrijednost se upisuje prilikom promjene naziva igrača, a čita se prilikom pokretanja igre. Polje *PlayerName* se automatski popunjava nazivom, a prilikom ulaska u arenu ili trku, naziv igrača se prikazuje kao registracija vozila.
- PlayerPrefs.SetInt("Score", m_score);
  Služi za pamćenje ukupnog broja bodova igrača. Čita se prilikom pokretanja igre, a ažurira se sa svakim klikom na tipku *Disconnect* koja se nalazi u scenama borbene vožnje i arene.
- PlayerPrefs.SetInt("ChosenVehicle", m_vehicle);
  Služi za pamćenje postavki odabranog vozila.
- PlayerPrefs.SetString("localhost", m_localhost);

Služi za pamćenje odabranog naziva igre.

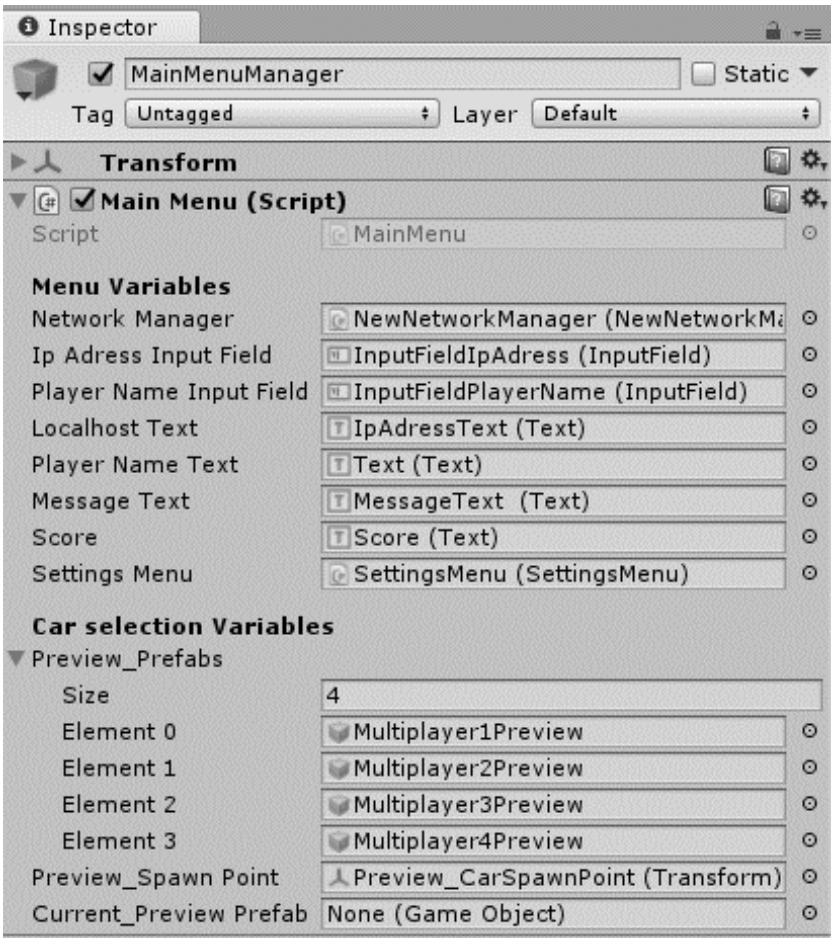
- PlayerPrefs.SetFloat("Volume", m_volume);
  Služi za pamćenje postavki odabrane razine igre.
- PlayerPrefs.SetFloat("Quality", m_quality);
  Služi za pamćenje izabrane razine kvalitete grafičkog prikaza.

Osim podizbornika i izbornika, početna scena *MainMenu* se sastoji i od nekoliko *GameObjecata*:

- *MainMenuManager*,
- *NewNetworkManager*,
- *SettingsManager*.

MainMenuManager

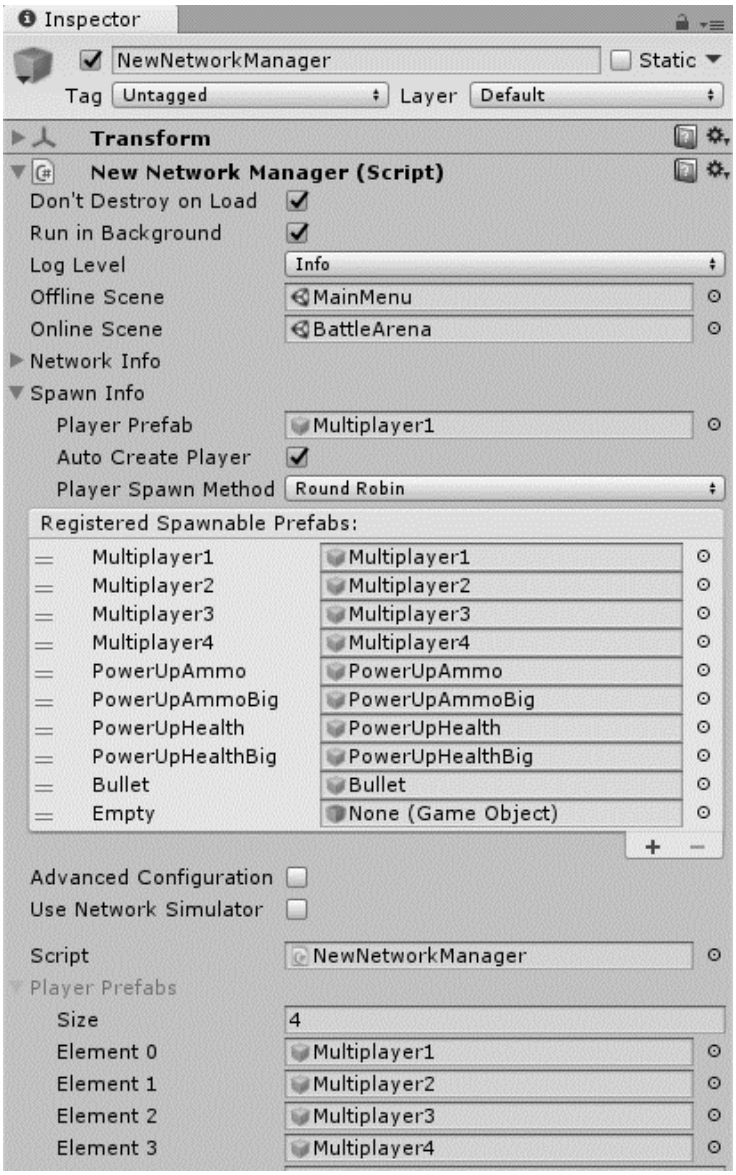
Za kreiranje svakog *GameObjecta* uvijek je najbolje prvo kreirati prazan *GameObject* te mu resetirati položaj i rotaciju. *MainMenuManager* sadrži sve osnovne funkcije koje upotrebljavam preko glavnog izbornika. Svaka funkcija koja se poziva klikom na bilo koju tipku glavnog izbornika, poziva neku od definiranih funkcija. Te funkcije su definirane i realizirane preko skripte *MainMenu.cs* koju sam dodijelio *MainMenuManager GameObjectu*. Odabir auta, prikaz karakteristika odabranog auta te kontrole na unos naziva igrača kontrolira *MainMenuManager*. Da bi sve uredno radilo, potrebno je ispravno popuniti sve elemente kako je prikazano na slici 41.



Slika 41. *MainMenuManager*

NewNetworkManager

Za potporu višekorisničkom igranju potrebno je igri dodijeliti *NetworkManager* komponentu. Ta komponenta se dodaje samo na početnom ekranu igre, u mom slučaju radi se o *MainMenu* sceni. Kreirao sam prazan *GameObject* kojega sam nazvao *NewNetworkManager* te sam mu pridružio *NetworkManager* komponentu.



Slika 42. *NewNetworkManager*

Komponenti je potrebno dodijeliti sve *GameObjecte* kao na slici 42. Posebnu pažnju treba posvetiti registriranju svih objekata koji se mogu instancirati na poslužitelju u igri. To su svi *GameObjecti* kojima je dodijeljena *NetworkIdentity* komponenta. Svi takvi objekti se moraju dodati u *Registered Spawnable Prefabs* odjeljku. Da bi *NewNetworkManager* bio vidljiv kroz sve scene, potrebno je označiti opciju *Don't Destroy on Load*. U poglavlju 4.3.1 sam ukratko opisao *NetworkManager* komponentu. Osim postojećih funkcionalnosti, za potrebe moje igre morao sam dodati neke nove funkcionalnosti. *NetworkManager* komponenta ne podržava mogućnost odabira igrača, tj. igra uvijek kreće s igračem koji je definiran kao *Player Prefab*. Ista stvar je i s odabirom igre. Igra uvijek započinje sa scenom definiranom u *Online Scene*. Nove funkcionalnosti su realizirane novom klasom koju sam definirao skriptom

NewNetworkManager.cs. Ta skripta nasljeđuje *NetworkManager* klasu što znači da će pored onog što sadržio osnovna klasa sadržavati dodatne funkcionalnosti izabranog vozila.

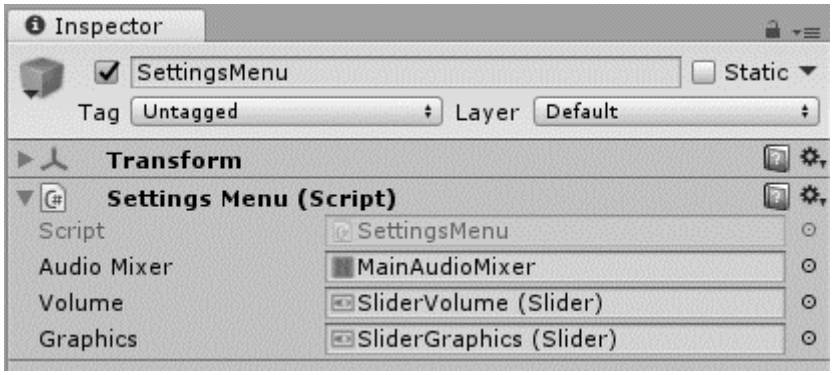
Sadržaj *NewNetworkManager.cs* skripte:

```
public class NewNetworkManager : NetworkManager
{
     public List<GameObject> PlayerPrefabs = new List<GameObject>();

     public override void OnClientSceneChanged(NetworkConnection
conn)
     {
          ChosenVehicleMessage message = new ChosenVehicleMessage();
          message.ChosenVehicle =
               PlayerPrefs.GetInt("ChosenVehicle");
          ClientScene.AddPlayer(conn, 0, message);
     }

     public class ChosenVehicleMessage : MessageBase
     {
          public int ChosenVehicle;
     }

     public override void OnServerAddPlayer(NetworkConnection conn,
          short playerControllerId, NetworkReader
     ChosenVehicleReader)
     {
          ChosenVehicleMessage VehicleMessage =
          ChosenVehicleReader.ReadMessage<ChosenVehicleMessage>();
          int ChosenVehicleInt = VehicleMessage.ChosenVehicle;
          Transform Spawn =
               NewNetworkManager.singleton.GetStartPosition();
          GameObject player =
               Instantiate(PlayerPrefabs[ChosenVehicleInt],
          Spawn.transform.position, Spawn.transform.rotation) as
               GameObject;
          NetworkServer.AddPlayerForConnection(conn, player,
               playerControllerId);
     }
}
```

Odabrano vozilo se dohvaća PlayerPrefs.GetInt("ChosenVehicle") naredbom te se instancira u igri prilikom svake prijave igrača u igru. Zbog toga je bilo potrebno prvo definirati listu s predlošcima svih vozila kao što je prikazano na slici 42. u odjeljku *Player Prefabs*.

SettingsManager

SettingsManager je *GameObject* preko kojega ćemo upravljati osnovnim postavkama igre. Osnovne postavke su klizači za glasnoću zvuka i kvalitetu grafike, te tipka za poništavanje rezultata. Upravljanje vrijednostima postavki je riješeno preko skripte *SettingsMenu.cs* koja mora biti popunjena kao na slici 43. Glasnoća zvuka se regulira preko audio komponente *AudioMixer*. Izmjenom vrijednosti komponente mijenja se glasnoća zvuka kroz cijelu igru. Za kvalitetu grafike zadužena je klasa *QualitySettings*.



Slika 43. S*ettingsManager*

Postavke zvuka su realizirane *SetVolume()* metodom:

```
public void SetVolume(float volume)
{
     audioMixer.SetFloat("volume", volume);
     PlayerPrefs.SetFloat("Volume", volume);
}
```

Poništavanje rezultata je realizirano *ResetScore()* metodom:

```
public void ResetScore()
{
     PlayerPrefs.SetInt("Score", 0);
}
```

Grafičke postavke su *realizirane SetQuality()* metodom;

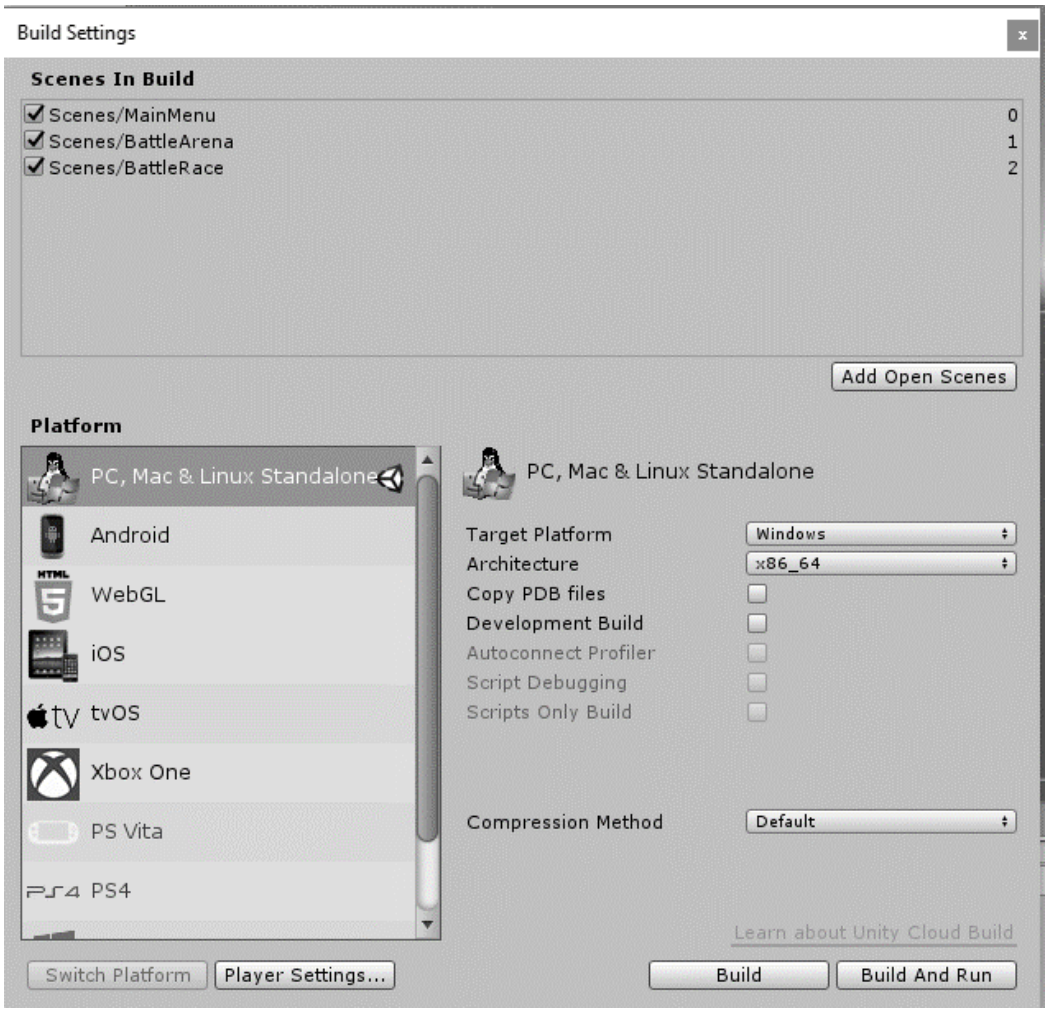
```
public void SetQuality(float quality)
{
     QualitySettings.SetQualityLevel((int)quality);
     PlayerPrefs.SetFloat("Quality", quality);
}
```

Na slici 44 je prikazan izgled sučelja za podešavanje postavki igre.

Slika 44. Postavke igre

5.4. Razvoj igre za različite platforme

Razvoj igara za različite platforme jedna je od značajnijih karakteristika *Unity* okruženja, pa se nudi mogućnost razvijanja i izvršavanja igre na više platformi. U bilo kojem trenutku tijekom stvaranja igre možemo vidjeti kako bi igra izgledala kada bi se pokretala samostalno, mimo *Unity* uređivača. Da bi pristupili postavkama razvoja igre trebamo pristupiti *Build Settings* opcijama do kojih dolazimo preko glavnog izbornika u kojem odabiremo File-> Build Settings. Pojavljuje se popis scena za uređivanje koje će biti uključene u igru kao što je vidljivo na slici 45.



Slika 45. *Build* postavke

Možemo izgraditi samostalne aplikacije za *MS Windows*, *Mac* i *Linux*, sve što treba je odabrati opciju *PC, Mac & Linux Standalone* u dijaloškom okviru te pritisnuti tipku *Build*. Kroz ovaj rad posebnu pažnju sam posvetio razvoju *desktop* igre i igre za mobilni uređaj s *Android* verzijom sustava. Za razvoj na mobilne uređaje sa *Android* sustavom potrebno je odabrati opciju *Android*. Tako razvijena igra je pogodna za sva *desktop* računala te za sve mobilne uređaje s *Android* sustavom (testirano na inačici 6.0.1). Razvoj na više platformi često može rezultirati i slabijom izvedbom. Zbog razlika u razvojnim postupcima, napisani programski kod nije uvijek optimalan za svaki uređaj.

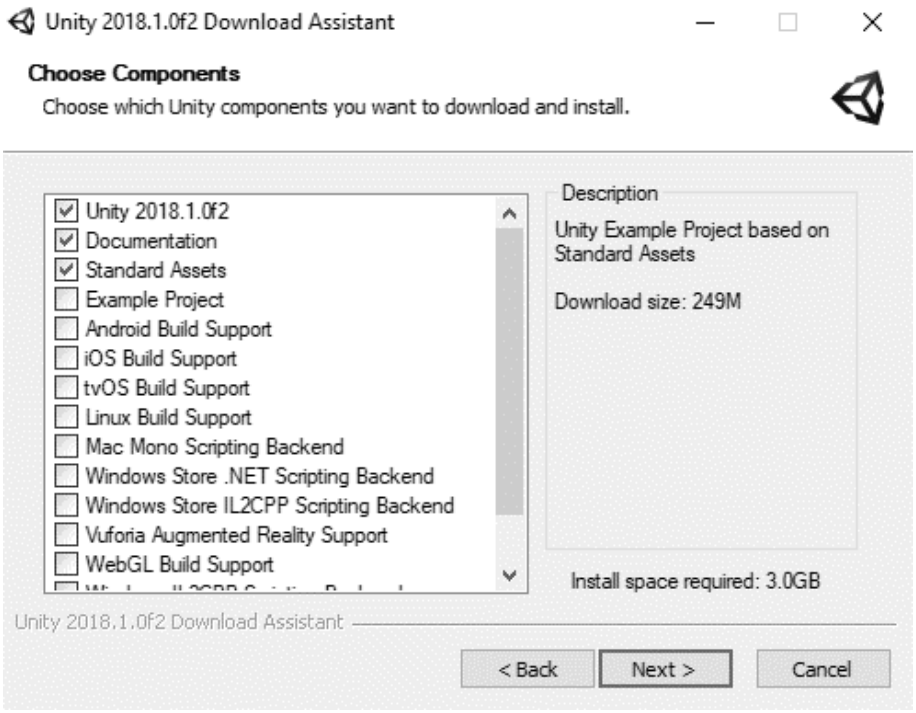
5.4.1. Razvoj na osobna računala

Podržan je razvoj na računala sa operacijskim sustavima *MS Windows*, *macOS* i *Linux*. Sve što treba je odabrati opciju *PC, Mac & Linux Standalone* u dijaloškom okviru prikazanom na slici 45 te pritisnuti tipku *Build*. Kao rezultat nastaju datoteke potrebne za izvođenje igre.

Datoteke će se razlikovati u ovisnosti o operacijskom sustavu odabranom u *Target platform* izborniku.

5.4.2. Razvoj na mobilne platforme - Android

Prije nego krenemo razvijati za mobilne platforme, moramo podesiti razvojno okruženje. Postavljanje je najjednostavnije pomoću *Unity download assistent* aplikacije koja se koristi za instalaciju *Unityja* ili za nadogradnju, dostupne na linku https://store.unity.com/download?ref=personal , prikazane na slici 46.



Slika 46. *Unity Download Assistent* [13]

Potrebno je označiti opciju za *Android Build Support* te klikom na tipku *Next* dovršiti instalaciju ili nadogradnju.

Za starije verzije *Unityja*, postavljanje *Android* razvojnog okruženja uključuje sljedeće korake:

1. Instaliranje *Android* okruženja, preko instaliranja aplikacije *Android Studio* ili instaliranjem samo *Android SDK* paketa dostupnih na adresi https://developer.android.com/studio/index.html
2. Instaliranje *JDK – Java Developer Kita*, dostupnog na adresi http://www.oracle.com/technetwork/java/javase/downloads/index.html

Kompletan proces detaljno je opisan na stranici:

https://docs.unity3d.com/Manual/android-sdksetup.html

Nakon uspješnog postavljanja razvojnog okruženja, napravljena igra se bez većih problema može razviti za mobilne uređaje. U postavkama prikazanima na slici 44 treba odabrati mobilnu platformu *Android* te kliknuti tipku *Build*. Nakon što se *build* odradi, dobit ćemo datoteku s nastavkom apk. To je instalacijska datoteka igre koja se prebacuje na mobilni uređaj i pokretanjem te datoteke izvršava se instalacija igre na mobilnom uređaju. Testiranje igre se također može odraditi preko *Android* emulatora ili fizičkim spajanjem mobitela na računalo. U tom slučaju možemo umjesto tipke *Build* kliknuti na tipku *Build And Run* te će se cijeli posao automatski odraditi.

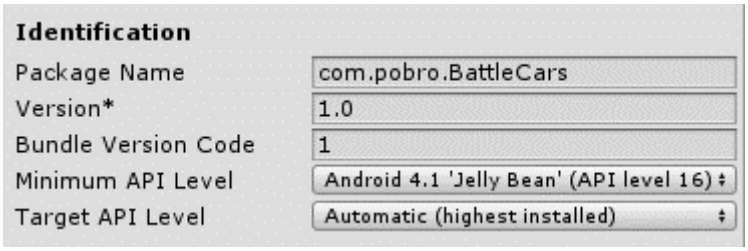
6. Korisnički priručnik

6.1. Instalacija igre

Instalacija igre je poprilično jednostavna i obavlja se u svega nekoliko koraka. Instalacija je moguća na više načina, ovisno na kojoj platformi se obavlja instalacija igre. Instalacijski paket se kreira kroz *Unity*, a za daljnju distribuciju se koriste kreirani instalacijski paketi.

6.1.1. Instalacija igre na mobilne uređaje

Instalacija igre na mobilne uređaje uključuje izvoz aplikacije u apk format koristeći mogućnosti razvojne okoline. U *Build Settings* opcijama prikazanim na slici 44 potrebno je odabrati opciju *Android*. Klikom na tipku *Build* nastaje apk datoteka koju zatim treba prebaciti na mobilni uređaj s *Android* verzijom. Prije klika na tipku *Build*, potrebno je u opcijama *Player Settings* popuniti podatke o nazivu igre te o firmi. Podaci o firmi su proizvoljni tako da pojedinac koji razvija igru ne mora voditi brigu o firmi već može upisati proizvoljni naziv. Također postoji i mogućnost odabira *Android* verzija, kao što je prikazano na slici 47.



Slika 47. Odabir *Android* verzije

Ako se igra distribuira putem trgovine mobilnih aplikacija (npr. *Google Play*), nakon izbora igre, kreće download i automatska instalacija igre na uređaj.

6.1.2. Instalacija igre na desktop računala

Instalaciju igre je moguće odraditi na *MS Windows*, *macOS* i *Linux* operacijske sustave. Ja sam koristio *MS Windows* operacijski sustav. U *Build Settings* opcijama prikazanim na slici 44 potrebno je odabrati opciju *PC, Mac & Linux Standalon*e. Klikom na tipku *Build* nastaju datoteke potrebne za igranje igre na stolnim računalima:

- BattleCarsPC.exe - izvršna datoteka,

- UnityPlayer.dll - dll datoteka,
- BattleCarsPC_Data - direktorij s datotekama potrebnim za igru.

Navedene datoteke se odnose samo na *MS Windows* operacijske sustave. Daljnja distribucija i instalacija igre na druga računala s *MS Windows* operacijskim sustavima se obavlja kopiranjem navedenih datoteka. Za ostale operacijske sustave postupak je isti, samo što za *Linux* i *macOS* nastaju aplikacijski paketi.

6.2. Igranje

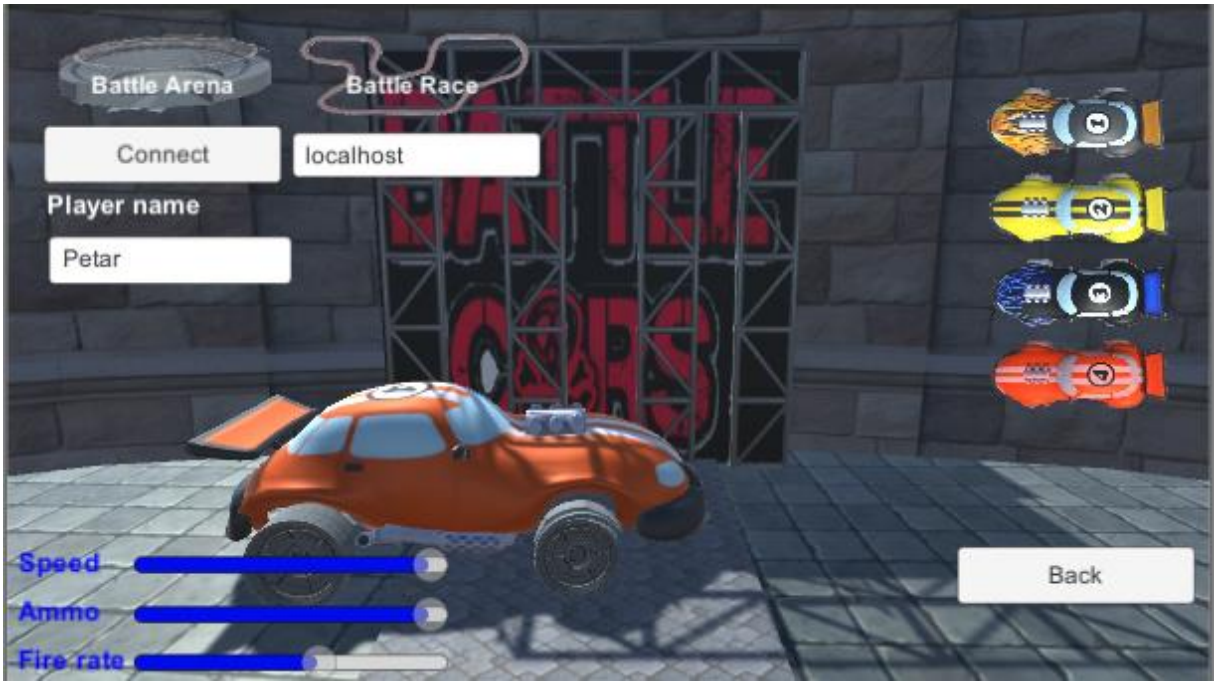
Pokretanjem igre prikazuje se početni izbornik s dostupnim opcijama, što je prikazano na slici 48.

Slika 48. Battle Cars

Dostupne opcije su:

- *Play*, otvara se sučelje za kreiranje igre
- *Settings*, jednostavne generalne opcije
- *Credits*, prikazuje se ekran sa zahvalama
- *Quit*, izlaz iz igre

Kod osobnih računala, odabir opcija se radi pomoću miša, a verzija za mobilne uređaje radi na dodir. Odabirom opcije *Play* prikazuje se sučelje kao na slici 49.



Slika 49. Sučelje za kreiranje igre

Postoje dvije vrste igranja: *Battle Arena* i *Battle Race*. Klikom na tipku *Battle Arena* pokreće se borba u areni, a klikom na tipku *Battle Race* se pokreće utrka. Uređaj koji će biti domaćin igri odabire vrstu igre i naziv, a ostali igrači se pridružuju igri klikom na tipku *Connect*. Prije same utrke treba upisati naziv igrača u polje ispod labele *Player name* te naziv igre u polju označenom labelom *localhost*. Za mobilne uređaje kao *localhost* se mora upisati IP adresa uređaja koji je domaćin, a svi mobilni uređaji moraju biti spojeni na pristupnu točku koju kreira domaćin.

Na ovom sučelju se također i odabire vozilo kojim se želi upravljati tijekom igre. U desnom dijelu sučelja prikazana su sva 4 dostupna vozila. Klikom na pojedino vozilo, u središtu sučelja prikazuje se uvećani model odabranog vozila sa svojim karakteristikama. Vrijednosti karakteristika prikazane su grafički vrijednostima odgovarajućeg klizača.

PC verzija igre koristi kursorske tipke za kontrolu vozila te lijevu tipku miša za pucanje. Android verzija se sastoji od posebnog kontrolora preko kojega se upravlja vozilom i puca.

6.2.1. Battle Arena

U ovoj vrsti igre igrači igru počinju u borbenoj areni gdje se svatko bori protiv svakoga. Uništenjem protivničkog igrača dobiva se jedan bod, a uništeni igrač se automatski ponovno kreira u areni. U areni je posebno programirano kreiranje *PowerUpova* za metke i zdravlje.

Slika 50. *Battle Arena*

Na slici 50 je prikazana igra u borbenoj areni. Klizači za zdravlje i broj metaka prate stvarno stanje. Nakon što se svi metci potroše, vozilo više nema mogućnost pucanja. Nakon što se potroši zdravlje, igrač se uništi te se ispočetka kreira. Početno zdravlje svakog igrača iznosi 100 bodova. Zbog povećanja igrivosti početni broj metaka je 80, a svaki metak koji pogodi vozilo, oduzima mu po 2 boda zdravlja. Igra traje sve dok ima aktivnih igrača, a pobjednik je igrač s najvećim brojem bodova. Klikom na tipku *Disconnect* igrač napušta borbenu arenu i vraća se na početni izbornik.

6.2.2. Battle Race

U ovoj vrsti igre cilj je pobijediti u utrci. Pobjeda se ostvaruje tako da igrač prvi dovrši utrku ili eliminira sve ostale sudionike utrke. Posebnost ove vrste igranja je u tome što se na

početku igre čeka na sve ostale igrače tj. utrka ne može početi dok se svi igrači ne uključe u igru. Početak igre prikazan je na slici 51.

Slika 51. *Battle Race*

To je riješeno jednostavnom skriptom koja kontrolira aktivan broj igrača u igri. Kada skripta otkrije drugoga igrača, kreće odbrojavanje za start utrke. Utrka se vozi jedan krug. Kroz stazu su postavljenje tri kontrolne točke koje se brinu o tome da igrač stvarno odvozi cijeli krug. Kontrolne točke su nevidljive te se aktiviraju prolaskom vozila kroz njih. Prva kontrolna točka predstavlja startnu liniju, druga se nalazi na oko polovice staze, a treća predstavlja ciljnu liniju. Prolaskom preko ciljne linije odrađuje se kontrola koja provjerava je li igrač prošao sve tri kontrolne točke. U slučaju uspješne kontrole, igrač je uspješno odvozio krug utrke.

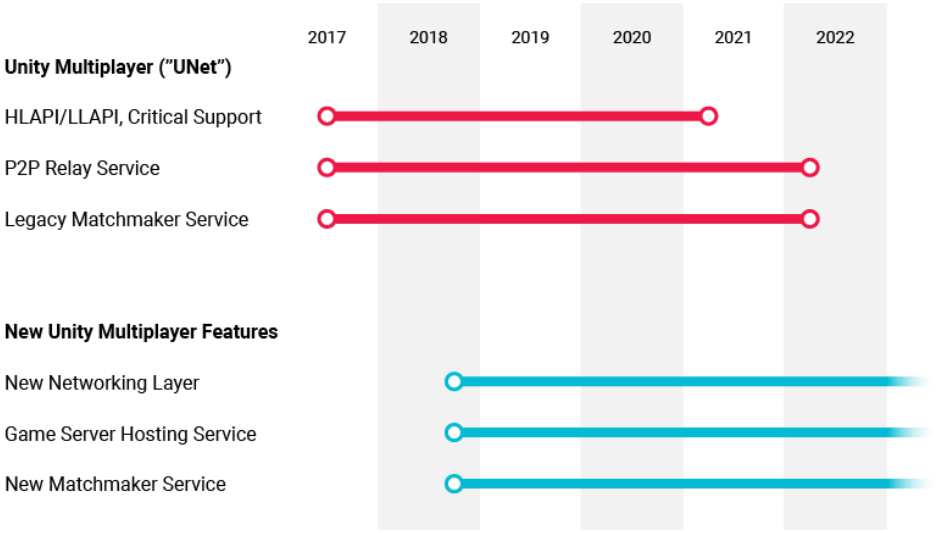
Klikom na tipku *Disconnect* igrač napušta borbenu arenu i vraća se na početni izbornik.

7. ZAKLJUČAK

Razvojno okruženje je okosnica na kojoj počiva svaka video igra, od jednostavnih *Android* igara, sve do najzahtjevnijih i najnaprednijih PC naslova. Razvojno okruženje podržava sve mogućnosti napredne igre – od fizike, preko grafike, umjetne inteligencije pa do mrežnog igranja. Danas postoji mnogo okruženja i teško je odabrati najboljeg. *Unreal* i *CryEngine* se najčešće spominju kao najnaprednija rješenja u području razvoja igara. Međutim, najnaprednije ne mora nužno biti i najprihvatljivije rješenje. Za velike timove i već uhodane programere možda *Unreal* ili *CryEngine* jesu prihvatljiva rješenja, ali što ako se radi o manjim timovima ili pojedincima koji su možda početnici? Oni bi možda trebali provesti i nekoliko godina na učenju programiranja i razvoja igara. Toliko vremena da bi se možda nešto za nekoliko godina napravilo ne djeluje previše motivirajuće. Početni entuzijazam bi brzo nestao, a igra teško da bi napredovala. Da bi se razvoj igara približio što većem broju ljudi bilo je potrebno napraviti neko pristupačnije i jednostavnije razvojno okruženje dostupno svima. 2004. godine nastao je *Unity*, a od 2009. postoji potpuno besplatna verzija.

Kroz ovaj rad napravio sam višekorisničku igru koristeći *Unity* razvojno okruženje. Bez problema sam kreirao borbene scene, vozila, importirao 3D objekte, koristio razne komponente (fizika, zvuk, kamera, mrežne komponente), a kako znam programirati u programskom jeziku *C#*, napisao sam i potrebne skripte. U kratko vrijeme savladao sam sučelje i shvatio osnove *Unity* razvojnog okruženja te su prvi rezultati bili ne samo brzo vidljivi već i poprilično zadovoljavajući. Višekorisničku podršku sam poprilično brzo ostvario ali samo u najjednostavnijoj varijanti. Za izbor scena i vozila te evidenciju zdravlja i pogodaka sam se ipak morao malo više potruditi. Od dodataka sam kreirao predloške za *PowerUp* koji predstavljaju dodatno zdravlje i metke. Proces razvijanja igre se dosta može vremenski skratiti upotrebom *Unity* trgovine predložaka. U trgovini ima dosta besplatnih elemenata ali ponekad možda nije loše i uložiti nekoliko kuna, naravno u skladu s mogućnostima, za elemente koji bi nam značajno olakšali rad. Puno je jednostavnije i brže kupiti 3D model nego učiti modelirati kroz *Blender* ili neki slični alat. Osim gotovih 3D modela, u trgovini se može pronaći i dosta skripti, gotovih predložaka, materijala, zvukova i svega ostaloga što bi nam moglo zatrebati u izradi ili unapređenju igre. Mogućnosti za unapređenje igre ima nebrojeno. Unapređenje vozila, štit, dodatno oružje, rakete, brzina, dodatne scene, samo su neke od mogućnosti. Dodatno nagrađivanje u vidu novčića kojima se kupuju dodaci bi također bilo zanimljivo. Naravno, ima još dosta prostora za napredak i poboljšanja. Igranje preko Interneta bi svakako bilo veliko poboljšanje. Nakon izrade igre mogu zaključiti da *Unity* ima dosta mogućnosti te da je jednostavan za korištenje. Jednom kada se korisnik privikne na sučelje te savlada osnovne pojmove poput komponenti i predložaka sve ideje se mogu lako realizirati. Ipak, moram navesti

mrežnu komponentu razvojnog okruženja kao nepotpunu. Da, *Unity* jest jednostavan i ima veliku zajednicu kao i dokumentaciju. Međutim, moj zaključak je da je mrežni dio ipak ostao nedorečen, naročito u segmentu dokumentacije. Nije dovoljno samo navesti i ukratko objasniti mrežne komponente. Izrada višekorisničke igre je veliki posao koji nije lako odraditi. Nedostaje praktičnih primjera razvoja višekorisničke igre, a prvi znak da se nešto ne razvija kako treba je nedostatak potpune višekorisničke podrške. Toga su vjerojatno bili svjesni i u *Unity Technologies* jer su tijekom kolovoza 2018. objavili planove za podršku novoga višekorisničkog okruženja. Na slici 52. je prikazan plan podrške za *Unet* i novo mrežno okruženje.



Slika 52. Podrška za *Unity* višekorisničko okruženje [13]

Podrška za *Unet* će ostati sve do 2021. godine, a novo višekorisničko okruženje će biti dostupno već krajem ove godine. Vjerujem da će novo mrežno okruženje biti veliki korak naprijed u razvoju višekorisničkih igara, ne samo onih jednostavnih već i zahtjevnijih igara. To bi svakako bio korak naprijed u pravome smjeru jer višekorisničko igranje zauzima veliki segment u industriji video igara.

8. LITERATURA

1. *Blizzard Entertainment, World of Warcraft First Official Infographic*, dostupno na URL: http://media.wow-europe.com/infographic/en/world-of-warcraft-infographic.html
2. *Wikipedia*, dostupno na URL: https://en.wikipedia.org/wiki/Multiplayer_video_game (pristupljeno 11.9.2018.)
3. *Epyx Inc, Pit Stop II*, dostupno na: URL https://upload.wikimedia.org/wikipedia/en/0/02/Pitstop_ii-c64.png
4. *Electroncs Arts, Star Wars Battlefront*, dostupno na URL: www.ea.com
5. *Ubisoft, Heroes of Might & Magic*, dostupno na URL: www.ubisoft.com/en-gb/game/might-and-magic-heroes-7
6. *Newzoo, Global market research*, dostupno na URL: https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/ (pristupljeno 23.06.2018.)
7. *Unity Technologies, Unity* razvojno okruženje, dostupno na URL: https://unity3d.com/
8. *Photon*, dostupno na URL: https://www.photonengine.com/en/pun
9. *uLink*, dostupno na URL: http://developer.muchdifferent.com
10. *Forge*, dostupno na URL: https://developers.forgepowered.com/
11. *PlayFab*, dostupno na URL: https://playfab.com
12. *DarkRift*, dostupno na URL: https://darkriftnetworking.com/
13. *Unity user manual*, dostupno na URL: docs.unity3d.com (pristupljeno 1.5.2018.)
14. *Wikipedia*, dostupno na URL: https://en.wikipedia.org/wiki/Video_game_development (pristupljeno 11.9.2018.)
15. *Unreal* razvojno okruženje, dostupno na URL: https://www.unrealengine.com/
16. *CryEngine* razvojno okruženje, dostupno na URL: https://www.cryengine.com/
17. *Construct* razvojno okruženje, dostupno na URL: https://www.scirra.com/
18. *PlayCanvas* razvojno okruženje, dostupno na URL: https://playcanvas.com/
19. *Godot* razvojno okruženje, dostupno na URL: https://godotengine.org/
20. *GameMaker* razvojno okruženje, dostupno na URL: https://www.yoyogames.com/
21. *Artjoker, Game engines*, dostupno na URL: https://artjoker.net/assets/images/engblog/blog-Vlad/Mobile-Game-Development/Game-engines.png
22. *Maya*, dostupno na URL: https://www.autodesk.com/products/maya/overview
23. *Blender*, dostupno na URL: www.blender.org
24. *Cinema 4D*, dostupno na URL: https://www.maxon.net/en-us/

25. *3D Studio Max*, dostupno na URL: https://www.autodesk.com/products/3ds-max/overview
26. *Wikipedia*, *Polygon mesh*, dostupno na URL: https://en.wikipedia.org/wiki/Polygon_mesh (pristupljeno 23.06.2018.)
27. *Unity Asset Store*, dostupno na URL*:* https://assetstore.unity.com/ (pristupljeno 11.9.2018.)
28. *DevAssets*, dostupno na URL: http://devassets.com/assets/medieval-arena/

IZJAVA

Izjavljujem pod punom moralnom odgovornošću da sam diplomski rad izradio samostalno, pod stručnim vodstvom mentora doc. dr. sc. Krunoslav Žubrinića. U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio izvore podataka navedena u diplomskom radu.

Petar Obrovac

Petar Obrovac