Razvoj mrežne igre za više igrača u okruženjima Unity i Photon

Crnković, Edi

Master's thesis / Diplomski rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:346572

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-23**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

EDI CRNKOVIĆ

RAZVOJ MREŽNE IGRE ZA VIŠE IGRAČA U OKRUŽENJIMA UNITY I PHOTON

Diplomski rad

Pula,svibanj 2019.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

EDI CRNKOVIĆ

RAZVOJ MREŽNE IGRE ZA VIŠE IGRAČA U OKRUŽENJIMA UNITY I PHOTON

Diplomski rad

JMBAG: 0303033026, redoviti student

Studijski smjer: diplomski sveučilišni studij Informatika

Znanstveno područje: društvene znanosti

Znanstveno polje: informacijske i komunikacijske znanosti

Znanstvena grana: informacijski sustavi i informatologija

Predmet: Dizajn i programiranje računalnih igara

Mentor: doc. dr. sc. Tihomir Orehovački

Pula, svibanj 2019.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Edi Crnković, kandidat za magistra informatike, ovime izjavljujem da je ovaj diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da ni jedan dio diplomskog rada nije napisan na nedozvoljen način, odnosno prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

U Puli, 14. svibanj 2019.

IZJAVA

o korištenju autorskog djela

Ja, Edi Crnković, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Razvoj mrežne igre za više igrača u okruženjima Unity i Photon“ koristi na način da navedeno autorsko djelo kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu diplomskih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu sa Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na navedeni način ne potražujem naknadu.

U Puli, 14. svibanj 2019.

Potpis

Sadržaj:

1. UVOD 1
2. VIDEOIGRE 3
2.1. Povijest videoigara 3
2.2. Višekorisnička igra 5
3. DIZAJN MREŽE I RAZVOJ PROTOKOLA 6
3.1. Izazovi pri razvoju mrežne igre 6
3.2. Arhitektura mreže 7
3.3. Protokoli 8
3.3.1. TCP 9
3.3.2. UDP 10
3.4. Topologija mreže 10
3.4.1. Klijentsko-poslužiteljska arhitektura 10
3.4.2. Mreža ravnopravnih korisnika 12
3.4.3. Deterministička mreža ravnopravnih korisnika 14
4. UNITY 15
4.1. Uvod u Unity 15
4.2. Unity urednik 15
4.2.1 Alatna traka 16
4.2.2. Prikaz scene 16
4.2.3. Prikaz igre 17
4.2.4. Hijerarhija 18
4.2.5. Preglednik projekta 19
4.2.6. Inspektor 19
4.2.7. Postavke izgradnje 20
4.2.8. Trgovina elementima 21
4.3. Glavna načela djelovanja Unityja 23
4.3.1. Komponente 23
4.3.2. Objekt 24
4.3.3. Prefab 24
4.3.4. Oznake 24
4.3.5. Scene 25
4.4. Skripte 25
5. PHOTON UNITY NETWORKING 27
5.1. Pregled značajki 27
5.2. Početno postavljanje 28

5.3. Bitne komponente ... 29
5.4. Matchmaking ... 31
5.5. RPC ... 31
5.6. Instantiation ... 32
5.7. Prava igrača ... 33
5.8. Sinkronizacija stanja ... 34
6. IZRADA IGRE ... 35
6.1. Izrada elemenata igre ... 35
6.2. Kreiranje igrača ... 35
6.2.1. Predlošci za PowerUp ... 36
6.2.2. Predložak za oružje ... 38
6.3. Lobby Scene ... 41
6.4. Kreiranje staze ... 46
6.5. Cilj ... 47
7. KORISNIČKE UPUTE ... 49
7.1. Instalacija ... 49
7.2. Igranje ... 49
7.2.1. Utrka ... 50
8. ZAKLJUČAK ... 51
Literatura ... 52
Popis slika ... 54
Programski kodovi ... 54
Sažetak ... 56
Summary ... 56

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. Za razliku od jednokorisničkih igara (engl. *Singleplayer*), koje postavljaju igrače na unaprijed programirane izazove ili protivnike s kontroliranom umjetnom inteligencijom, višekorisničke igre omogućuju igračima da komuniciraju s drugim pojedincima u partnerstvu, natjecanju ili suparništvu, osiguravajući im interakciju u društvu. Tipično, višekorisničke igre zahtijevaju od igrača da dijele resurse sustava jednog igrača ili da koriste mrežnu tehnologiju kako bi zajedno igrali na većim udaljenostima.

Kako su se tehnologije umrežavanja razvijale i napredovale, tako su i igre za više igrača. Od brzog povećanja dostupnosti interneta u 1990-ima, nakon čega su uslijedila poboljšanja u brzini povezivanja u ranim 2000-im. Kroz svoje internetske veze, igrači su sada u mogućnosti družiti se s čak tisućama drugih pojedinaca iz cijelog svijeta na istom poslužitelju. Moderni razvoj igara u velikoj se mjeri usredotočuje na društvene interakcije i višekorisničke aspekte igranja. Fokus na stvaranje zajednice oko igara može se vidjeti na igrama poput Clash Royale (Supercell, 2016), World of Tanks (Wargamings, 2016), PlayerUnknown's Battlegrounds (PUBG Corporation, 2017). Svaka od ovih igara implementira višekorisničku značajku na svoj način, ali zajednički faktor je da koriste poslužitelja kako bi omogućili tu razinu interakcije.

Budući da je socijalizacija važan aspekt modernog igranja, implementacija mrežnih značajki je nešto što treba razmotriti tijekom razvoja svakog većeg izdanja, kao i kod manjih neovisnih igara. Upravo iz tih razloga u ovom radu sam istražio mrežno rješenje kako bi se implementirala *online* funkcionalnost u prototip igre. Izbor alata za kreiranje *online* igre bili su Unity 3D, koji je danas jedan od najpopularnijih alata za razvoj igara, te mrežni dodatak Photon Unity Networking (PUN). Cilj diplomskog rada je objasniti strukturu mrežnih igara te kroz praktičnu primjenu prikazati mogućnosti Unityja i Photon Unity Networking u kreiranju funkcionalne *online* igre. Svrha diplomskog rada je obrana dogovorene teme s osvrtom na moguću praktičnu primjenu. Diplomski rad se temelji na predavanjima iz kolegija, osobnim iskustvima u praktičnom radu te na literaturi navedenoj u popisu literature.

U drugom poglavlju je kratko opisana industrija videoigara i razlika između jednokorisničkih i višekorisničkih igara.

Treće poglavlje bavi se mrežnom arhitekturom kod videoigara, obradit ću različite mrežne topologije i protokole koji se koriste te raspravljati o njihovim različitim učincima.

U četvrtom i petom poglavlju opisao sam razvojni alat Unity 3D i Photon koji sam koristio za stvaranje igara. Riječi će biti ukratko o nastanku i razvoju, sučelju te najčešće korištenim mogućnostima alata.

U šestom poglavlju predstavljen je prototip projekta zajedno s ilustrativnim opisima o tome kako su ostvarene mrežne funkcionalnosti.

Zaključak je predočen u sedmom poglavlju, gdje će se dati spoznaje do kojih se došlo razradom teme te popisom značajki koje su planirane za daljnji razvoj.

2. VIDEOIGRE

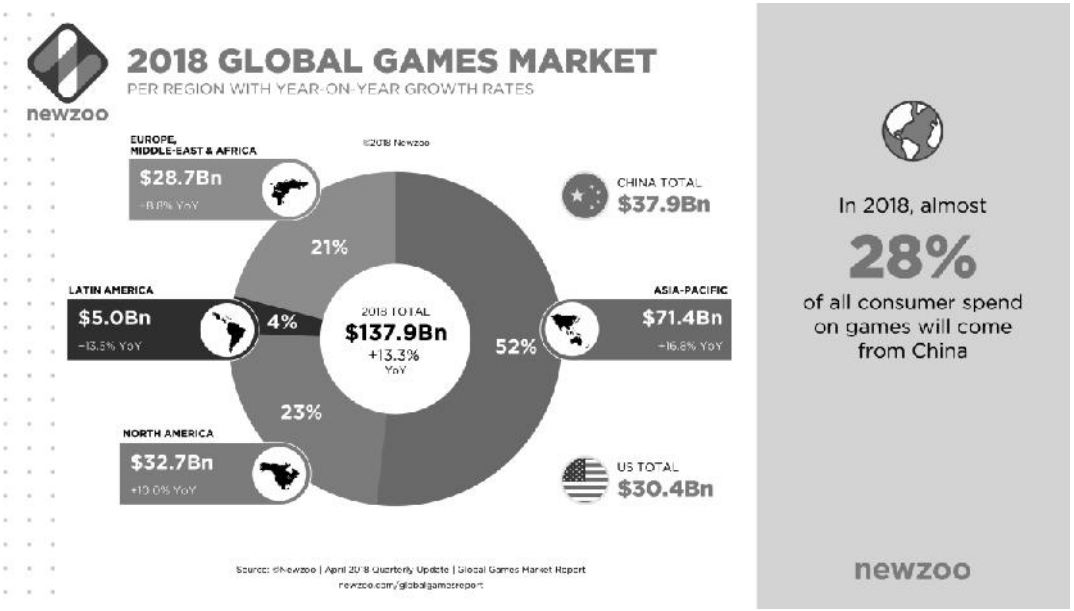
Videoigra je elektronička igra koja uključuje interakciju s korisničkim sučeljem za generiranje vizualnih povratnih informacija na dvodimenzionalnom ili trodimenzionalnom uređaju za prikaz videa, kao što su TV zaslon, naočale za virtualnu stvarnost (engl. virtual reality headset) ili monitor računala [1]. Od 1980-ih, video igre postaju sve važniji dio industrije zabave, i neka vrsta umjetnosti.

2.1. Povijest videoigara

Prva komercijalna videoigra bila je Computer Space, koja potječe iz 1971. godine, a to je bila arkadna verzija igre Spacewar koja potječe iz 1962. godine (Esposito, 2005:4). Prema PEGI-ju[1] (Pan European Game Information), to je bila prekretnica kada je u pitanju razvoj videoigara, a do danas taj razvoj se može objasniti kroz četiri razdoblja. Prvo razdoblje bilo je od 1971. do 1978. godine, a karakteriziraju ga počeci videoigara i prvi uspjesi, drugo razdoblje bilo je od 1978. do 1983. godine te ga možemo nazvati „zlatnim dobom" kada je došlo do podjele videoigara po žanrovima, treće razdoblje koje karakteriziraju manja tehnološka ograničenja i jake ideje na području razvitka videoigara bilo je od 1983. do 1994. godine, a posljednje razdoblje počelo je 1994. godine i prema Espositu traje još dandanas, a to je doba 3D-a, PlayStationa, PC-a te internetskih igara.

Industrija videoigara je gospodarski sektor uključen u razvoj, marketing i monetizaciju videoigara. Obuhvaća desetke radnih disciplina i zapošljavaju tisuće ljudi širom svijeta. Prema istraživanju tvrtke Newzoo „Godina u pregledu 2018", globalni prihod od industrije igara u 2018. iznosi 137,9 milijardi američkih dolara te se očekuje da će do 2021. iznositi 180 milijardi američkih dolara [1]. Sljedeća slika prikazuje prihod videoigara u 2018. godini.

[1] Sveeuropski informacijski sustav za ocjenjivanje igara koji se primjenjuje u većini europskih zemalja te ima potporu Europske unije (https://pegi.info/, pristupljeno 21. 11. 2017.)



Slika 1. Prihod od videoigara u 2018. [1]

Prema PEGI klasifikaciji, videoigre i računalne igre možemo podijeliti u sljedeće žanrove:

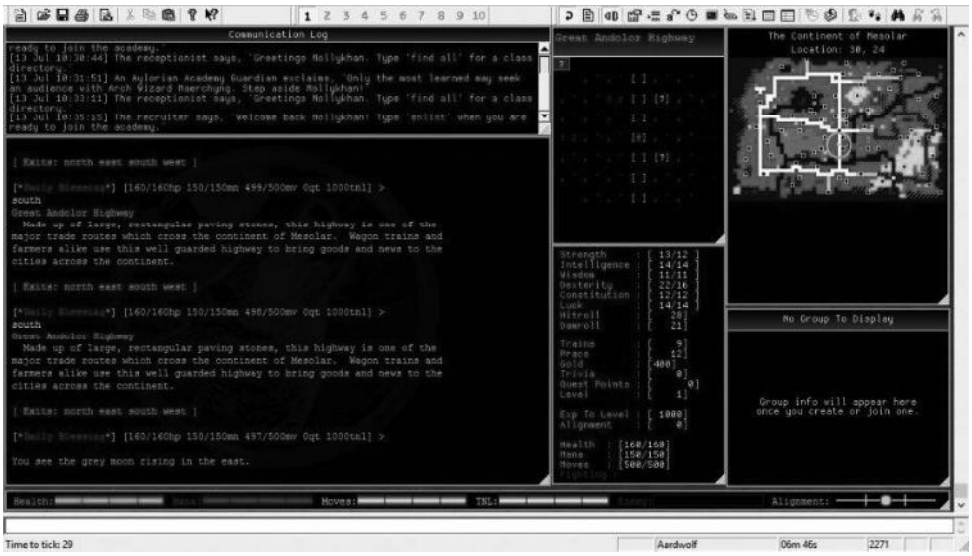
- Akcijsko-avanturističke gdje igrač kontrolira lika u igrici pomoću rješavanja zagonetki ili upotrebom borilačkih vještina.
- Avanturističke gdje se igračima dodjeljuje uloga, a cilj je rješavanje zagonetki.
- MMO (Massively Multiplayer Online gaming) je tip igre koja uključuje tisuće igrača iz cijeloga svijeta, a svatko se može priključiti putem interneta.
- Platformske gdje se igra temelji na skakanju ili prolasku kroz platformu.
- Igre slaganja (*puzzle*) su jednostavne, a obično se igraju putem ručne konzole.
- RPG (*Rolle playing games*) ili igre uloga koje uključuju borbu.
- Igre utrke koje uključuju simulaciju vožnje kroz određeni put kako bi se došlo do cilja u najkraćem mogućem roku; igru je moguće igrati pojedinačno ili protiv drugih natjecatelja.
- Ritmično-plesne igre koje uključuju glazbu, ritam i ples.
- Igre pucanja gdje je cilj upucati objekte ili likove u igri.
- Igre simulacije koje uključuju simulaciju realnih životnih aktivnosti.
- Sportske igre koje uglavnom uključuju simulaciju određenog sporta.
- Strateške igre koje uglavnom uključuju strategije kako bi se došlo do cilja.

2.2. Višekorisnička igra

Prva i osnovna razlika između jednokorisničkih i višekorisničkih igara je broj igrača koji igraju igru. Dok su videoigre s jednim igračem obično usmjerene oko jednog igrača koji se natječe s protivnicima AI i postizanjem unaprijed određenih ciljeva, igre za više igrača osmišljene su oko interakcije s drugim ljudskim igračima. Ove interakcije mogu biti u obliku natjecanja, partnerstva ili jednostavno društvenog angažmana.

Popularnost interneta donijela je mogućnost da se igrači širom svijeta povežu i igraju zajedno na sasvim novi način. Za razliku od starih LAN zabava[2], igrači bi sada mogli igrati i natjecati se s drugim igračima iz cijelog svijeta bez napuštanja udobnosti vlastitog doma.

Povijest *online* višekorisničkih igara može se pratiti sve do ranih primjera kao što je MUD (Multi-User Dungeon) koji vidimo na Slici 2. je virtualni svijet u stvarnom vremenu za više igrača, koji se obično temelji na tekstu, gdje korisnici mogu igrati jednostavne RPG-ove[3] preko interneta.



Slika 2. Aardwolf MUD [2]

Online višekorisničkih igara danas obuhvaćaju gotovo sve žanrove igara, od strijelaca u prvom licu do strateških igara u stvarnom vremenu. Internetsko igranje također je stvorio novi žanr igara pod nazivom masivne višekorisničke igre (engl. *Massively Multiplayer Online* (MMO)). U MMO-u masivne količine igrača mogu se povezati i međusobno djelovati u jednoj instanci ili svijetu. Jedan od najpopularnijih MMO igara danas je World of Warcraft [2].

[2] LAN zabava je skup ljudi s računalima ili kompatibilnim konzolama za igre, gdje je veza između lokalne mreže (LAN) uspostavljena između uređaja koji koriste *router* ili *switch*

[3] Igre igranja uloga (engl. *RPG, Role Playing Games*)

3. DIZAJN MREŽE I RAZVOJ PROTOKOLA

Dva najveća razmatranja pri dizajniranju i razvoju višekorisničke igre je odlučivanje o mrežnoj topologiji i protokolu povezivanja za korištenje. Svaki izbor ima značajan utjecaj na provedbu i samu igru. U sljedećem dijelu poglavlja opisani su različiti izazovi koji se javljaju pri kreiranju višekorisničke igre, mrežne topologije i protokoli koji se koriste te će se raspravljati o njihovim različitim učincima.

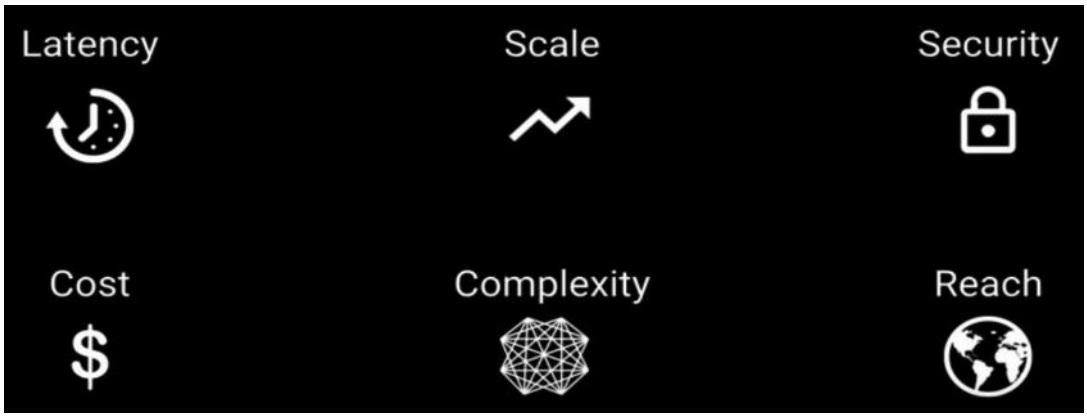
3.1. Izazovi pri razvoju mrežne igre

Višekorisničke igre u stvarnom vremenu dale su ogroman tehnološki napredak posljednjih godina. Konzolni i PC naslovi poput Overwatch (Blizzard Entertainment, 2016) i Total War (The Creative Assembly, 2000) imali su milijune igrača, a sada su čak i mobilne igre rađene za višekorisničko igranje. Kako uređaji postaju snažniji, igrači zahtijevaju kvalitetna iskustva, kao standard.

Pokretanje uspješne mrežne igre u stvarnom vremenu uključuje pružanje dosljedne i stabilne usluge. Iz tehničke perspektive, to znači osmišljavanje zabavne igre koja će dosljedno povezivati igrače s ljudima na njihovoj razini igre sve dok se izbjegava kašnjenje, pad i prekid veze.

Neki od temeljnih izazova s kojima se suočavaju igre za više igrača u realnom vremenu prikazani su na Slici 3. i objašnjeni u nastavku [3]:

- Latencija – vrijeme čekanja odnosno vrijeme koje protekne od trenutka zahtjeva za podacima do trenutka dok se ti podaci ne pojave.
- Razmjer – koliko igrača želite imati u istom prostoru za igru.
- Sigurnost – koliko je lako igru hakirati i stavlja li bilo koje druge strojeve korisnika pod rizik.
- Cijena – postići razumnu cijenu po korisnika tako da se može postići profitabilnosti .
- Kompleksnost – koliko kompleksna je cijela tehnologija i koliko će inženjerski rad biti potreban.
- Opseg – koliko ljudi je u mogućnosti ući u igru.

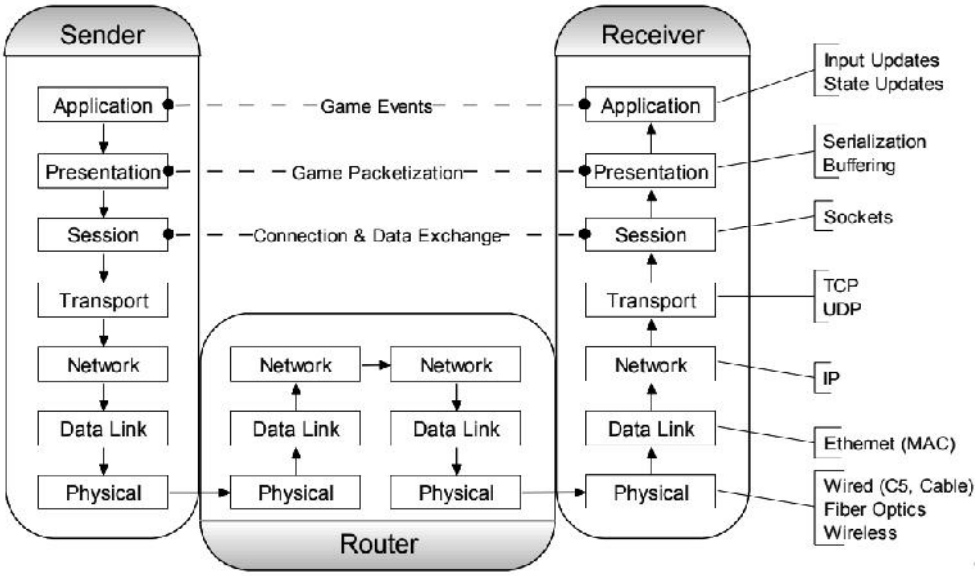


Slika 3. Izazovi pri razvoju višekorisničke igre [3]

Način na koji će se riješiti svaki od tih problema i koji problem ima veći prioritet ovisi o vrsti igre koja se razvija.

3.2. Arhitektura mreže

Internacionalna organizacija za standarde (ISO) je 1978. g. definirala standard od 7 razina (slojeva) za povezivanje računala u mrežu pod nazivom OSI referentni model (Open System Interconnection Reference Model). OSI model u suštini opisuje načine upravljanja funkcijama prijenosa podataka. Svaki sloj pruža usluge sloju koji je neposredno iznad njega [4]. Primjer OSI modela možemo vidjeti na sljedećoj slici.



Slika 4.OSI-model [4]

Slojevi OSI modela od najnižeg prema najvišemu su:

Fizički sloj (engl. *Physical Layer*) sastoji se od fizičkih komponenti u lokalnoj mreži (LAN) kao što su mrežni hardver i *ethernet* kabeli.

Sloj podatkovne veze (engl. *Dana Link Layer*) odgovoran je za stvaranje okvira koji se kreću preko mreže. Ovi okviri enkapsuliraju pakete i koriste MAC adrese za identificiranje izvora i odredišta.

Mrežni sloj (engl. *Network Layer*) odgovoran je za kreiranje paketa koji se kreću preko mreže. Ona koristi IP adrese za identifikaciju izvora i odredišta paketa.

Transportni sloj (engl. *Transport Layer*) uspostavlja vezu između aplikacija koje se izvode na različitim hostovima. Koristi TCP za pouzdane veze i UDP za brze veze. Prati procese koji se prikazuju u aplikacijama iznad njega tako da im dodjeljuje brojeve portova i koristi mrežni sloj za pristup TCP/IP mreži

Sloj sesije (engl. *Session Layer*) nalazi se na sloju 5 i upravlja postavljanjem i otkidanjem povezanosti između dvije komunikacijske krajnje točke. Komunikacija između dvije krajnje točke poznata je kao veza.

Prezentacijski sloj (engl. *Presentation Layer*) osigurava da su komunikacije koje prolaze kroz njega u odgovarajućem obliku za aplikaciju primatelja. Drugim riječima, prikazuje podatke u čitljivom formatu iz perspektive aplikacijskog sloja. Na primjer, program prezentacijskog sloja može formatirati zahtjev za prijenos datoteka u binarnom kodu kako bi se osigurao uspješan prijenos datoteke. Budući da je binarni najosnovniji od računalnih jezika, on osigurava da će uređaj za primanje moći dešifrirati i prevesti ga u format koji aplikacijski sloj razumije i očekuje.

Aplikacijski sloj (engl. *Application Layer*) je protokol koji igra koristi. To su poruke kao što su događaji u vezi početka igre, igrači koji se pridružuju, igrači koji ažuriraju svoje pozicije, oružje ili inventar.

3.3. Protokoli

Internet se uglavnom temelji na dva protokola – TCP i UDP. Oba ova protokola imaju prednosti i nedostatke u svrhu razvoja višekorisničke igre. I TCP i UDP prosljeđuju pakete podataka s uređaja pomoću portova na različite usmjerivače dok ne stignu do konačnog odredišta. Također se koriste za slanje paketa na IP adresu primatelja [4].

I TCP i UDP rade na vrhu IP-a (Internet Protocol). Zbog toga se može čuti izraze kao što su TCP/IP ili UDP/IP. Međutim, budući da se TCP/IP i UDP/IP koriste vrlo često, oni se nazivaju samo TCP i UDP.

Iako su TCP i UDP najčešće korišteni protokoli, oni nisu jedini koji se koriste za prijenos paketa podataka. Drugi protokol koji se može koristiti je ICMP (Internet Control Message Protocol). Međutim, budući da se većina veza oslanja na TCP ili UDP, usredotočit ćemo se na ova dva.

3.3.1. TCP

TCP je kratica za Transmission Control Protocol te je došao prije UDP-a. Često će ga se vidjeti kao TCP/IP, iako ne postoji razlika između toga i TCP-a.

IP protokol razdvaja podatke u pakete i šalje ih na odredište preko interneta, ali kako te pakete vratiti zajedno kada stignu? Za to je izumljen TCP. Kada paketi dođu na svoje odredište, uređaj za primanje ih ponovno spaja u njihov izvorni oblik.

TCP zahtijeva da obje strane komuniciraju kako bi uspostavile vezu i poslale podatke. TCP jamči da će primatelj primati pakete po redoslijedu prema rednim brojevima uključenim u zaglavlje. Primatelj će poslati poruku pošiljatelju za svaki paket, potvrđujući da su primljeni. Svi paketi koje primatelj ne potvrdi ponovno se šalju.

Zbog svega toga, između klijenta i poslužitelja, TCP može pouzdano osigurati integritet podataka koji se razmjenjuju putem interneta. Jednostavno rečeno, može jamčiti da će podaci stići točno onako kako su poslani bez izmjena ili dijelova koji nedostaju. To čini TCP korisnim za velik broj aplikacija te je najčešće korišten protokol na internetu. Svaki put kada se klikne veza, preuzme datoteka u *web*-pregledniku, ažurira aplikacija ili otvori e-pošta, vjerojatno se koristi TCP [5].

Međutim, sva ta povratna komunikacija usporava TCP. Ako paket nestane, cijela se radnja zadržava sve dok se ne pošalje ponovno. Iako se u stvarnom životu to prevodi samo u milisekunde, to može utjecati na performanse za aplikacije koje zahtijevaju veliku propusnost kao što su višekorisničke igre.

3.3.2. UDP

UDP je kratica za User Datagram Protocol. UDP je također izgrađen na IP protokolu, radi slično kao TCP-u, ali je jednostavniji i brži. Glavna razlika je u tome što UDP ne zahtijeva od primatelja da potvrdi da je svaki paket primljen. Svi paketi koji se gube u tranzitu ne vraćaju se. Na taj način računala brže komuniciraju, ali primljeni podaci možda se ne podudaraju s poslanim podacima. UDP paketi nemaju brojeve redoslijeda, tako da mogu doći redom kojim nisu poslani. Oni ipak imaju kontrolne zbrojeve, tako da su paketi koji stižu zaštićeni od korupcije ili modifikacije u tranzitu.

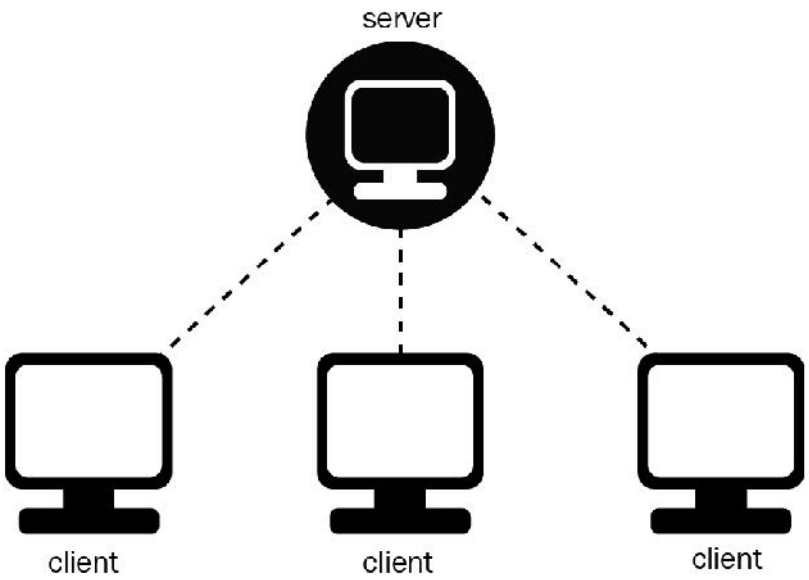
Iz tog razloga, UDP se koristi kada se preferira brzina u odnosu na integritet i ispravljanje pogrešaka. Neke uobičajene aplikacije uključuju *streaming* video i glazbu, prijenos uživo, glasovne i videopozive i *online* igre. U ovim scenarijima, nije bitno ako se izgubi povremeni videookvir što pogoduje UDP-u [5].

3.4. Topologija mreže

Jednostavno rečeno, mrežna topologija je način na koji su računala na mreži međusobno povezana. Za *online* igre, topologija mreže će odrediti kako su računala u mreži organizirana kako bi se korisnicima omogućilo primanje ažuriranja za igru. Kako su računala umrežena, odredit će mnoge aspekte cjelokupnog višekorisničkog dizajna i svaka vrsta topologije ima svoje prednosti i slabosti. U sljedećem odjeljku opisat ću tri najpopularnije topologije koje se koriste u razvoju igara: klijent – poslužitelj model, model ravnopravnih računala i deterministička mreža ravnopravnih korisnika.

3.4.1. Klijentsko-poslužiteljska arhitektura

U topologiji klijent – poslužitelj (engl. *client server model*), jedna instanca je označena kao poslužitelj, a sve druge instance igrača spajaju se na njega. Svaka instanca igrača će samo komunicirati s poslužiteljem. Poslužitelj je, pak, odgovoran za komunikaciju svih ažuriranja igrača drugim klijentima povezanim na mreži [6]. Sljedeća slika prikazuje ovu topologiju mreže:



Slika 5. Klijent – poslužitelj arhitektura [6]

Iako nije jedini način, mreža klijent – poslužitelj obično provodi autoritativni dizajn. To znači da, dok igrač izvodi radnju, kao što je premještanje njihova karaktera na drugo mjesto, ta se informacija šalje u obliku ažuriranja poslužitelju. Poslužitelj provjerava smatra li se ažuriranje ispravnim, a ako jest, poslužitelj zatim prenosi ove informacije o ažuriranju ostalim igračima koji su spojeni na mrežu. Ako postoji situacija u kojoj se klijent i poslužitelj ne slažu s informacijama o ažuriranju, poslužitelj se smatra ispravnom verzijom.

Baš kao i kod mreže ravnopravnih korisnika, postoje neke stvari koje treba razmotriti pri implementaciji. Kada govorimo o propusnosti, u teoriji, zahtjev za propusnim opsegom za svakog igrača neće se mijenjati ovisno o broju priključenih igrača. Međutim, za razliku od mreže ravnopravnih korisnika, topologija klijent – poslužitelj je asimetrična, što znači da će poslužitelj imati jedan prema jedan po klijentu. To znači da će se s povećanjem broja priključenih igrača povećati širina pojasa potrebna za podršku veza. Međutim, u praksi, kako se više igrača pridružuje, potrebno je simulirati više objekata, što može uzrokovati neznatno povećanje zahtjeva za propusnim opsegom i za klijenta i za poslužitelja

Bitni nedostatak klijent – poslužitelj arhitekture je ako klijent izvodi radnju, kao što je premještanje iz jedne točke u drugu, te se informacije šalju poslužitelju. Poslužitelj provjerava je li informacija ispravna, a zatim ažurira stanje igre. Nakon toga propagira informacije svim klijentima, kako bi mogli ažurirati svoje stanje igre. Takva radnja dodaje značajno zaostajanje za sve ulazne podatke, što je naravno potpuno neprihvatljivo. Da bi se igra mogla igrati s ovim modelom, potrebne su različite vrste „trikova“ kako bi se postigla sinkronizacija.

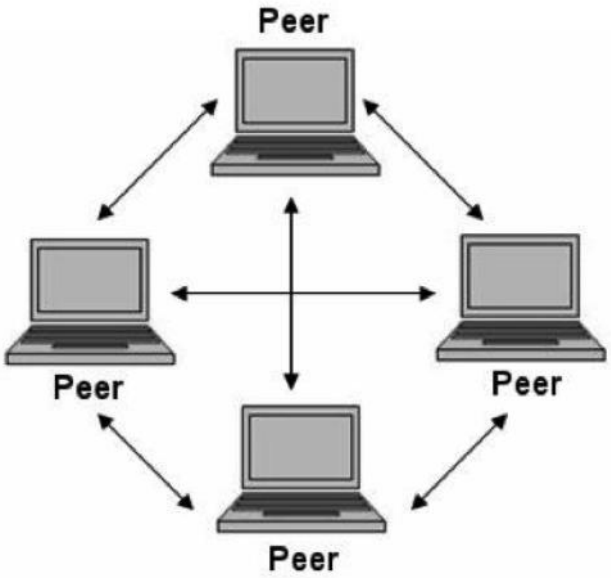
Jedan od trikova opisan je u članku „must-read article by Valve“ [7].
Osnovna ideja je:

- Kada igrač pritisne gumb, klijent ga odmah obrađuje kao da ima ovlasti za to, pokreće animaciju i slično. Poruka se šalje i poslužitelju.
- Poslužitelj će pritisnuti tipku malo kasnije, pa će se poslužitelj premotati na vrijeme pritiska tipke, izvršiti ga, a zatim ponovno simulirati na trenutno vrijeme.
- Poslužitelj zatim šalje klijentu trenutno stanje.
- Klijent dobiva najnovije stanje, ali u međuvremenu je prošlo više vremena. Tako se klijent vraća na vrijeme u kojem je poslužitelj poslao poruku, ispravlja svoje stanje s onim što je autoritativni poslužitelj odlučio, a zatim ponovno simulira lokalno na trenutno vrijeme.

Drugim riječima: i klijent i poslužitelj premotavaju, a zatim ponovno simuliraju svaki put kad se paket primi. Primjena mehanizama premotavanja je složen zadatak i vrlo je teško dodati postojećoj igri.

3.4.2. Mreža ravnopravnih korisnika

Mreža ravnopravnih korisnika (engl. *Peer to Peer*) obično se implementira u neautoritativnom dizajnu. U neautoritativnoj skupini ne postoji središnji entitet i svaki korisnik (*Peer*) kontrolira svoje stanje igre. U mreži ravnopravnih korisnika, korisnik šalje podatke svim ostalim korisnicima i prima podatke od njih, pod pretpostavkom da su informacije pouzdane i točne. Slika 6. prikazuje arhitekturu mreže ravnopravnih korisnika.



Slika 6. Mreža ravnopravnih korisnika [7]

Svako računalo ima cijelu kopiju aplikacije pomoću koje se odvija cijeli sustav. Stoga je svako računalo zaduženo za izvedbu svih ulazno-izlaznih operacija i mrežnu komunikaciju što podrazumijeva slanje informacije o promjeni stanja svim drugim računalima, za što je u prijašnjoj arhitekturi bio zadužen poslužitelj. Iz toga je očigledno kako to znatno doprinosi povećanju prometa na mreži koji raste kvadratno s rastom broja radnih stanica. Budući da sada stanica mora obavljati puno više posla nego klijent u prijašnjoj arhitekturi, sama stanica zauzima više resursa.

Najveći nedostatak mreže ravnopravnih korisnika je da zaostajanje postaje mnogo nepredvidljivije. Dok u arhitekturi klijentskog poslužitelja samo igrač koji zaostaje pati od vlastitog kašnjenja, u mreži ravnopravnih korisnika ostali igrači će također primijetiti ako jedan igrač ima lošu internetsku vezu [8].

Glavna prednost ovakve arhitekture je njezina robusnost. Budući da ne postoji centralizirani sustav, ne postoji ni usko grlo rušenja sustava. Tako će prilikom gašenja jedne stanice sve ostale stanice i cijeli sustav nastaviti funkcionirati [8].

Ostale prednosti P2P arhitekture je da se ulazni podaci igrača mogu odmah obraditi. Nije potrebna nikakva dodatna struktura premotavanja i nema nikakvog ulaznog kašnjenja za uređaj.

3.4.3. Deterministička mreža ravnopravnih korisnika

Posljednja osnovna struktura je deterministička mreža ravnopravnih korisnika (engl. *Deterministic peer to peer lockstep*). Ovaj model se uglavnom koristi za strategije u realnom vremenu[4]. To je također model ravnopravnih korisnika, ali ovdje se ne moramo brinuti o tome koji igrač upravlja kojim objektima. Umjesto toga svaki klijent simulira sve na isti način [9]. Jedina stvar koju treba poslati putem mreže je akcija svakog igrača.

Igra se odvija u mnogo kratkih pokreta: svaki korak u igri prikuplja naredbe od svih igrača preko mreže, a zatim simulira sljedeći korak. Ovo nije ograničeno na potezne igre (engl. *turn-based*): čineći puno stvarno kratkih koraka može se osjećati kao igra u stvarnom vremenu.

Prednost ove mreže je da je potrebno poslati samo akcije igrača. Ako svatko započne igru u istoj situaciji i izvodi iste korake, igra će ostati usklađena bez slanja ažuriranja preko mreže. Stoga je ovaj model vrlo pogodan za strategijske igre, budući da imaju toliko jedinica da je sinkroniziranje svega često nemoguće.

Loša strana ovog modela je u tome što obično dodaje dosta zaostajanja kontrolama, jer se akcije ne mogu izvršiti dok svi igrači ne znaju za njih. Takvo kašnjenje unosa može se sakriti reproduciranjem zvukova i vizualnih efekata odmah kada korisnik klikne. Na taj način igrač neće primijetiti da njegove jedinice ne reagiraju odmah.

Deterministička mreža ravnopravnih korisnika također se može kombinirati s modelom klijent – poslužitelj, mrežom gdje podaci uvijek teku kroz poslužitelj umjesto izravno između svih igrača.

[4] Strategija u realnom vremenu (engl. *Real-time strategy*) je žanr ratnih računalnih igara, čije odvijanje nije oslovljeno potezima kao kod poteznih strategija.

4. UNITY

4.1. Uvod u Unity

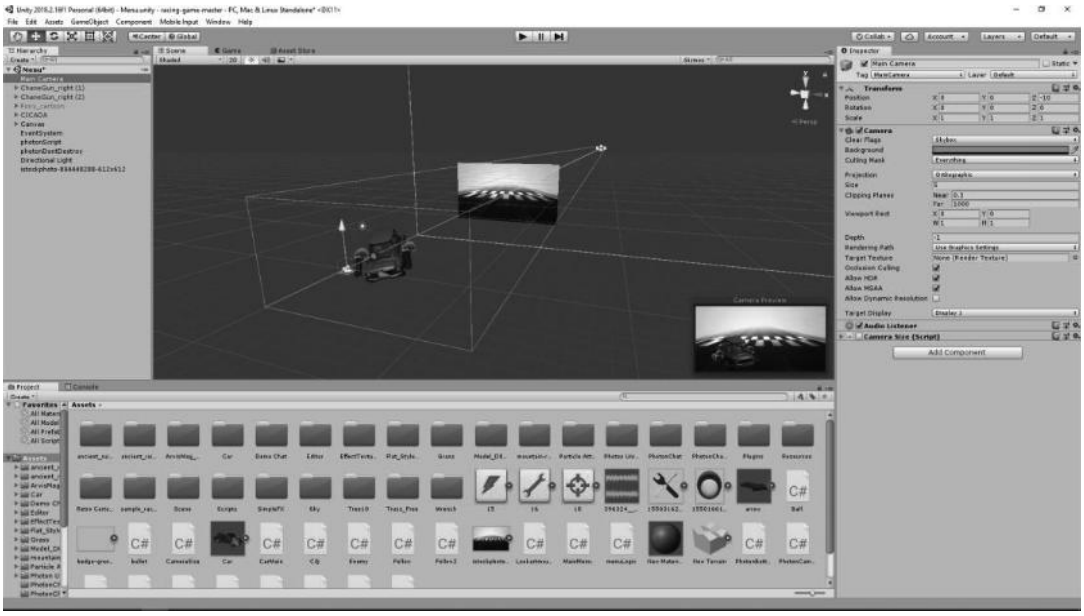
Unity je platforma za razvoj igara koju je razvila tvrtka Unity Technologies i koristi se za razvoj 2D ili 3D videoigara za računala, konzole, mobilne uređaje ili *web*-preglednike. Posebno popularan među proizvođačima *indie* igara, Unity se može pohvaliti s više od 4 milijuna registriranih korisnika širom svijeta. U 2018. godini dominira kao najpopularniji globalni softver za razvoj igara [10].

Prvobitno izdan 2005. samo za Mac OS X i od tada prošireno, Unity je 2019. godine došao do svoje 5.3.4 verzije. Velika prednost Unityja je da omogućuje programerima igara da lakše ciljaju nekoliko različitih platformi sa samo manjim promjenama u programiranju, kao što su kontrole igrača ili druge specifične optimizacije za uređaj. Sam editor može se jednostavno proširiti dodacima drugih proizvođača, a sadržaj koji generira korisnik može se pronaći i preuzeti iz Trgovina elementima (engl. Asset Store). Ovi korisnički generirani sadržaji mogu biti u rasponu od zvučnih efekata i skripti do 3D modela i više.

Uspješna zajednica razvila se oko Unityja. Forumi zajednice su prepuni savjeta ili rješenja za različite vrste tema, dok https://unity3d.com/learn nudi sveobuhvatne lekcije za početnike. Jedinstveno osobno izdanje (engl. *personal edition*) dostupno je besplatno za preuzimanje, dok stručno izdanje (engl. *professional edition*) košta 75 američkih dolara mjesečno. Pretplatnici profesionalnog izdanja dobivaju pristup za pregled dodatnih statističkih podataka i detaljnu analizu svoje igre, uz različite opcije prilagodbe, rukovanja *bugovima* i pregled drugih profesionalnih alata. Osobno izdanje ne može biti licencirano ili korišteno od strane subjekta s godišnjim bruto prihodom ili proračunom većim od 100.000 američkih dolara [11].

4.2. Unity urednik

Glavno sučelje urednika Unity (engl. Unity editor) sastoji se od različitih prozora s karticama koji se mogu preraspodijeliti, grupirati, odvojiti i usidriti. Svaki od ovih prozora ili pogleda ima vlastite svrhe i funkcionalnost u razvoju igre. Najvažniji zadani prikazi su: Scena, Igra, Hijerarhija, Projektna ploča i Inspektor (engl. Scene, Game, Hierarchy, Project panel and Inspector). Osim toga, prozor konzole može se pregledavati za pogreške, upozorenja i druge poruke koje urednik ili korisnik generira za pomoć u otklanjanju pogrešaka.
Glavno sučelje urednika može se pregledati na Slici 7.



*Slika 7. Glavno sučelje Unitya (*Izvor: snimka zaslona autora*)*

4.2.1 Alatna traka

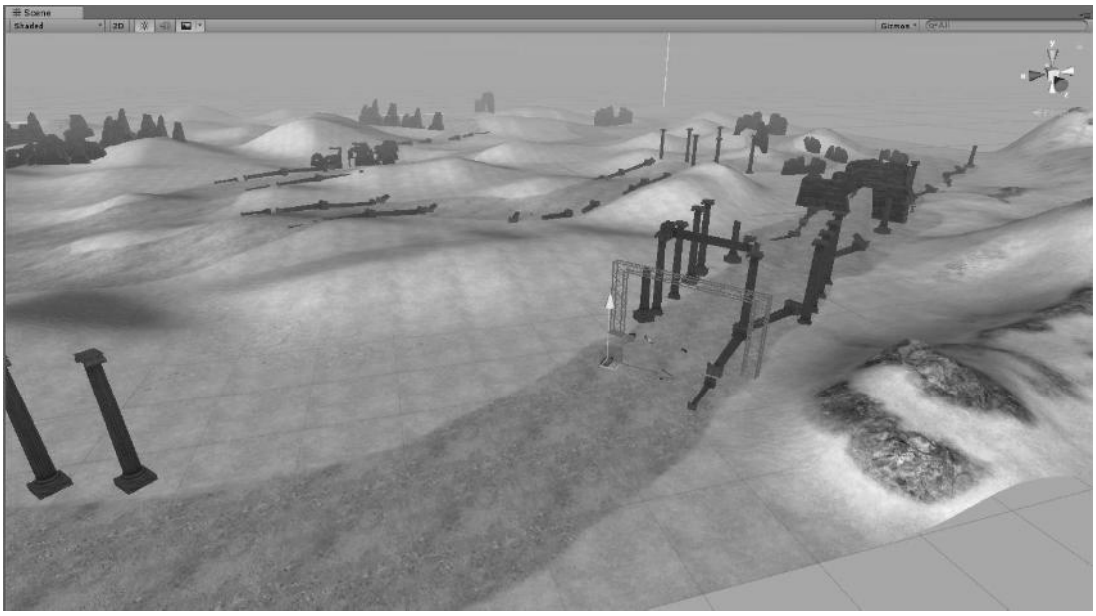
Iako izgled urednika korisnik može lako modificirati i proširiti dodacima, alatna traka (engl. *Toolbar*) je jedini dio sučelja Unityja koji se ne može preraspodijeliti. Alatna traka sadrži skup važnih alata koji se odnose na različite dijelove urednika.

Alatna traka (Slika 8.) sastoji se od pet osnovnih kontrola, svaka povezana s drugim dijelom uređivača (engl. Editor). Alati za transformaciju (1 i 2) služe za manipuliranje prikazom scene, Play, Pause i Step tipke (3) služe za prikaz igre, tj. animirane scene, padajući izbornik Layers (4) služi za odabir objekta koji se prikazuje u prikazu scene dok padajući izbornik Layouts (5) služi za kontrolu svih prikaza.

Slika 8. Alatna traka (izvor: snimka autorova zaslona)

4.2.2. Prikaz scene

Jedan od najvažnijih pogleda u uredniku je prikaz scene (engl. *Scene view*) – vizualni prikaz svijeta igara ili razine (Slika 9.). Omogućuje manevriranje, manipuliranje i pozicioniranje svih objekata i sredstava navedenih u Hijerarhiji, čime se stvara fizički prostor koji igrači mogu istraživati i komunicirati [12].



Slika 9. Prikaz scene (izvor: snimka autorova zaslona)

Kada se objekt klikne u prikazu scene, on se odabire i inspektor će biti ažuriran s odgovarajućim podacima objekta. Koristeći navedene alate koji se nalaze na alatnoj traci, korisnik sada može ručno promijeniti položaj, rotaciju i mjerilo objekta. Te se manipulacije nazivaju transformacijama. Odgovarajuće vrijednosti mogu se mijenjati i putem inspektora.

4.2.3. Prikaz igre

Prikaz igre (engl. *Game view*) uglavnom se koristi za pregled i testiranje, prikaz igre prikazan na Slici 10. prikazuje igru točno onako kako bi bilo u izgrađenoj verziji. Pritiskom gumba Reproduciraj (engl. *Play*) na alatnoj traci urednik aktivira taj prikaz i pokreće igru. Sve promjene koje su napravljene za to vrijeme neće biti spremljene, međutim, izmjena vrijednosti tijekom igre može pomoći tijekom testiranja.

Poput prikaza scene, pogled igre također ima vlastitu kontrolnu traku. Prvo područje, padajući izbornik Aspect, mijenja omjer prikaza igre, čak i dok se trenutno reproducira. Klikom na gumb za prebacivanje na maksimiziranje na reprodukciji proširit će se prikaz Igra da zauzme cijeli prikaz uređivača tijekom reprodukcije. Pritiskom na Gizmos dugme omogućuje korisniku da odabere koji će gizmosi biti prikazani u prikazu igre. Na kraju, gumb Stats na kontrolnoj traci će prikazati stranicu Render Statistics koja prikazuje korisne informacije o optimizaciji igre, kao što su FPS[5] [12].

[5] Sličice po sekundi (engl. FPS – frames per second) je frekvencija u kojoj uređaj izrađuje jedinstvenu sliku te s više slika može nastati animacija.

Slika 10. Prikaz igre (izvor: snimka autorova zaslona)

4.2.4. Hijerarhija

Hijerarhijski pogled (engl. *Hierarchy*), prikazan na Slici 11., sadrži svaki objekt u trenutnoj sceni, ažurirajući svaku promjenu kako korisnik dodaje ili uklanja objekte. Neki od njih mogu biti izravni primjerci datoteka imovine kao što su 3D modeli, a drugi mogu biti stavke Prefabova. Svaka instanca objekta bit će navedena pojedinačno, što čini dobru konvenciju imenovanja posebno važnom.

Slika 11. Hijerarhija (izvor: snimka autorova zaslona)

Odabirom objekta u Hijerarhiji i brisanjem uklonit će se objekt s trenutne scene u igri, ali ne i iz mape Projekti.

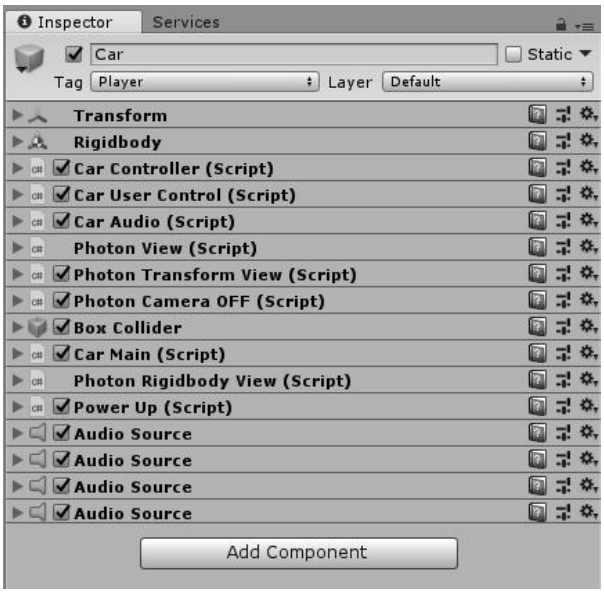
4.2.5. Preglednik projekta

Sve datoteke trenutačnog projekta organizirane su i uređene u pregledniku Projekta (engl. *Project browser*). Lijeva ploča preglednika, kao što se vidi na Slici 12., prikazuje sve u hijerarhijskoj strukturi kao što je mapa, točno kako se mape i njihovi sadržaji mogu naći na tvrdom disku. Pomoću malog trokuta proširuje se ili sažima mapa i tako se uklanjaju sve ugniježđene mape koje sadrži. Kada je mapa odabrana s popisa klikom, njezin će se sadržaj prikazati na ploči s desne strane kao ikone [13].

Slika 12. Preglednik projekta (izvor: snimka autorova zaslona)

4.2.6. Inspektor

Kroz pogled inspektora (engl. *Inspector*), sve komponente i vrijednosti prisutne u trenutno odabranom Objektu (engl. *GameObject*) ili Assetu mogu se promatrati i uređivati. Dodavanjem ili ponovnim pomicanjem komponenti i uređivanjem njihovih vrijednosti, inspektor se koristi za izmjenu funkcije Objekata. Ako je istodobno odabrano nekoliko Objekata sa zajedničkim komponentama, odgovarajuće vrijednosti mogu biti višestruko uređene istodobno. Komponente pridodane igraču mogu se vidjeti na Slici 13.



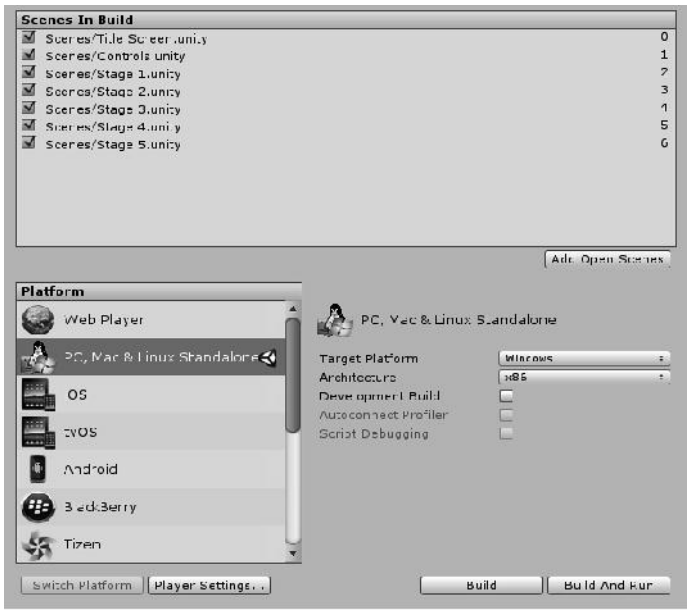
Slika 13. Komponente pridodane igraču (izvor: snimka autorova zaslona)

Kada Objekti (engl. GameObject) u igri imaju skripte priključene na njih, javne varijable u toj skripti također se prikazuju u inspektoru i mogu se uređivati točno kao svojstva ugrađenih komponenti Unityja. Komponente se dodaju Objektima jednostavnim pritiskom na gumb Dodaj komponentu (engl. *Add Component*) u inspektoru. Objekt se može deaktivirati tako da se uključi potvrdni okvir s lijeve strane imena Objekta. Ispod imena Objekta, padajući izbornici za Oznake (engl. Tag) i Layer mogu se pronaći, objašnjeno u poglavlju 4.3.

Na desnoj strani komponente prisutne u Objektu, postoji upitnik koji vodi do stranice Unity reference te određene komponente. Uređaj uz upitnik može se koristiti za resetiranje svih vrijednosti komponente, kopiranje, potpuno uklanjanje komponente iz objekta ili za pomicanje gore i dolje u prikazu Inspektor.

4.2.7. Postavke izgradnje

Kada je potrebno testiranje izvan urednika ili kada je igra spremna za objavljivanje, prozoru postavki za izgradnju može se pristupiti tako da se odaberu Postavke izgradnje (engl. Build Settings) iz izbornika Datoteka. Opcije u ovom prozoru omogućuju odabir ciljne platforme i različitih postavki prilagodbe za proces izgradnje i krajnjeg proizvoda. Sljedeća slika prikazuje prozor postavke izgradnje s dodanim scenama.



Slika 14. Postavke izgradnje (izvor: snimka autorova zaslona)

Dio prozora Scenes in Build prikazuje dodane scene iz trenutnog projekta koji će biti uključen u gradnju. Scene se mogu dodati pritiskom na gumb Dodaj otvorene (engl. Add Open Scenes) scene ili povlačenjem scene u ovaj prozor. Grafičke postavke kvalitete za gradnju mogu se pronaći u Edit> Project Settings> Quality. Izbor između memoriranih razina kvalitete koje je osmislio programer daje se igraču prilikom pokretanja igre. Nakon što se navedu postavke izgradnje, klikom na Build ili Build And Run stvorit će se projekt koji se može izvesti na navedenoj platformi [14].

4.2.8. Trgovina elementima

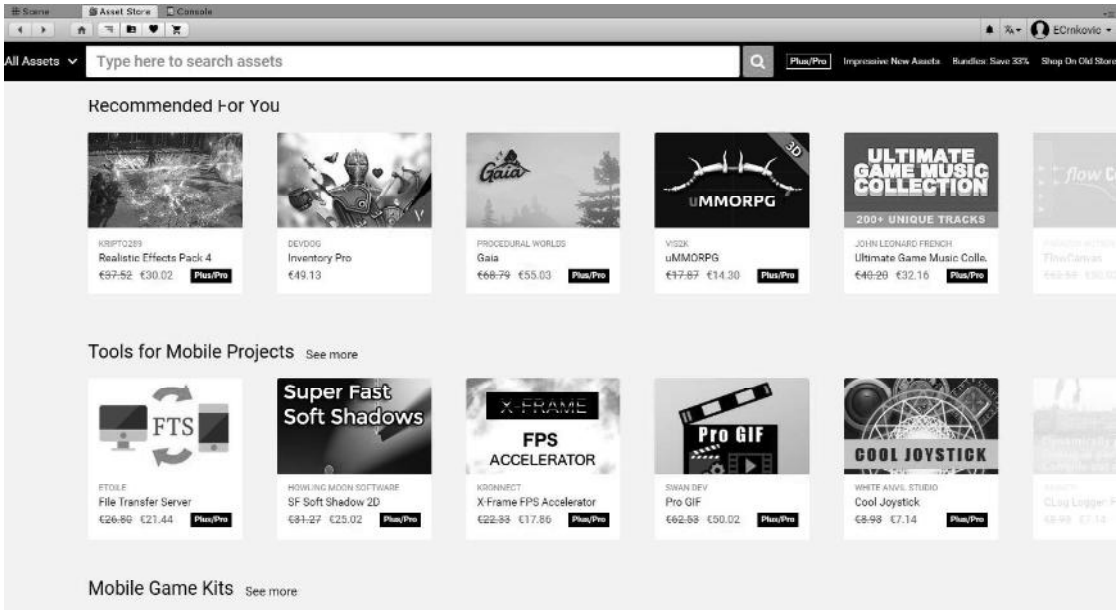
Prvobitno pokrenut u studenom 2010., Unity trgovina (engl. Unity Asset store) djeluje kao *online* market za Unity korisnike i ostale programere ili umjetnike kako bi međusobno prodali imovinu projekta. S više od 40.000 paketa dostupnih u raznim kategorijama i cijenama u rasponu od besplatnog do preko $ 1000, Unity trgovina nudi potencijalnu šansu za očuvanje resursa tijekom razvoja igara. Izdavač imovine dobiva 70 % smanjenja svoje zadane cijene u trgovini, dok Unity uzima 30 % od svake prodaje [10].

Trgovini se može pristupiti putem jednostavnog sučelja ugrađenog u Unity Editoru ili *web*-preglednika.

Imovina (engl. Asset) je prikaz stavki koje programer može koristiti u igrama ili projektima. Objekt može potjecati iz datoteke stvorene izvan jedinice Unity, kao što je 3D model, audiodatoteka, slika ili bilo koja druga vrsta datoteka koje Unity podržava. Postoje i neke

vrste imovine koje se mogu kreirati unutar Unityja, kao što su Animator, Audio Mixer ili Render Texture. Sredstva kreirana izvan jedinice Unity moraju biti uvezena spremanjem ili kopiranjem izravno u aktualnu mapu imovine projekta. Unity će automatski otkriti datoteke kada se dodaju ili izmijene.

Trgovini se može pristupiti putem *web*-preglednika ili jednostavnog sučelja ugrađenog u Unity uređivaču koji je prikazan na Slici 15., koja 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 15. Trgovina elementima (izvor: snimka autorova zaslona)

Paketi korišteni u kreiranju igre su sljedeći:

- Standard Assets, dostupno 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.
- Mountain Race Track – Night, dostupno na linku https://assetstore.unity.com/packages/3d/environments/roadways/mountain-race-track-night-68199
- Space Pirate Weapons, dostupno na linku https://assetstore.unity.com/packages/3d/props/weapons/space-pirate-weapons-11189

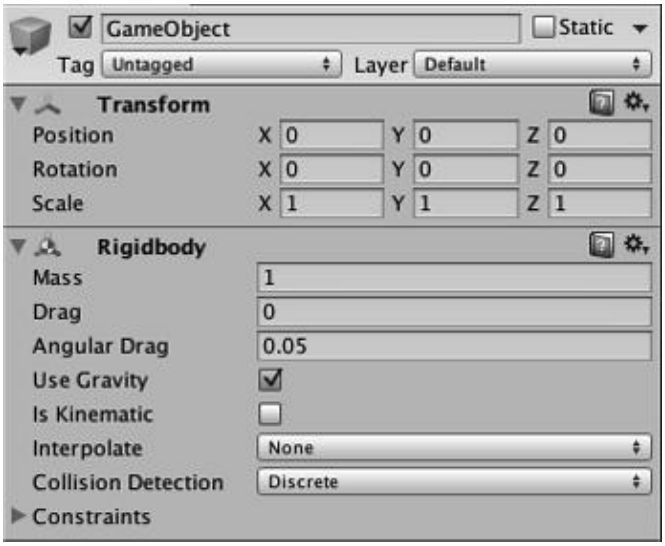
- Ancient Ruins in the desert, dostupno na linku https://assetstore.unity.com/packages/3d/environments/ancient-ruins-in-the-desert-part2-19178
- Retro Cartoon Cars - Cicada, dostupno na linku https://assetstore.unity.com/packages/3d/vehicles/land/retro-cartoon-cars-cicada-96158

4.3. Glavna načela djelovanja Unityja

Unity koristi komponentni model u stvaranju svjetova igara i njihovih funkcionalnosti. U praksi, komponente su funkcionalni dijelovi Objekta, koji pak predstavljaju scene, rekvizite, scenografiju itd. u scenama.

4.3.1. Komponente

Komponenta se može smatrati manjim dijelom većeg stroja. Oni su funkcionalni dijelovi svakog Objekta (engl. GameObject) i kako bi se izvršile navedene funkcionalnosti, uvijek moraju biti priključeni na objekte. Komponente svoje funkcionalnosti ostvaruju kombinacijom metoda i varijabli. Dodavanjem, brisanjem i uređivanjem komponente i njezinim vrijednostima upravlja se putem pogleda inspektora. Na Slici 16., komponenta Rigidbody je priključena na objekt.



Slika 16. Objekt s Rigidbody komponentom (izvor: snimka autorova zaslona)

Osim ugrađenih komponenti, kao što je fizika, prilagođene komponente mogu se kreirati pisanjem skripti. Kada je skripta priključena na Objekt, pojavljuje se u inspektoru kao komponenta. To je zato što se skripta kompajlira kao vrsta komponenata i kao takve ih tretira Unity [15].

4.3.2. Objekt

Objekt (engl. GameObjects) su temeljni dijelovi u Unityju. Iako sami od sebe ne postižu mnogo, djeluju kao spremnici za komponente koje implementiraju stvarne funkcionalnosti [8]. Prema zadanim postavkama, svi objekti sadrže samo jednu komponentu Transform, koja definira svoj položaj, orijentaciju i mjerilo u prostoru. Kada se dodaju različite vrste kombinacija komponenti, one se mogu sastaviti u svjetla, drveće, zvukove, upravitelje logičkih igara ili sve što je korisnik spreman izgraditi. Komponente priključene na potpuno različite objekte mogu komunicirati međusobno.
Za svaki objekt može se postaviti ime, oznaka i sloj prema potrebama korisnika. Pohranjeni objekti spremaju se kao Prefabs [16].

4.3.3. Prefab

Unity ima Prefab tip imovine koji omogućuje pohranjivanje objekata zajedno s komponentama i svojstvima. Sve modifikacije prefaba odmah se odražavaju u svim instancama (kopijama) proizvedenim iz njega, međutim, nadjačavanje komponenti i postavki za svaki pojedini slučaj je također moguće ako je potrebno. Prefabovi su posebno korisni pri stvaranju objekata koji se ponavljaju poput metaka, neprijatelja i kolekcionarskih predmeta [17].

Prefab djeluje kao predložak iz kojeg se mogu stvarati nove instance objekta u sceni. Objekti stvoreni kao montažne instance prikazat će se u prikazu hijerarhije u plavom tekstu. Kao i drugi objekti, prefabovi se također uređuju u pogledu inspektora.

4.3.4. Oznake

Oznake (engl. *Tag*) su riječi ili nizovi koji se mogu koristiti za objekte koji se koriste za njihovu identifikaciju u svrhe skriptiranja. Oznaka se može definirati za stavke koje igrač može prikupiti na sceni. U skriptama su korisne ako se isti kod skripte koristi u nekoliko objekata, na primjer kada se koriste TriggerColliders za otkrivanje sudara igrača i metka [18].

U nastavku je prikazan kod sudara metka i neprijatelja pomoću funkcije OnTriggerEnter(Collider other)

```
public void OnTriggerEnter(Collider other)  {
        if (other.tag == "Bullet") {
            Destroy(gameObject);
        }}
```

Programski kod 1. Sudar igrača i metka (Izvor: Autor)

Oznake se primjenjuju na objektima u prikazu Inspektor. Iz istog padajućeg izbornika dodaju se nove oznake koje je stvorio korisnik.

4.3.5. Scene

Svaka scena, sa svim objektima u njoj, može se smatrati jedinstvenom razinom. Scene sadrže sve informacije o tome što će se dogoditi kada se igra. U svakoj sceni okruženja, dekoracije, likovi i mehanika igranja definiraju razinu igre.

Po zadanim postavkama, svi objekti se unište pri prebacivanju između scena, međutim, oni mogu biti postavljeni da budu neuništeni, pomoću DontDestroyOnLoad() metode.

4.4. Skripte

Ponašanje objekata kontroliraju komponente koje su pridodane njima. Iako Unity ugrađene komponente mogu biti vrlo svestrane, potrebne su vlastite komponente za implementaciju vlastitih značajki igranja. Unity omogućuje stvaranje vlastitih komponenti pomoću skripti. To omogućuje da se pokrene događaj u igri, promijene svojstva komponente tijekom vremena i odgovori na korisničke unose na bilo koji način.

Dok se može koristiti klasično objektno orijentirano programiranje (OOP), tijek rada Unity se nadograđuje na strukturu komponenti i skriptiranje prilagođene komponente. Tanka linija između programiranja i skriptiranja postala je još više nejasna tijekom godina. U osnovi, skriptiranje ili pisanje skripti je programiranje unutar programa dok programiranje piše softver koji radi neovisno o vanjskom programu [14]. Unity podržava izvorni C # programski jezik. C # je industrijski standard sličan Java ili C ++.

MonoBehaviour je osnovna klasa iz koje svaka skripta potječe. Kada je MonoBehaviour naslijeđen u klasi, on može koristiti sve osnovne značajke i funkcije Unity Enginea. Najčešće

korištene i najvažnije funkcije su Awake (), Start (), Update (), FixedUpdate (), OnGUI () i funkcije koje se tiču sudara (engl. *collisions*).

Redoslijed izvršavanja funkcija tijekom trajanja skripte je sljedeći [19]:

- Editor
  - Reset
- Initialization
  - Awake
  - OnEnable
  - Start
- Physics
  - FixedUpdate
  - yield waitForFixedUpdate
  - OnTrigger
  - OnCollision
- Input Events
- Game logic
  - Update
  - yield waitForSeconds
  - StartCoroutine
  - Animation update
  - LateUpdate
- Scene rendering
- Gizmo rendering
- GUI rendering
  - OnGUI
- End of frame
  - yield waitForEndOfFrame
- Pausing
  - OnApplicationPause
- Disable
  - OnDisable
- Decommissioning

- OnApplicationQuit
- OnDisable
- OnDestroy

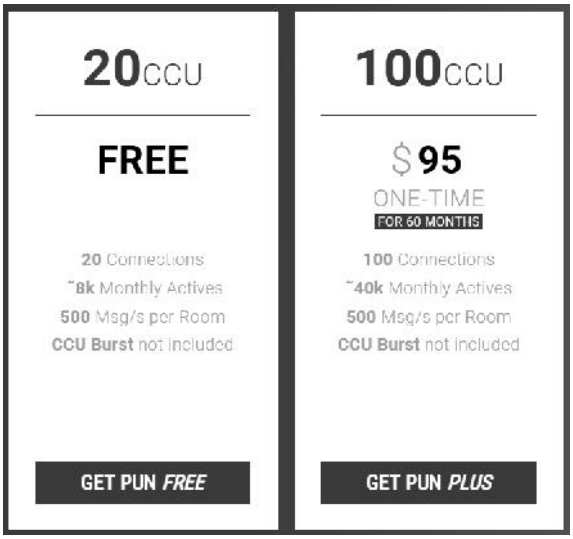
5. PHOTON UNITY NETWORKING

5.1. Pregled značajki

Photon Unity Networking, ili PUN, je Unity plugin paket za stvaranje igara za više igrača. On pruža mogućnosti provjere autentičnosti, podudaranje (engl. matchmaking) i komunikaciju u igri putem Photona. PUN multiplayer značajke temelje se na kreiranju soba, a igre su smještene u globalno distribuiranim Photon oblacima, kako bi se zajamčila niska latencija za igrače širom svijeta.

Igre koje ovise o niskoj latenciji, poput vrsta FPS ili RTS igara, povezuju igrače s najbližom regijom. Igre koje su u stanju nositi se s većom latencijom, kao što su vrste igara koje se temelje na potezu, mogu povezati sve igrače s istom regijom.

PUN izvozi gotovo sve platforme koje podržava Unity, a mogu se birati između dva različita paketa: PUN Free i PUN Plus. Slika 17. prikazuje usporedbu između ova dva paketa. [20]



Slika 17. Pun i Pun plus [14]

PUN također ima vlastite mrežne komponente, povratne funkcije, pozive s daljinskim postupcima i klase. Najvažnije klase su:

1. PhotonNetwork, koja je glavna klasa za korištenje PhotonNetwork plugina.

2. Photon.Monobehaviour, nasljeđuje MonoBehaviour i dodaje PhotonView property.

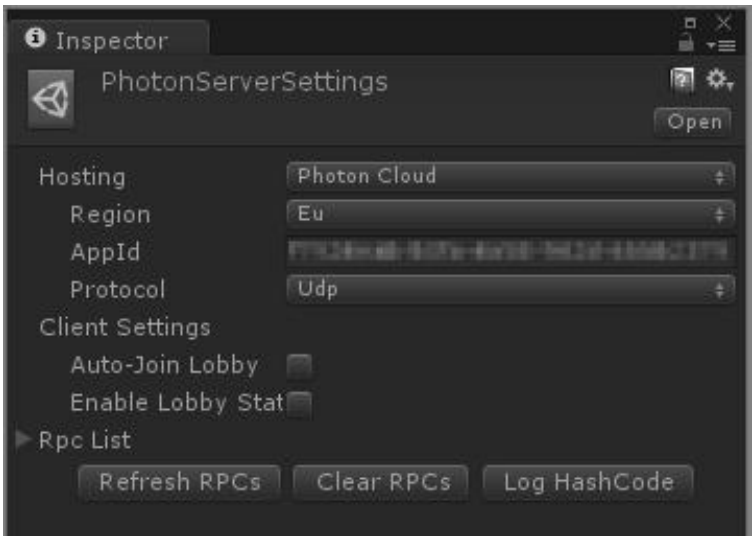
3. Photon.PunBehaviour pruža PhotonView i sve povratne pozive koje PUN može pozvati.

5.2. Početno postavljanje

Nakon što se PUN doda u Unity projekt, pojavit će se PUN Setup Wizard. Kreiranje novog Photon Cloud računa osigurava osobni identifikacijski broj za korisnika, koji je potreban za Photon Cloud hosting. Slika 18. prikazuje PUN Wizard.

Slika 18. PUN Wizard (izvor: snimka autorova zaslona)

PUN Wizard također dodaje datoteku PhotonServerSettings u mapu imovine projekta (*asset folder*), koja se može koristiti za lako mijenjanje konfiguracija poslužitelja. Kroz konfiguracijsku datoteku korisnik može uređivati vrste usluge poslužitelja (engl. *Hosting*), regije poslužitelja, protokole i postavke klijenta. Slika 19. prikazuje datoteku PhotonServerSettings i opcije koje sadrži.



Slika 19. PhotonServerSettings (izvor: snimka autorova zaslona)

Na padajućem izborniku Hosting korisnik može odabrati koji će poslužitelj obraditi umrežavanje igre. Obje opcije Photon Cloud i Best Region odnose se na usluge kojima upravlja Photon. Najbolja regija (engl. Best Region) će pingirati sve navedene regije kada se aplikacija prvi put pokrene i omogućuje klijentima da odaberu regiju s najboljim pingom[6] ako je specificirano više od jedne regije. Po prvobitnim postavkama je protokol veze postavljen kao UDP[7], međutim, Photon također podržava TCP[8].

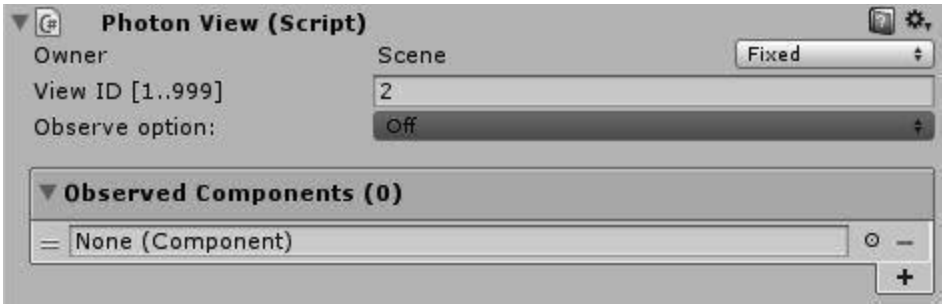
Odjeljak Client Settings (Postavke klijenta) sadrži opcije za Auto-Join Lobby i Enable Lobby Stats. Ako je Auto-Join Lobby označen, PUN će se automatski pridružiti igračima u zadanom predvorju kada je veza uspostavljena ili kada napušta sobu. Statistika lobiranja omogućuje primanje statističkih podataka s poslužitelja, što se može pokazati korisnim kada se radi o igri koja koristi višestruke lobije ili kada se klijentima trebaju prikazati aktivnosti poslužitelja, kao što je broj igrača.

5.3. Bitne komponente

PhotonView koristi se za slanje poruka preko mreže. PUN zahtijeva jedan PhotonView po instanci ranom prefabu kako bi pratio mrežne reference, vlasništvo objekata i promatrane referencije komponenti. PUN prati PhotonViews koje je instancirala lokalno i referentne

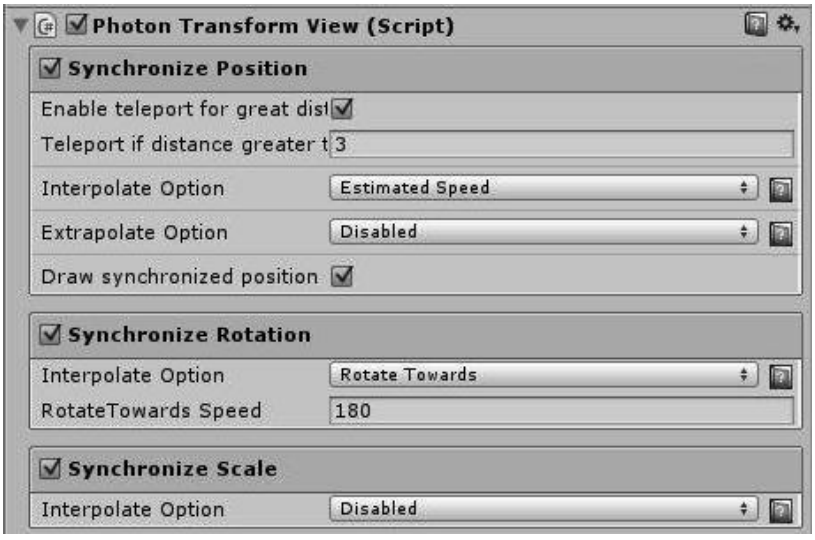
[6] Ping je mjera latencije (kašnjenja) od računala do poslužitelja i natrag na računalo.
[7] UDP (User Datagram Protocol) je protokol koji se nalazi u dijelu transportne razine OSI modela. UDP omogućuje slanje kratkih poruka (*datagram*) između aplikacija na umreženim računalima.
[8] TCP (*Transmission Control Protocol*) Korištenjem protokola TCP aplikacija na nekom poslužitelju umreženom u računalnu mrežu kreira virtualnu vezu prema drugom poslužitelju te putem te ostvarene veze zatim prenosi podatke.

promatrane komponente mogu slati ažuriranja drugim klijentima tijekom izvođenja. Na primjer, ako je Transformna komponenta postavljena kao „promatrana“, njezina pozicija, rotacija i vrijednosti skale sada su sinkronizirane preko mreže s drugim igračima. Slika 20. prikazuje komponentu PhotonView. [21]



Slika 20. PhotonView komponenta (izvor: snimka autorova zaslona)

Komponente PhotonTransformView, PhotonRigidbodyView i PhotonRigidbodyView2D nude izbor prilagodljivih opcija za naprednu sinkronizaciju. Komponente RigidbodyView mogu se koristiti za sinkronizaciju brzina prema Rigidbody fizici, dok PhotonTransformView otvara više opcija o tome kako se podaci transformiraju sinkronizirano. Sve tri komponente trebaju biti dodane u „promatrano“ polje PhotonViewa, ako su priključene na Objekt. Slika 21. prikazuje opcije u komponenti PhotonTransformView.



Slika 21. PhotonTransformView komponenta (izvor: snimka autorova zaslona)

5.4. Matchmaking

Klasa PhotonNetwork uvijek koristi glavni poslužitelj i jedan ili više poslužitelja igara (engl. Game servers). Glavni poslužitelj upravlja trenutno dostupnim igrama i ne spaja se, dok poslužitelji igara obrađuju stvarnu igru nakon što je soba pronađena ili stvorena. PhotonNetwork.ConnectUsingSettings je potreban korisniku kako bi se iskoristile značajke Photona. Ona postavlja verziju igre klijenta i koristi Pho-tonServerSettings za povezivanje. U PUN-u Glavni klijent je uvijek igrač s najnižim ID-om u sobi. Svi klijenti mogu provjeriti jesu li trenutno glavni s PhotonNetwork.isMasterClient.

U osnovi postoje tri pristupa za ulazak u sobu za igru s nekim [21]:

1. poslužitelj pronalazi odgovarajuću sobu
2. pristup sobi putem imena sobe
3. dohvat popis soba te korisnik sam odabire jednu.

Sljedeći kod prikazuje osnovne funkcije za stvaranje, spajanje i popis soba:

```
//Spoji se u sobu
PhotonNetwork.JoinRoom(roomName);
//Stvori sobu
PhotonNetwork.CreateRoom(roomName);
//Spoji se nasumično u postojeću sobu
PhotonNetwork.JoinRandomRoom();
 //Vrati listu svih postojećih igri
PhotonNetwork.GetRoomList();
//Kreira ili spoji u postojeću sobu sa custom opcijama
//Soba ce se stvorit jedino ako soba s istim imenom vec nepostoji
RoomOptions roomOptions = new RoomOptions(){IsVisible = true, maxPla
yers= 4};
PhotonNetwork.JoinOrCreateRoom (roomName, roomOptions, TypedLobby.De
fault);
```

Programski kod 2. Stvaranje, spajanje i popis soba (Izvor: Autor)

5.5. RPC

U Photonu, PhotonView komponente su kao „mete“ za pozive s daljinskim procedurama (RPC). Kada je funkcija označena s [PunRPC], izvršava se na klijentima samo na umreženom Objektu s određenim PhotonView. Na primjer, ako klijent ošteti neki drugi objekt u igri i pozove RPC funkciju ApplyDamage, svi klijenti koji primaju primijenit će štetu na isti objekt na njihovu kraju. Kako bi pozvali funkcije označene kao RPC, potreban je PhotonView na

GameObjectu, kao i Photon.MonoBehaviour u skriptama. Primjer koda poruke koja se šalje putem RPC funkcije je sljedeća:

```
PhotonView photonView = PhotonView.Get(this);
photonView.RPC("ChatMessage", PhotonTargets.All, playerName, message
);
[PunRPC]
void ChatMessage(string a, string b){
    Debug.Log("Poruka " + a + " " + b));
}
```

Programski kod 3. Slanje poruka pomoću udaljenih procesnih poziva (RPC) (Izvor: Autor)

Poslužitelj pamti te RPC funkcije i kada se pridruži novi igrač, on prima RPC, iako se to dogodilo prije pridruživanja.

5.6. Instantiation

U Photonu, prefabs se stvara s PhotonNetwork.Instante funkcijom. PUN automatski registrira i brine se o mriješćenju tih umreženih objekata prolaskom početne pozicije, rotacije i prefab imena u funkciju *instantiate*. Svi umreženi prefabovi moraju sadržavati PhotonView komponentu i trebao bi se nalaziti izravno pod resursima – mapom za pristupanje tijekom izvođenja. Sljedeći kod prikazuje stvaranje objekta koristeći PhotonNetwork.Instante:

```
GameObject objekt=PhotonNetwork.Instantiate (mainPlayer.name,
pozicija.transform.position, pozicija.transform.rotation, 0);
```

Programski kod 4. Stvaranje objekta (Izvor: Autor)

Ako se programer ne želi osloniti na mapu resursa za instanciranje objekata, također je opcija „ručno“ izvođenje. Instanciranjem objektima putem udaljenih procesnih poziva (engl. remote procedure calls RPC) ostvarit će se taj zadatak. PhotonView.viewID je ključ za usmjeravanje mrežnih poruka u ispravne Objekte i skripte, dok PhotonNetwork.AllocateViewID () dodjeljuje nove viedID-ove, tako da svatko u sobi ima isti ID na novom objektu. Ručno izvođenje izvodi se na sljedeći način:

```
void SpawnPlayerEverywhere(){
int id1 = PhotonNetwork.AllocateViewID();
PhotonView photonView = this.GetComponent<PhotonView>();
photonView.RPC("SpawnOnNetwork", PhotonTargets.AllBuffered,    trnas
form.position, transform.rotation, id1,PhotonNetwork.player);
}
```

```
[RPC]
void SpawnOnNetwork (Vector3 pos, Quaternion rot,internal id1,Photon
Player np){
Transform newPlayer = Instantiate(playerPrefab,pos,rot) as Transform
;
PhotonView[] nViews = newPlayer.GetComponentsInChildren<PhotonView>(
);
    nViews[0].viewID = id1;}
```

Programski kod 5. Instanciranje pomoću udaljenih procesnih poziva (RPC) (Izvor: Autor)

5.7. Prava igrača

Postoji nekoliko načina za ovlaštenje nad kontrolama igrača u programu PUN. Prva je da kad se stvori objekt pomoću Photon-Network.Instantiate metode uključimo *bool* varijablu koja je istinita u slučaju da je korisnik stvorio avatara. Ovo je prikazano u donjem kodu, gdje se objekt igrača dobiva instanciranjem i njegova vrijednost isControllable je postavljena kao istinita za registriranje ulaza.

```
void OnJoinedRoom(){
GameObject player = PhotonNetwork.Instantiate ("playerprefab", Vecto
r3.zero, Quaterni                    on.identity, 0);
player.GetComponent<ThirdPearsonController> ().isControllable = true
;
}
if(isControllable){
        Input.GetAxisRaw("Vertical");
        Input.GetAxisRaw("Horizontal");
}
```

Programski kod 6. Prava igrača pomoću bool varijable. (Izvor: Autor)

Drugi način za postizanje slične funkcionalnosti i koja je korištena u ovom radu je upotreba isMine svojstva komponente PhotonView, koja vraća istinu ako klijent posjeduje dotični PhotonView.

```
if(photonView.isMine){
    Input.GetAxis("Vertical");
    Input.GetAxis("Horizontal");
}
```

Programski kod 7. Prava igrača pomoću svojstva komponente (Izvor: Autor)

5.8. Sinkronizacija stanja

Kao što je spomenuto u poglavlju 5.3., objekt se lako može pretvoriti u mrežnu svijest dodjeljivanjem PhotonView komponente. PhotonView mora biti postavljen tako da opslužuje druge komponente, kao što je Transform, ili češće druge prilagođene komponente skripte. Dok se skripta promatra, PhotonView komponenta redovito poziva metodu OnPhotonSerializeView. Dužnost ove funkcije je stvoriti informacije koje klijenti žele prenijeti drugima i upravljati takvim dolaznim informacijama, ovisno o tome tko je stvorio PhotonView. PhotonStream se prenosi na OnPhotonSerializeView i vrijednost isWriting govori klijentu treba li pisati ili čitati udaljene podatke s njega. Sljedeći isječak koda prikazuje ovu funkcionalnost dok šalje i prima pozicijske i rotacijske podatke.

```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessage
Info info){
    if (stream.isWriting){
        // Pošalji podatke
        stream.SendNext(transform.position);
        stream.SendNext(transform.rotation);
    } else {
        // Prima podatke
this.transform.position = (Vector3)   stream.ReceiveNext();
this.transform.rotation = (Quaternion) stream.ReceiveNext();
 }}
```

Programski kod 8. Korištenje funkcije OnPhotonSerializeView (Izvor: Autor)

Također je moguće postaviti prilagođena sinkronizacijska svojstva pomoću koda za objekte igrača i sobe pomoću SetCustomProperties () [21].

6. IZRADA IGRE

Speed Battle je jednostavna višekorisnička igra koja podržava istovremeno igranje do 4 igrača. Igra je dizajnirana isključivo za igranje preko mreže. U igri igrač upravlja posebno dizajniranim vozilom koje ima mogućnost ispaljivanja metaka, ubrzanje i evidencije zdravlja. Kao protivnik će poslužiti drugi igrači koji također imaju mogućnost ispaljivanja metaka. Igra omogućuje izbor 3 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 neki predlošci su napravljeni kroz Unity.

6.1. Izrada elemenata igre

6.2. Kreiranje igrača

Glavni igrač u igri je automobil. Osim što auto mora biti u voznom stanju, zbog borbe mora imati i dodatne borbene funkcionalnosti. 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 korišten je Unity standardni predložak Sky Car. Radi se o potpuno funkcionalnom autu futurističkog izgleda.

Sky Car predložak sadrži 3D model CarBody koji predstavlja tijelo auta. CarBody model je izbrisan i na njegovo mjesto postavljen je Retro Cartoon Cars 3D model auta. Nakon toga je napravljen predložak za novi auto tako da se s drag-and-drop metodom dovuče auto sa scene u Prefab direktorij i predlošku promijeni naziv u Player1.

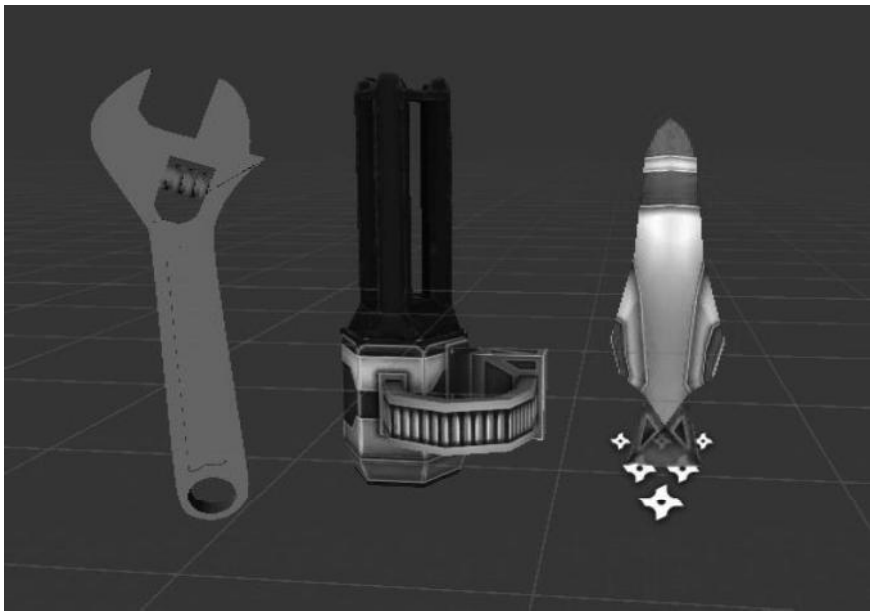
Pošto se radi o igri trke i borbe s autima, uvijek je dobro imati mogućnost izbora između više auta. Slika 22. prikazuje automobile koje je moguće izabrati.

Slika 22. Tipovi automobila (izvor: snimka autorova zaslona)

Svaki igrač bi trebao biti u mogućnosti odabrati vrstu automobila koji su željeli kontrolirati. Takav je odabir bio dodan u zaslon lobija, prije svakog kruga igre. Kada se odabere novi automobil, novi tip automobila šalje se na poslužitelj, čime se izbor automobila spremi odgovarajućem igraču. Kada igra započne, poslužitelj šalje ispravan tip automobila.

6.2.1. Predlošci za PowerUp

PowerUp predlošci su predlošci preko kojih će igrač stjecati razne prednosti tijekom igre. Predlošci koji su izrađeni su predložak koji će igraču dodavati zdravlje, metke i brzinu. Za kreiranje predložaka korišteni su besplatni 3D modeli skinuti s Unity trgovine. Predlošcima su dodijeljene oznake preko koje će se igračima omogućiti skupljanje PowerUp objekata tijekom igre. Slika 23. prikazuje 3d modele PowerUp predloška.

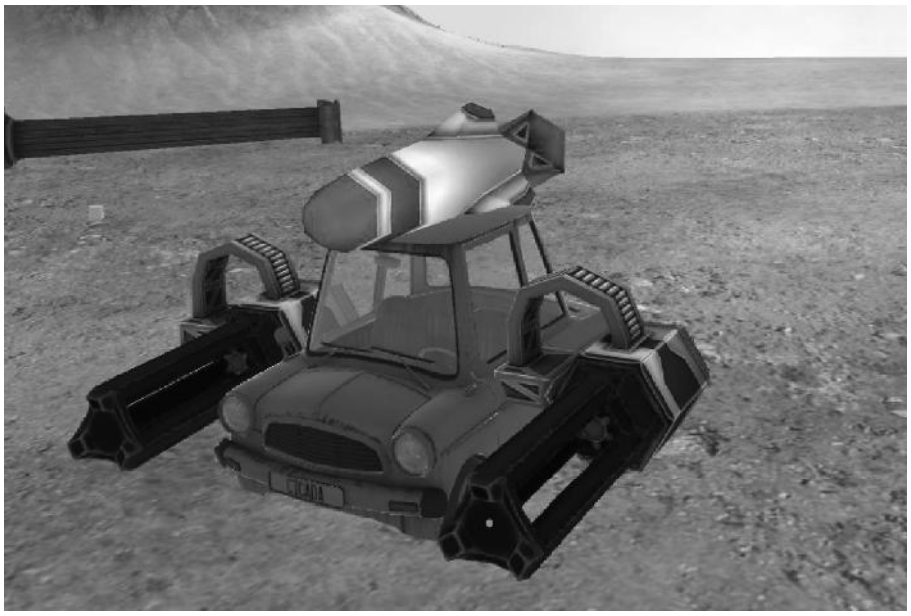


Slika 23. PowerUp predlošci (izvor: snimka autorova zaslona)

PowerUp.cs skripta je zadužena za interakciju predložaka s igračima. Detekcija kupljenja predloška realizirana je preko OnTriggerEnter metode koja prvo provjerava radi li se o interakciji sa Strojnicom, Hitnom ili Raketom predloška, igraču se zatim dodaje zdravlje, ubrzanje ili meci, ažurira se tekst koji prikazuje preostalo zdravlje, metke i ubrzanje te se aktivira 3D objekt Strojnice ili Rakete koji se vide na Slici 24. Igrač koji je pokupio PowerUp realiziran je naredbama:

```
if (other.tag == "Strojnica") {
            Strojnica.SetActive (true);
            Strojnica2.SetActive (true);
            Bullets =500;
        }
        if (other.tag == "Raketa") {
            Raketa.SetActive (true);
            raketa = 200;
        }
        if (other.tag == "Health") {
            Health = 100;
        }
```

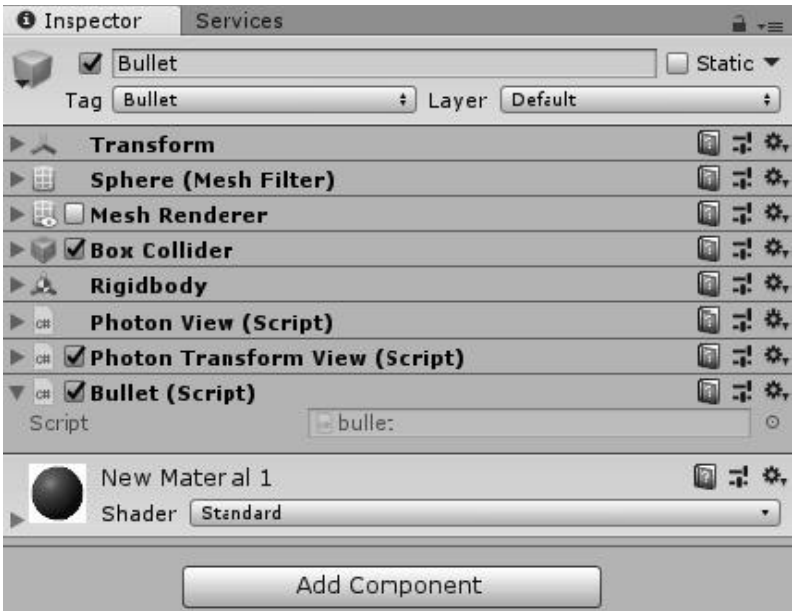
Programski kod 9. Skupljanje PowerUp (Izvor: Autor)



Slika 24. Model auta s naoružanjem (izvor: snimka autorova zaslona)

6.2.2. 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 kocke na koji je dodana komponenta ParticalSystem vatre radi boljeg efekta, komponentu Rigidbody te skriptu Bullet.cs koja je zadužena za detekciju pogotka. Važnije metode skripte su OnTriggerEnter () koja provjerava je li metak pogodio igrača ili ne. U slučaju pogotka igrača, pogođenom igraču se nanosi šteta. Pošto je metak predložak koji nije statičan u igri te mora biti sinkroniziran za sve igrače, potrebno je predlošku dodijeliti Photon Transform View i Photon View komponentu kao što vidimo na Slici 25.



Slika 25. Objekt s komponentama (izvor: snimka autorova zaslona)

Skripta PowerUp.cs je zadužena za pucanje, tj. za ispaljivanje metka. Da bi pucanje predloška bilo moguće, predlošku se preko PowerUp.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 photonView.isMine – 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. Sljedeća skripta pokazuje pucanje i stvaranje metka.

```
if (Input.GetMouseButton (0) && photonView.isMine) {

if (Bullets > 0) {
  pv.RPC ("Shoot", PhotonTargets.All);
} else {
          Strojnica.SetActive (false);
          Strojnica2.SetActive (false);
          }
   BulletsText.text = "" + Bullets;
   RaketaText.text = "" + raketa;
   }
```

```
[PunRPC]
void Shoot(){
Debug.Log("shoot");
Rigidbody instantiatedProjectile = Instantiate(projectile,StvaranjeS
trojnicaPozicija.positi          on,transform.rotation) as Rigidbody;
instantiatedProjectile.velocity =transform.forward * speed;
Rigidbody instantiatedProjectile2 = Instantiate(projectile,Stvaranje
StrojnicaPozicija2.position,transform.rotation) as Rigidbody;
instantiatedProjectile2.velocity =transform.forward * speed;
 Bullets--;
 }
```

Programski kod 10. Pucanje i stvaranje metka (Izvor: Autor)

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 [PunRPC].

Nakon svakog ispaljenog metka odrađuje se i ažuriranje vizualnog prikaza preostalih metaka preko BulletsText koji se nalazi na vrhu ekrana u scenama utrke. Ista stvar vrijedi i za prikaz preostalog zdravlja i ubrzanja pomoću rakete. Slika 26. prikazuje vizualni prikaz zdravlja, preostalih metaka i ubrzanja koji su vidljivi samo lokalnom igraču tako da nije potrebna sinkronizacija između klijenata.

Slika 26. Vizualni prikazi zdravlja, preostalih metaka i ubrzanja (izvor: snimka autorova zaslona)

6.3. Lobby Scene

Photon je uglavnom usmjeren na usluge oblak (*cloud*) servera koje pruža u stvaranju soba. Iako je moguće da igrači kreiraju vlastite poslužitelje i hostove lokalno, to je donekle nepraktično i suvišno. Slika 27. prikazuje glavni izbornik predvorja.

Slika 27. Glavni izbornik (izvor: snimka autorova zaslona)

Za početak, korisnici se automatski povezuju s *lobby* sustavom kada je otvoren glavni izbornik. To se postiže provjerom Auto-Join Lobby iz PhotonServerSettings datoteke i korištenjem sljedećeg koda u Awake (), u skripti PhotonLobby koja upravlja mrežnim funkcionalnostima lobija.

Prilikom pristupanja postoje dvije UI sekcije – prva prikazuje da se igra povezuje s poslužiteljem usluge, to jest Photonu i druga sa svim glavnim UI sekcijama kao što su dugmad za kreiranje soba, polja za upis imena, dugmad za odabir modela automobila i odabir staze. Sljedeća skripta pokazuje povezivanje na Photon poslužitelj te u slučaju neuspjelog povezivanja ili izgubljene veze aktivirat će se funkcija OnFailedToConnectedToPhoton() ili OnDisconnectedFromPhoton().

```
public string versionName = "0.1";
public GameObject section1, section2;
private void Awake(){
    PhotonNetwork.ConnectUsingSettings (versionName);
    Debug.Log ("conecting to photon...");}
private void OnConnectedToMaster(){
    PhotonNetwork.JoinLobby(TypedLobby.Default);
```

```
        Debug.Log("spojio si se na master");
    }
    private void OnJoinedLobby(){
        section1.SetActive (false);
        section2.SetActive (true);
        Debug.Log("Spojio si se u sobu");
    }
    private void OnDisconnectedFromPhoton(){
        Debug.Log ("Izgubio si vezu");
    }
    private void OnFailedToConnectedToPhoton(){
        Debug.Log ("Neuspjela konekcija");
    }
```

Programski kod 11. Povezivanje na poslužitelja (Izvor: Autor)

Nakon što je veza uspostavljena, korisnici sada mogu kreirati sobe s unosom specifičnog imena, postaviti ime igrača, odabrati auto te stazu koju žele igrati.

U Photonu, korisnička imena igrača mogu se lako pohraniti lokalno s PhotonNetwork.playerName. Kada se kliknula tipka za kreiranje sobe, ime igrača se sprema kako bi se osiguralo da su sve promjene u imenu ispravno pohranjene. Prvi odjeljak u sljedećem kodu dohvaća ime iz skripte PhottonButtons koje je korisnik upisao, a drugi odjeljak brine se o nasumičnom imenovanju u slučaju da korisnik nije upisao ime.

```
public PhotonButtons photonB;
PhotonNetwork.playerName = photonB.ime.text;
 if (string.IsNullOrEmpty (PhotonNetwork.playerName)){
    PhotonNetwork.playerName = "Gost" + Random.Range (1, 999);
}
```

Programski kod 12. Imenovanje igrača (Izvor: Autor)

Gumb Create Room (Stvori sobu) stvara sobu s navedenim nazivom i povezuje korisnika u nju. Svi gumbi u sustavu lobija izvršavaju funkcije putem OnClick() događaja. Sljedeći kod prikazuje stvaranje sobe pomoću funkcije CretRoom() koja u slučaju praznog imena daje upozorenje da je ime sobe potrebno:

```
public PhotonButtons photonB;
public void CreatRoom(){
 if (string.IsNullOrEmpty (photonB.creatRoomInput.text)) {
            Debug.Log ("upisi ime sobe");
```

```
            return;
        }
PhotonNetwork.CreateRoom (photonB.creatRoomInput.text, new RoomOptio
ns (){
MaxPlayers = 4 }, null);
```

Programski kod 13. Stvaranje sobe (Izvor: Autor)

Kada se klikne na gumb List Rooms, on traži popis trenutno aktivnih igara i stvara vizualni prikaz za svaku od njih putem funkcije PhotonNetwork.GetRoomList ().

Sljedeći kod prikazuje funkciju RoomList() koja prikazuje popis svih postojećih soba pomoću dugmeta koji kad se stvori ima tekst s imenom sobe, trenutni broj igrača i maksimalni broj igrača te mu se dodaje funkcija Gumb(string RoomName)pomoću koje se pristupa igri. Ako nema aktivnih soba, tekst No servers found postavljen je kao aktivan.

```
public void RoomList(){
Roomlist.SetActive (true);
MainScreen.SetActive (false);
createdRooms = PhotonNetwork.GetRoomList();
if(createdRooms.Length == 0 ){
Debug.Log ("Nema pronađenih poslužitelja");
NemaSobeText.GetComponent<Text> ().enabled = true;
} else {
for(int i = 0; i < createdRooms.Length; i++)  {
GameObject newButton = Instantiate(ButtonPrefab,Button
Position[i].position,transform.rotation) as GameObject;
newButton.transform.parent = Roomlist.transform;
newButton.GetComponentInChildren<Text>().text =  createdRooms[i].Nam
e + "   " +createdRooms[i].PlayerCount + "/" + createdRooms[i].MaxPl
ayers;
string name = createdRooms [i].Name;
Button b = newButton.GetComponent<Button>();
b.onClick.AddListener(() => Gumb(name));}}}
```

Programski kod 14. Popis slobodnih soba (Izvor: Autor)

Sljedeća slika prikazuje Room list sekciju s dugmetom za spajanje na aktivnu scenu pod nazivom PHOTONROOM1 u kojoj je trenutno jedan igrač od moguća 4.

Slika 28. Dugme za pridruživanje (izvor: snimka autorova zaslona)

Kada se klikne na gumb, povezuje se klijenta s tom posebnom sobom. Gumb zatim identificira ispravnu sobu za povezivanje klijenta pomoću imena sobe. Slično kao i kod stvaranja sobe, prefab igrača stvara se kada se pridruži sobi. Kod za pridruživanje sobi je sljedeći:

```
public void Gumb(string RoomName){
        Debug.Log ("button is pressed " + RoomName );
        pHandler.joinRoom (RoomName);     }
```

Programski kod 15. Pridruživanje sobi (Izvor: Autor)

Kada je soba stvorena ili kad se igrač spoji u postojeću sobu, stvori se prefab igrača u sceni pomoću PhotonNetwork.Instantiate() unutar funkcije spawnPlayer(). Prije stvaranja igrača želimo provjeriti na koju poziciji će se stvoriti ovisno o tome ima li već igrača na sceni ili ne te s kojom vrstom automobila. Sljedeći kod prikazuje stvaranje igrača:

```
public GameObject[] mainPlayer = new GameObject[4];

public void spawnPlayer(){
int i = photonB.CarChoice;
numberPlayers = PhotonNetwork.room.PlayerCount;
GameObject objekt;
GameObject pozicija = GameObject.FindGameObjectWithTag ("Player" + n
umberPlayers);
objekt = PhotonNetwork.Instantiate (mainPlayer[i].name, pozicija.tra
nsform.position, po     zicija.transform.rotation, 0);}
```

Programski kod 16. Stvaranje igrača (Izvor: Autor)

Nakon stvaranja kreirana je funkcija koja pokazuje igraču da je igra počela. Ranije je igrač skočio izravno iz zaslona lobija u igru. To je riješeno pomoću sljedeće skripte koja koristi RPC koji poziva funkciju „Kreni“ na svim igračima za početak igre, čim se spoje svi igrači. Funkcija sadržava broj „odbrojavanja“, počevši od 3. Za svako „odbrojavanje“ dan je vizualni prikaz.

```
private CarMain c_main;
void OnPhotonPlayerConnected(PhotonPlayer newPlayer){
            if (PhotonNetwork.room.PlayerCount == 4) {
                Invoke("Sync", 2);
           }
    public void Sync(){
             pv.RPC ("Go", PhotonTargets.All);
        }
        [PunRPC]
        void Go(){
            c_main.Kreni ();
        }
```

Programski kod 17. Poziv na početak igre svim igračima (Izvor: Autor)

CarMain.cs skripta:

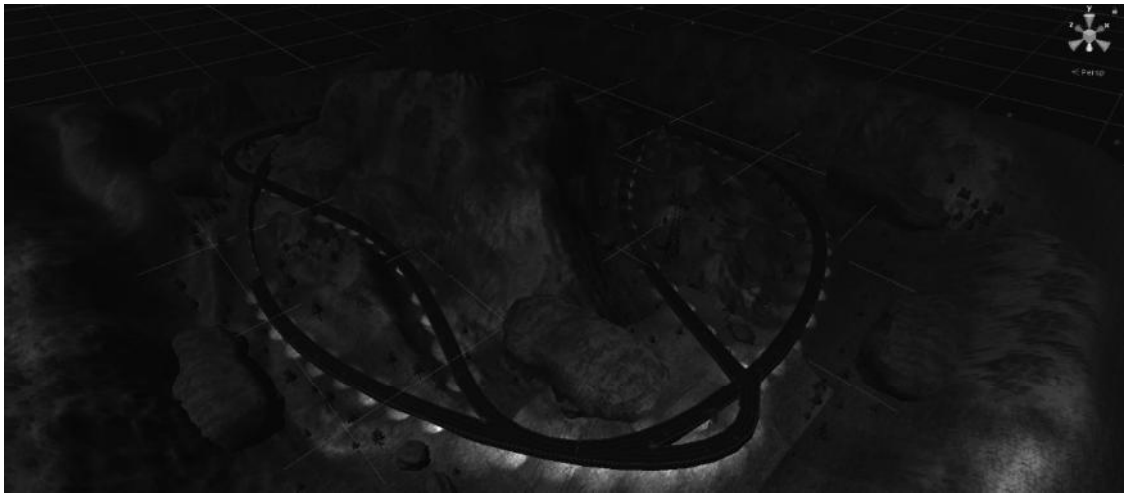
```
    public Image go1, go2, go3;
    public bool start ;
    public void Kreni(){
        if (start == false) {
            StartCoroutine (Go ());
        }}
 IEnumerator Go(){
        go1.gameObject.SetActive(true);
        yield return new WaitForSeconds (1.0f);
        go2.gameObject.SetActive(true);
        yield return new WaitForSeconds (1.0f);
        go3.gameObject.SetActive(true);
        yield return new WaitForSeconds (1.0f);
        go3.gameObject.SetActive(false);
        go2.gameObject.SetActive(false);
```

```
go1.gameObject.SetActive(false);
start = true;     }
```

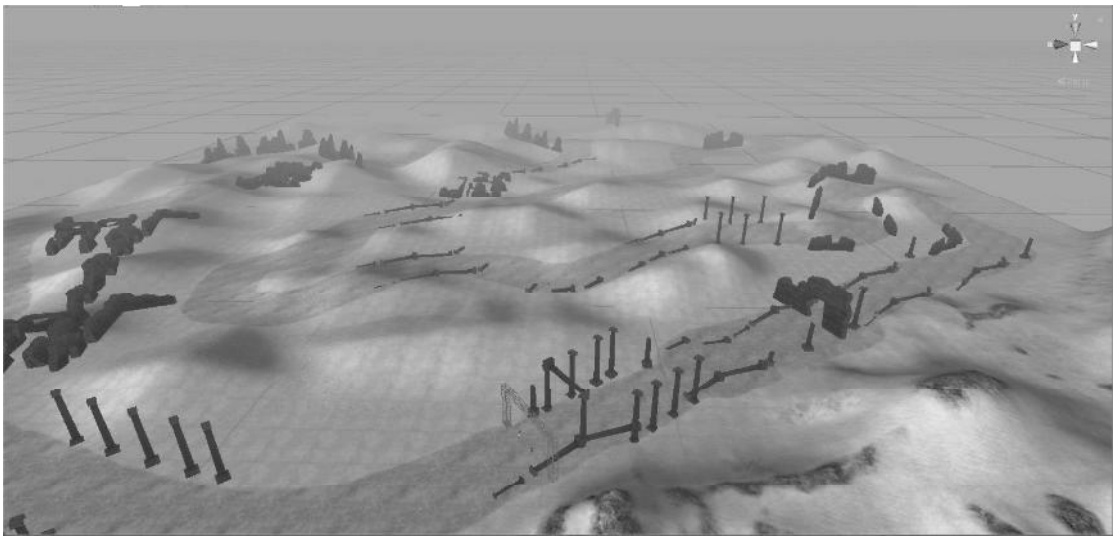
Programski kod 18. Prikaz semafora za početak igre (Izvor: Autor)

6.4. Kreiranje staze

Staza je scena u kojoj se odvijaju utrke između igrača. Kao staza korištena je besplatna postojeća scena Mountain Race Track – Night preuzeta s Unity trgovine (Slika 29.) i staza koja je napravljena pomoću Unity Terrain komponente (Slika 30.).



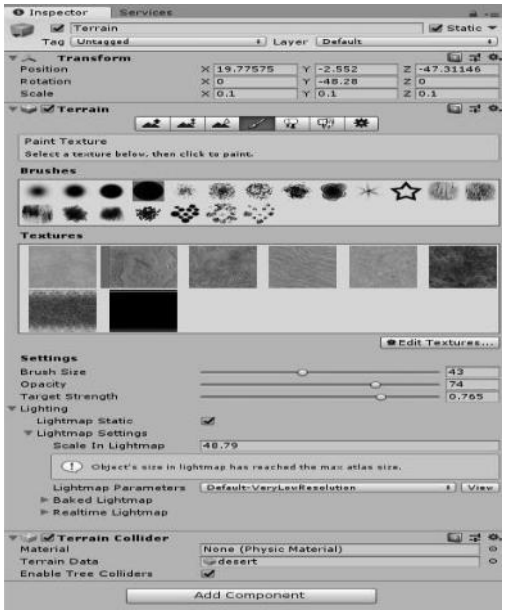
Slika 29. Mountain Race Track – Night (izvor: snimka autorova zaslona)



Slika 30. Desert (izvor: snimka autorova zaslona)

Da bi se dodao Teren Objekt na scenu, odabere se GameObject> 3D objekt> Terrain iz izbornika. Ovo također dodaje odgovarajuću imovinu terena u prikaz projekta. Krajolik je u početku velika ravnina. Prozor Inspektor Terrain pruža brojne alate za izradu detaljnih značajki pejzaža.

Alatna traka sa sedam ikona pruža opcije za oblikovanje i bojenje terena, dodavanje stabala i detalja kao što su trava, cvijeće i stijene te promjena općih postavki za odabrani teren. Slika 31. prikazuje objekt s komponentom Terrain.



Slika 31. Objekt s komponentom Terrain (izvor: snimka autorova zaslona)

Na sam teren dodani su i 3d objekti koji su preuzeti s Unity trgovine kako bi se popunile praznine na terenu. U staze su dodane i četiri prazna objekta koji su nazvani SpawnPlayer1, SpawnPlayer2, SpawnPlayer3 i SpawnPlayer4 te im je dodijeljena mrežna komponenta NetworkStartPosition koja definira poziciju i rotaciju na kojoj se mogu pojaviti igrači.

6.5. Cilj

U ovoj vrsti igre cilj je pobijediti u utrci. Pobjeda se ostvaruje tako da igrač odvozi dva kruga oko staze.

Kroz stazu su postavljene 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 pomoću skripte FinishLine. U slučaju uspješne kontrole, igrač je uspješno odvozio krug utrke. Nakon uspješno odvožena 2 kruga igraču se pokazuje na kojoj poziciji je završio te će se ime igrača spremiti kod svih sudionika utrke pod

rednim brojem. Sljedeći isječak koda pokazuje spremanje imena kod svih sudionika igre pomoću pozivne udaljene procedure [PunRPC].

```
private void OnTriggerExit(Collider other)  {
       if (other.tag == "Player") {
           int Lap = other.GetComponent<CarMain>().Lap;
           if (Lap == 2) {
      name =other.GetComponent<PhotonView> ().owner.NickName;
      photonView.RPC("Imena" , PhotonTargets.All,name);
     }    }   }
   [PunRPC]
   public void Imena(string name){
       ime [i] = name;
       i++;     }
```

Programski kod 19. Spremanje imena igrača (Izvor: Autor)

7. KORISNIČKE UPUTE

7.1. Instalacija

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

7.2. Igranje

Pokretanjem igre prikazuje se početni izbornik s dostupnim opcijama, što je prikazano na Slici 32. Uređaj koji će biti domaćin igri odabire stazu i naziv, a ostali igrači se pridružuju igri klikom na tipku List Rooms. Prije same utrke treba upisati naziv igrača u polje PLAYER NAME.

Na ovom sučelju se također i odabire vozilo kojim se želi upravljati tijekom igre. U lijevom dijelu sučelja prikazana su dostupna vozila. Klikom na pojedino vozilo, u središtu sučelja prikazuje se uvećani model odabranog vozila. Igra koristi kursorske tipke za kontrolu vozila, lijevu tipku miša za pucanje te lijevi Shift za ubrzanje.

Slika 32. Glavni izbornik (izvor: snimka autorova zaslona)

7.2.1. Utrka

Cilj igre je pobijediti u utrci. Pobjeda se ostvaruje tako da igrač odvozi dva kruga oko staze. Na početku igre čeka se na sve ostale igrače, tj. utrka ne može početi dok se svi igrači ne uključe u igru. Početak igre prikazan je na Slici 33.

Slika 33. Početak igre (izvor: snimka autorova zaslona)

Kada skripta otkrije drugoga igrača, kreće odbrojavanje za start utrke. Utrka se vozi dva kruga. Kroz stazu su postavljeni PowerUpovi koji kad se skupe daju oružje, brzinu ili popravak automobila. Klikom na tipku Disconnect igrač napušta igru i vraća se na početni izbornik.

8. ZAKLJUČAK

Računalne igre danas zauzimaju velik udio tržišta računalnog softvera te se očekuje da će u budućnosti tržište videoigara rasti. Budući da je socijalizacija važan aspekt modernog igranja, implementacija mrežnih značajki je nešto što treba razmotriti tijekom razvoja svakog većeg izdanja, kao i kod manjih neovisnih igara.

Primarni fokus ovog rada bio je osmisliti i razviti funkcionalnu višekorisničku igru koristeći Unity i Photon. Iako je Unity Engine bio poznato radno okruženje iz mog prethodnog iskustva s njim, Photon nije bio. U početku, prije stjecanja osnovnih znanja implementacije Photona, nisam imao ni najmanji trag gdje bih uopće mogao početi. Nadalje, stalno sam se suočavao s raznim problemima, za što nije bilo primjera. Iako se to može smatrati korisnim aspektom procesa učenja, ono je također dovelo do programiranja pomoću pokušaja i pogrešaka. S moje strane, cjelokupna realizacija projekta trajala je oko 300 sati, kada su svi završeni zadaci uzeti u obzir zajedno s programiranjem i pisanjem.

Usprkos činjenici da je naišlo na nekoliko teških problema, razvijanje je uglavnom bilo glatko i jednostavno. Najizazovniji zadatak bio je nepravilna sinkronizacija. Kao što pokazuju poglavlja 5 i 6, uspio sam stvoriti dva odvojena, zadovoljavajuće funkcionalna mrežna sustava predsoblja i mehanike igre korištenjem nepoznate tehnologije. Po mom mišljenju, to ispunjava u početku postavljene istraživačke ciljeve kao i poboljšanje vlastitih profesionalnih vještina.

U sklopu potencijalnih kasnijih verzija moguće je prekrajanje idejnog okruženja i dodavanje dodatnih funkcionalnosti kao što su, primjerice:

- estetski ljepši i pristupačniji dizajn elemenata i sučelja
- mogućnost kupovine novih vrsta automobila
- više staza i novih prepreka
- implementacija *chata*.

Ovaj rad ukazuje i na to da izrada igre danas nije tolika nepoznanica te da se uz nešto truda i znanja u nekoliko programa može razviti kompletna igra. Razvoj tehnologije omogućuje nam da pojedinac vlastitim trudom može napraviti igru, bez nekih prevelikih troškova te da u današnje vrijeme izrada mrežne videoigre više nije rezervirana za velike razvojne timove.

Literatura

[1]. newzoo global games market 2018 Dostupno 24. veljače 2019. na:

https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half

[2]. The History of Online Gaming by Logan Rivenes Dostupno 24. veljače 2019. na:

https://datapath.io/resources/blog/the-history-of-online-gaming/

[3] Multiplayer Game Programming: An Overview of Networked Games By Josh Glazer and Sanjay Madhav Dostupno 24. veljače 2019. na:

http://www.informit.com/articles/article.aspx?p=2461064

[4] Microchipdeveloper.com TCP/IP Five-Layer Software Model Overview. Dostupno 24. veljače 2019. na:

http://microchipdeveloper.com/tcpip:tcp-ip-five-layer-model

[5] UDP vs. TCP Which protocol is best for games? Posted by Glenn Fiedler. Dostupno 24. veljače 2019. na:

https://gafferongames.com/post/udp_vs_tcp/

[6] Studytonight.com Types of Network Topology. Dostupno 24. veljače 2019. na:

https://www.studytonight.com/computer-networks/network-topology-types

[7]. Source Multiplayer Networking,VALVE developer comunity. Dostupno 24. veljače 2019. na: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking/

[8]. Building a Peer-to-Peer Multiplayer Networked Game by. Fernando Bevilacqua Dostupno 24. veljače 2019. na: https://gamedevelopment.tutsplus.com/tutorials/building-a-peer-to-peer-multiplayer-networked-game--gamedev-10074/

[9]. Theinfozones: computer -networking. Dostupno 24. veljače 2019. na: http://www.theinfozones.com/2014/10/computer-networking.html

[10]. Fast Facts. 2016. Unity homepage. Dostupno 24. veljače 2019. na: https://unity3d.com/public-relations/

[11]. Get Unity. 2016. Unity homepage. Dostupno 24. veljače 2019. na: https://unity3d.com/get-unity/

[12]. Unity Manual: Scenes (2018.3) Dostupno 24. veljače 2019. na:

https://docs.unity3d.com/Manual/CreatingScenes.html

[13]. Unity User Manual (2018.3) Dostupno 24. veljače 2019. na:

https://docs.unity3d.com/Manual/UnityManual.html/

[14]. Unity – Manual: BuildSettings 2018 Dostupno 24. veljače 2019. na: https://docs.unity3d.com/Manual/BuildSettings.html/

[15]. Porter. 2013. Unity: Now you are thinking with components. Dostupno 24.veljače.2019. na: http://gamedevelopment.tutsplus.com/articles/unity-now-youre-thinking-with-components--gamedev-12492/

[16]. Unity – Manual: GameObjects Pristupljeno Dostupno 24. veljače 2019. na: https://docs.unity3d.com/Manual/GameObjects.html/

[17]. Unity – Manual: Prefabs Dostupno 24. veljače 2019. na: https://docs.unity3d.com/Manual/Prefabs.html/

[18]. Unity - Manual: Tag 2018 Dostupno 24. veljače 2019. na: https://docs.unity3d.com/Manual/Tags.html/

[19]. Unity - Manual: Scripting 2018 Dostupno 24. veljače 2019. na: https://docs.unity3d.com/Manual/ScriptingSection.html

[20]. Photon – Photon Unity Networking Intro. N.d. Documentation about Photon features. Dostupno 24. veljače 2019. na; http://doc.photonengine.com/en/pun/current/getting-started/pun-intro/

[21]. Photon – Feature Over-view Dostupno 24 veljače 2019. na https://doc.photonengine.com/en-us/pun/current/getting-started/feature-overview/

Popis slika

Slika 1. Prihod od videoigara u 2018. [1] 4
Slika 2. Aardwolf MUD [2] 5
Slika 3. Izazovi pri razvoju višekorisničke igre [3] 7
Slika 4. OSI-model [4] 7
Slika 5. Klijent – poslužitelj arhitektura [6] 11
Slika 6. Mreža ravnopravnih korisnika[7] 13
Slika 7. Glavno sučelje Unityja 16
Slika 8. Alatna traka 16
Slika 9. Prikaz scene 17
Slika 10. Prikaz igre 18
Slika 11. Hijerarhija 18
Slika 12. Preglednik projekta 19
Slika 13. Komponente pridodane igraču 20
Slika 14. Postavke izgradnje 21
Slika 15. Trgovina elementima 22
Slika 16. Objekt s Rigidbody komponentom 23
Slika 17. Pun i Pun plus[14] 27
Slika 18. PUN Wizard 28
Slika 19. PhotonServerSettings 29
Slika 20. PhotonView komponenta 30
Slika 21. PhotonTransformView komponenta 30
Slika 22. Tipovi automobila 36
Slika 23. PowerUp predlošci 37
Slika 24. Model auta s naoružanjem 38
Slika 25. Objekt s komponentama 39
Slika 26. Vizualni prikazi zdravlja, preostalih metaka i ubrzanja 40
Slika 27. Glavni izbornik 41
Slika 28. Dugme za pridruživanje 44
Slika 29. Mountain Race Track – Night 46
Slika 30. Desert 46
Slika 31. Objekt s komponentom Terrain 47
Slika 32. Glavni izbornik 49
Slika 33. Početak igre 50

Programski kodovi

Programski kod 1. Sudar igrača i metka 25
Programski kod 2. Stvaranje, spajanje i popis soba 31
Programski kod 3. Slanje poruka pomoću udaljenih procesnih poziva (RPC) 32
Programski kod 4. Stvaranje objekta 32
Programski kod 5. Instanciranje pomoću udaljenih procesnih poziva (RPC) 33
Programski kod 6. Prava igrača pomoću *bool* varijable 33
Programski kod 7. Prava igrača pomoću svojstva komponente 33
Programski kod 8. Korištenje funkcije OnPhotonSerializeView 34
Programski kod 9. Skupljanje PowerUp 37
Programski kod 10. Pucanje i stvaranje metka 40
Programski kod 11. Povezivanje na poslužitelja 42

Programski kod 12. Imenovanje igrača 42
Programski kod 13. Stvaranje sobe 43
Programski kod 14. Popis slobodnih soba 43
Programski kod 15. Pridruživanje sobi 44
Programski kod 16. Stvaranje igrača 45
Programski kod 17. Poziv na početak igre svim igračima 45
Programski kod 18. Prikaz semafora za početak igre 46
Programski kod 19. Spremanje imena igrača 48

Sažetak

U današnje vrijeme igre su najčešći oblik digitalne zabave te su ušle u sve pore svakodnevnog života i ne poznaju dobne granice. Budući da je socijalizacija važan aspekt modernog igranja, implementacija mrežnih značajki je nešto što treba razmotriti tijekom razvoja svakog većeg izdanja, kao i kod manjih neovisnih igara.

U ovom radu istraženo je mrežno rješenje kako bi se implementirala *online* funkcionalnost u prototip igre korištenjem Unityja. Metode implementacije koja se koristila je mrežni dodatak tvrtke Unity: Photon Unity Networking. Cilj je bio postići funkcionalnu *online* igru za više igrača.

Ključne riječi: Unity, Photon, Mrežne igre, C#

Summary

Nowadays, games are the most common form of digital entertainment and have entered into all the pores of everyday life and do not know the age limit. Since socialization is an important aspect of modern gaming, the implementation of network features is something that should be considered during the development of every major release, as well as in smaller independent games.

This paper explored a network solution to implement online functionality in a prototype game using Unity. The implementation methods used by the Unity Network Adapter: Photon Unity Networking. The goal was to achieve a functional online multiplayer game.

Keywords: Unity , Photon, Multiplayer games, C#