Online natjecateljska igra za više igrača sa sustavom rangiranja

Rabar, Marin

Master's thesis / Diplomski rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:053224

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

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

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Diplomski studij računarstva

Online natjecateljska igra za više igrača sa sustavom rangiranja

Diplomski rad

Marin Rabar

Osijek, 2018.

SADRŽAJ

1. UVOD 1
   1.1. Zadatak diplomskog rada 1
2. TEHNOLOGIJE 2
   2.1. Unity 2
   2.2. Visual studio 3
   2.3. C# 3
   2.4. Shader 3
   2.5. Pyxel edit 5
   2.6. Itch.io 6
3. ONLINE MULTIPLAYER 8
   3.1. Igra za više igrača 8
   3.2. Game server 8
   3.2.1. Namjenski poslužitelj 8
   3.2.2. Lokalni poslužitelj 9
   3.2.3. Peer-to-Peer (vršnjak-vršnjak) 10
   3.3. Unity Multiplayer (UNET) 10
4. RAZVOJ IGRE 13
   4.1. Ideja 13
   4.2. Likovi / kontroler 13
   4.2.1. Kontrole kretanja 14
   4.2.2. Kontrole oružja 16
   4.2.3. Fizika 17
   4.3. Okolina/ dizajn razine 17
   4.3.1. Mapa 17
   4.3.2. Protivnici 18
   4.3.3. Oružje 18
   4.4. Mrežna implementacija 20
   4.4.1. Unity Networking 23
   4.4.2. Player Lobby 32
   4.4.3. Unity Multiplayer servis 35
   4.5. Natjecateljska igra 36

4.5.1. Pobjeda 37
4.6. Grafika 38
4.6.1. Mapa 38
4.6.2. Likovi 40
4.7. Posljednji detalji i build 43
5. ZAKLJUČAK 44
LITERATURA 45
SAŽETAK 46
ABSTRACT 46
ŽIVOTOPIS 47

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 03.09.2019.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Marin Rabar
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 891 R, 26.09.2018.
OIB studenta:	08795412668
Mentor:	Doc.dr.sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva:	Robert Šojo
Naslov diplomskog rada:	Online natjecateljska igra za više igrača sa sustavom rangiranja
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno opisati teorijske osnove za stvaranje igre za više igrača s bazom podataka u svrhu prikupljanja podataka. Potrebno je opisano implementirati u online verziju igre koja bi bila stalno dostupna.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	03.09.2019.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 23.09.2019.

Ime i prezime studenta:	Marin Rabar
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 891 R, 26.09.2018.
Ephorus podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Online natjecateljska igra za više igrača sa sustavom rangiranja**

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

i sumentora

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

Potpis studenta:

1. UVOD

Tema ovog diplomskog rada je izrada natjecateljske igre s mogućnosti online umrežavanja s drugim igračima te mogućnosti rangiranja među igračima. Igra je u 2D stilu radi jednostavnosti izrade te je žanr igre akcijski shooter odnosno pucačina. Osnovna problematika ovog rada je stvoriti stabilnu igru koju je moguće postaviti na server preko kojega se više igrača može pridružiti i natjecati za najbolji rezultat. Najveća prepreka je načiniti dobru vezu između poslužitelja i klijenta kako bi kašnjenje među korisnicima bilo što manje te ne bi stvaralo prednosti ni jednom igraču. Cilj je napraviti igru u kojoj svi imaju jednake mogućnosti te vještina igrača odlučuje tko je najbolji, a sreća ne igra ulogu. U radu su objašnjene tehnologije koje su se koristile u realizaciji cijelog projekta, tip igre i bitni elementi koji utječu na stvaranje iste te određeni izbori i opcije koje su korišteni za realizaciju pojedinih dijelova same igre.

1.1. Zadatak diplomskog rada

Zadatak rada je napraviti online natjecateljsku igru sa sustavom rangiranja. Potrebno je opisati teorijske osnove za stvaranje igre za više igrača s bazom podataka u svrhu prikupljanja podataka te načiniti online verziju igre iz opisanog koja bi bila stalno dostupna.

2. TEHNOLOGIJE

2.1. Unity

Unity je višeplatformsko razvojno okruženje za izradu video igara razvijeno od kompanije Unity Technologies. Unity obuhvaća trenutno 27 platformi kao što su PC, Mac, Linux, konzole, mobilni i internet stranice te konstantno širi svoj opseg. Pruža mogućnost i podršku za razvoj 2D i 3D igara i simulacija te nudi API za kodiranje u C#-u u obliku pluginova za unity editor te drag and drop funkcionalnosti. Osnovni podržan programski jezik je C#. Nudi podršku Direct3D grafičkog API-a za Windows i Xbox One, OpenGL za Windows, Linux i MacOS, OpenGL ES za Android i iOS,WebGL za web te druge za konzole. Daje mogućnost pisanja vlastitih shadera koji omogućuju definiranje svakog izvršnog pixela/vertexa/poligona. Unity nudi četiri opcije licenciranja: personal, plus, pro i enterprise od kojih je personal licenca besplatna. Personal licenca nudi skoro sve iste mogućnosti Unityja kao i ostale verzije te ju je potrebno nadograditi na plus ili pro tek nakon određenog prihoda od igre. Unity sučelje se sastoji od raznih prozora koji koriste za izradu određenih aspekata igre. Osnovni prozor je scena na kojemu se odvija glavni dio slaganja razrade igre. U njemu se postavljaju izrađeni objekti te se namještaju i modificiraju što direktno utječe na izgled igre i ono što vidimo kada pokrenemo igru. Igra se uglavnom sastoji od više scena (npr. svaki nivo ima svoju scenu). Objekti čine osnovne cjeline od kojih se igra sačinjava te listu svih objekata u sceni vidimo u prozoru hijerarhije. Objekti mogu sadržavati širok opseg različitih opcija kao što su grafike koje prikazuju izgled objekta u igri, zvukovi koje objekt projicira te skripte pisane od strane programera koje utječu na ponašanje objekta u prostoru, interakcije, vizuale, animacije i drugo. Sve nadodane opcije na objektu možemo vidjeti u prozoru inspector gdje možemo izravno manipulirati izloženim varijablama objekata (npr. položaj, veličina, glasnoća zvuka, odabir boja). U project prozoru uvozimo vlastite grafike, zvukove, modele, skripte i druge materijale od kojih kasnije možemo stvarati objekte ili pridruživati drugim objektima. Unity pruža servise poput Unity Ads za implementaciju monetizacije u igru, Unity Analitics za praćenje ponašanja igrača, Unity Multiplayer za laku implementaciju multiplayera u igru i druge. Unity multiplayer servis je najlakši način za postavljanje mrežne igre za Unity. Unity osigurava svoje servere i matchmaking servisi omogućuju lagano nalaženje drugih igrača. U personal verziji Unity-a omogućuje 20 istovremenih igrača na serveru.

2.2.Visual studio

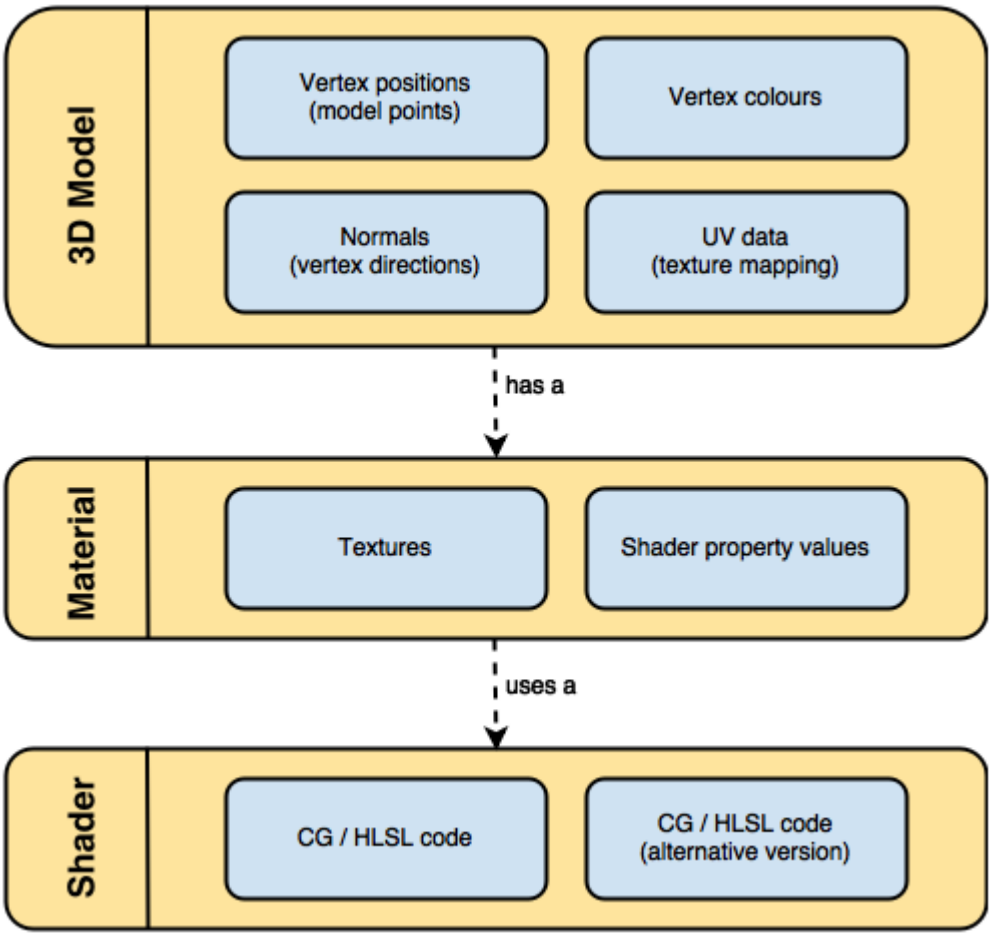
Microsoft Visual studio je integrirano razvojno okruženje od Microsofta. Koristi se za izradu kompjuterskih programa, web stranica, web aplikacija i mobilnih aplikacija. Visual studio podržava 36 različitih programskih jezika od kojih se koristi C# za razvoj igre u Unityju. Dolazi u tri edicije: Community, Professional i Enterprise. Community verzija je besplatna. Unityjeva integracija Visual studija omogućava kreiranje i održavanje Visual Studio projektnih datoteka automatski. Visual studio se otvara duplim klikom na neku od skripti u Unity-u te se greške prikazuju u Unity konzoli. Za korištenje Visual studia kao osnovnog script editora za Unity potrebno je postaviti opciju u Unity referencama.

2.3. C#

C# je objektno orijentirani programski jezik razvijen u Microsoftu s ciljem da .NET platforma dobije programski jezik. Namijenjen je da bude jednostavan, moderan, objektno orijentiran jezik. Pruža potporu za strong type provjeru, provjeru veličine polja, pronalazak i pokušaj korištenja neinicijaliziranih varijabli i automatsko sakupljanje smeća. C# je osnovni podržan jezik za pisanje koda u Unityju.

2.4. Shader

Shader je program specifično napravljen da se odvija na GPU. Omogućava definiranje svakog izvršenog pixela, vertexa ili poligona. Koriste se kako bi se dao specifičan izgled igri, uglavnom kod 3D igara no imaju svrhu i u 2D-u. 3D modeli su sačinjeni od skupa 3D koordinata koje se nazivaju verteksi. Oni su međusobno povezani i čine trokute. Svaki verteks može sadržavati nekoliko dodatnih informacija kao boja, direkcija u koju usmjeruje i neke koordinate za mapiranje tekstura (UV data). Modeli se ne mogu renderirati bez materijala. Materijali su „omotači“ koji sadrže shadere i njihove vrijednosti tako da različiti materijali mogu sadržavati isti shader i davati mu različite vrijednosti. Slikom je prikazano navedeno.



Slika 2.1 - shader

Unity podržava dvije vrste shadera: surface (površinski) shadere i fragment i verteks shadere. Anatomija shadera neovisno o vrsti je uvijek ista.

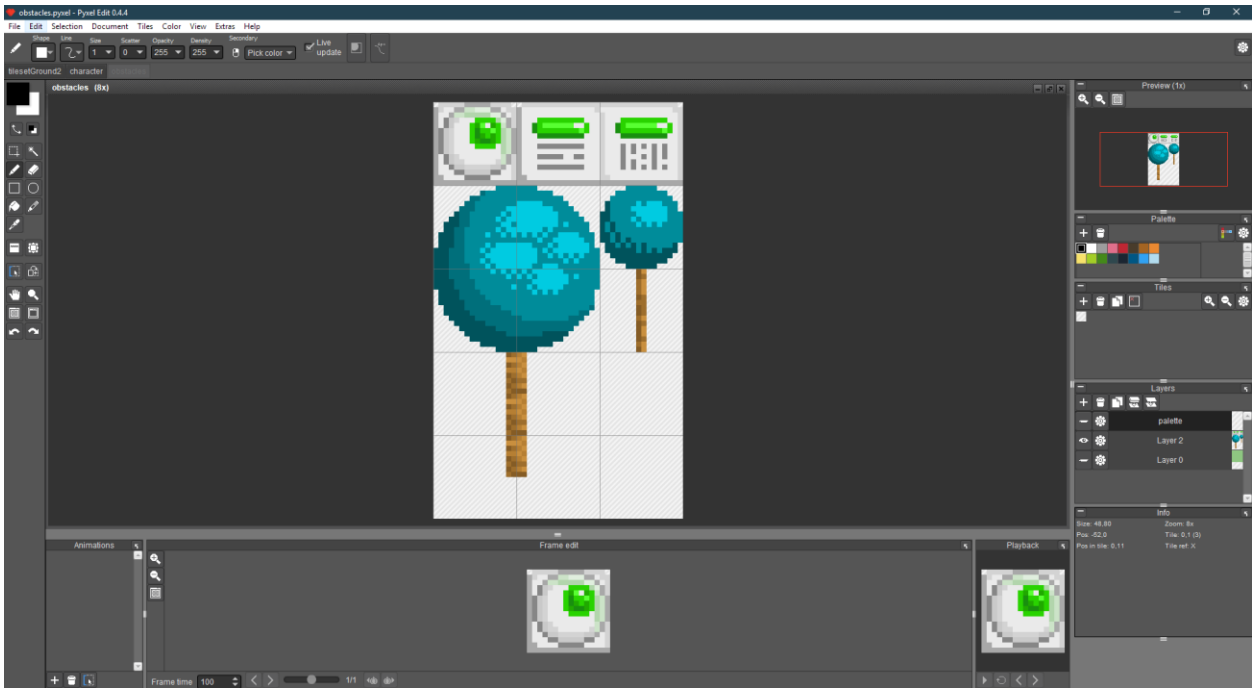
```
Shader "MyShader"
{
    Properties
    {
        // The properties of your shaders
        // - textures
        // - colours
        // - parameters
        // ...
    }

    SubShader
    {
        // The code of your shaders
        // - surface shader
        //    OR
        // - vertex and fragment shader
        //    OR
        // - fixed function shader
    }
}
```

Slika 2.2 - anatomija shadera

Shader može sadržavati više SubShader sekcija i one sadržavaju instrukcije za GPU te ih Unity izvodi po redu.

2.5. Pyxel edit

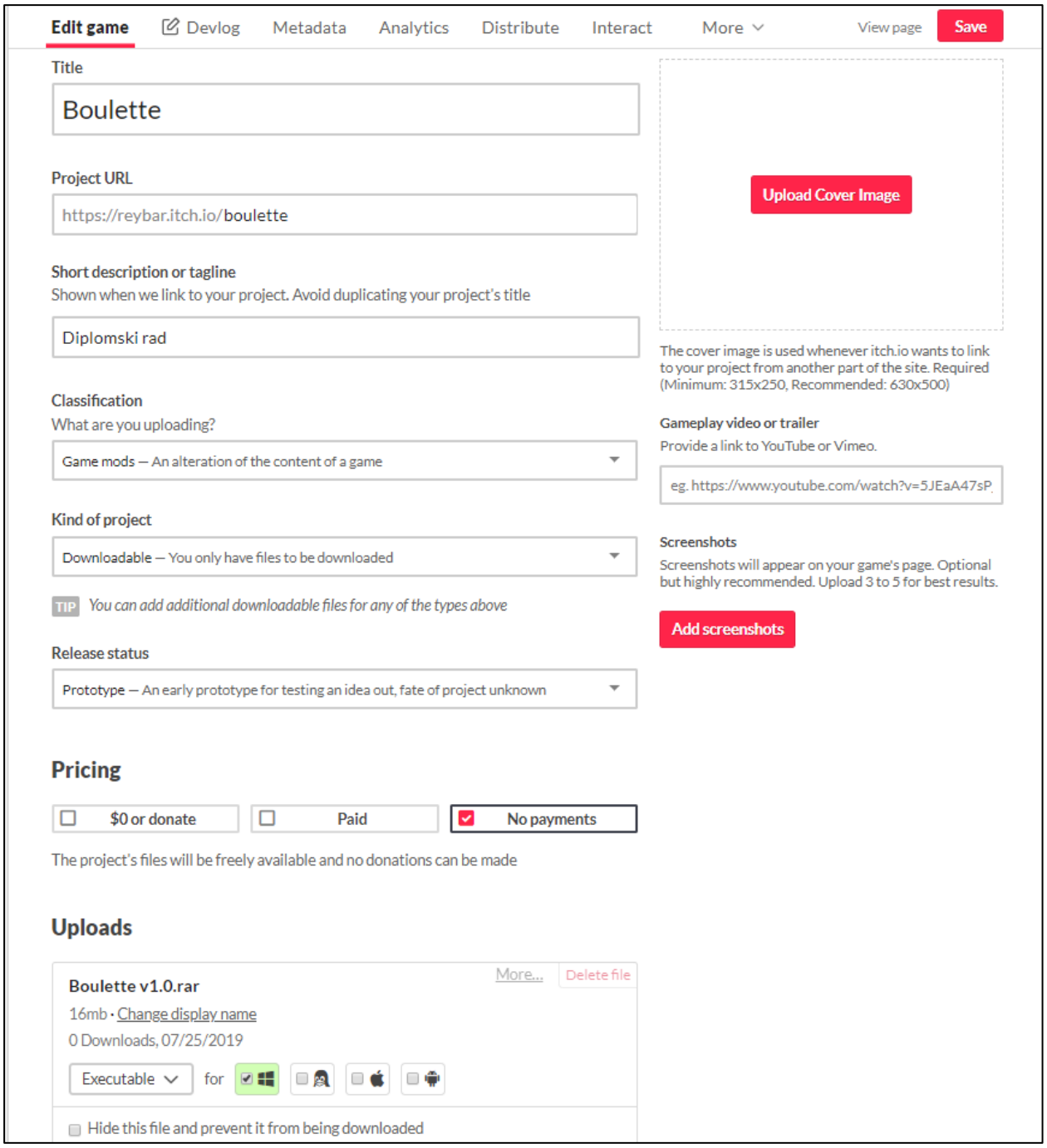
Pyxel edit je piksel art editor za laku izradu *tileseta*, nivoa i animacija. Pyxel edit je jedan od najboljih alata za izradu pikselizirane grafike. Dok mu je sučelje i dio funkcionalnosti sličan poznatijim alatima poput Photoshopa, Pyxel edit je fokusiran za rad s pikseliziranom grafikom bez potrebe za dodatnim postavkama. Nudi mogućnost lakog manipuliranja većeg broja *tileova* putem jednog osnovnog *tilea* te rotaciju tilea bez gubitka originalne reference slike. Omogućava laku izradu piksel art animacija te njihovo lako izvoženje u obliku spritesheetova ili gifa te opciju onion skinning za ublaživanje tranzicija među spriteovima. Moguće je lako uvoziti slike *tilesetova* te automatski identificirati tileove te izvoziti *tilemape* u XML, JSON i text formatu. Trenutno je u beta verziji i dostupan je za $9.00.



Slika 2.3 - Sučelje Pyxel edit alata

2.6.Itch.io

Itch.io je web stranica na kojoj korisnici mogu lako hostati, prodavati i skidati indi video igre. Na servisu se nalazi skoro 100,000 igara zabilježeno u veljači 2018. godine. Itch.io također dozvoljava korisnicima hostanje game jamova za vrijeme kojih sudionici imaju ograničeno vrijeme (uobičajeno 1-3 dana) da naprave igru. Neki od poznatijih gane jamova na itch.io uključuju Game Off i Game Maker Toolkit Game Jam. Nakon registracije, itch.io nudi razne mogućnosti za postavljanje i monetizaciju igre. Moguće je klasificirati vrsta softvera koja se postavlja od igara do knjiga i glazbe, status u kojem se projekt nalazi, način monetizacije: donacije, fiksna cijena ili bez plaćanja te prihvaća igre za platforme: widnows, linux, iOS, android i web. Također je moguće urediti stranicu igre po želji.



Slika 2.4 - itch.io postavke za igru

3. ONLINE MULTIPLAYER

Online igre su igre koje se djelomično ili u potpunosti igra je putem interneta ili neke druge mreže. Online igre su svuda prisutne na igraćim platformama današnjice, uključujući osobna računala, konzole i mobilne telefone te obuhvaćaju mnoge žanrove poput pucačina (FPS), strategija (RTS) i masivnih mrežnih online igara uloga (MMORPG). Dizajn online igara seže od jednostavne tekstualne okoline do kompleksnih grafika i virtualnih stvarnosti. Online komponente u igri mogu biti jednostavne poput rang ljestvica za uspoređivanje rezultata ili kompleksije gdje je srž igre u direktnom igranju s i protiv drugih igrača. Mnoge online igre imaju svoje društvene zajednice što ih čini nekom vrstom socijalnih aktivnosti za razliku od *offline* igara.

3.1. Igra za više igrača

Igre za više igrača ili multiplayer igre su igre u kojima, kao što samo ime govori, sudjeluje više od jednog igrača u isto vrijeme u istom igračem prostoru. Multiplayer igre mogu biti offline, što zahtjeva da igrači dijele igrači sistem, ili online koristeći mrežne tehnologije igrajući na veće udaljenosti. U igrama za više igrača, igraći mogu međusobno sudjelovati kako bi dostigli zajednički cilj ili se natjecati jedni protiv drugih za pobjedu. Igre za više igrača su veoma prikladne za korištenje online tehnologija te su u današnjici sve više popularne online multiplayer igre. Ta popularnost je dovela do profesionalnog bavljenja s kompetitivnim video igrama i stvaranja termina esport odnosno elektronički sportovi (electronic sports) što se odnosi na najpopularnije natjecateljske igre. Najveće takve igre su League of Legends, Dota 2, Counter Strike, Hearthstone. Najbitnija filozofija u dizajnu kompetitivnih natjecateljskih igara je da su igre jednako balansirane za sve igrače te da uspjeh ovisi o vještinama samog igrača a što manje o sreći.

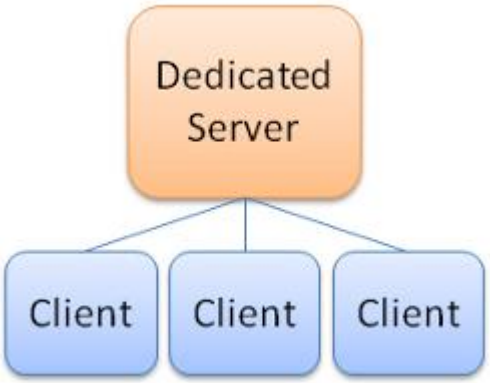
3.2. Game server

Najbitnija razlika u izradi online naspram offline igre je potreba za game serverom. Game server ili poslužitelj je poslužitelj koji je autoritativan izvor događaja multiplayer online igrama. Poslužitelj prenosi podatke o svom stanju spojenim klijentima kako bi održavali vlastitu preciznu verziju svijeta koju prikazuju igraču. Također primaju i obrađuju sve unosne podatke od strane igrača.

3.2.1. Namjenski poslužitelj

Namjenski ili posvećeni poslužitelji simuliraju svijet igre ne podržavajući izravne ulaze i izlaze osim onih potrebnih za administraciju. Igrači se moraju povezasti s poslužiteljem s odvojenog

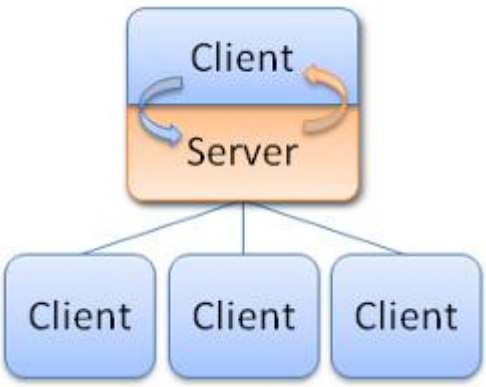
klijenta kako bi mogli vidjeti i komunicirati s igrom. Najveća prednost namjenskog poslužitelja je ta da su pogodni za hosting u profesionalnim podatkovnim centrima, kao i njihova pouzdanost i performanse. Također uklanjaju prednost niske latencije koje bi imali igrači koji hostaju server. S druge strane namjenski poslužitelji koštaju novaca da se vode i održavaju.



Slika 3.1 - Klijent/poslužitelj model s namjenskim poslužiteljem

3.2.2. Lokalni poslužitelj

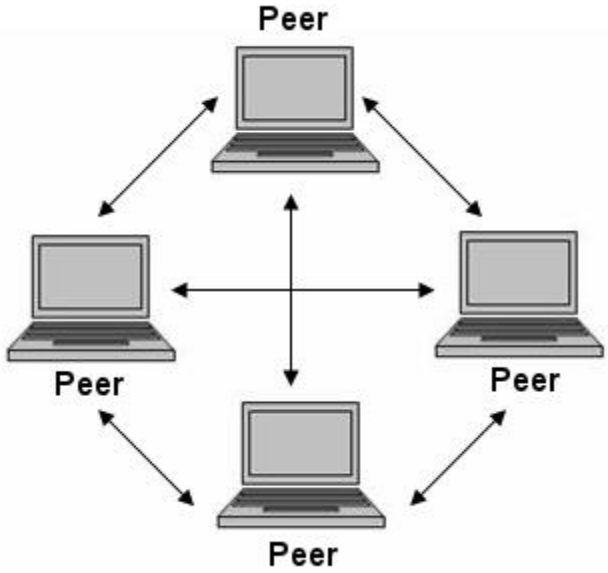
Kod lokalnog poslužitelja, poslužitelj je pokrenut u istom procesu kao i klijent. Imaju gotovo jednaku funkcionalnost kao i namjenski poslužitelji, no obično imaju nedostatak komunikacije s udaljenim igračima putem rezidencijalne internetske mreže igrača poslužitelja. Performanse su mu također ograničene kvalitete stoja koji ga pokreče te činjenicom da taj stroj uz poslužitelj mora pokretati i klijenta. Lokalni poslužitelji isto tako daju veliku prednost smanjene latencije igraču koji ih pokreče. Usprkos tomu lokalni poslužitelji imaju veliku prednost toga da su besplatni bez potrebe za dodatnom infrastrukturom.



Slika 3.2 - Klijent/poslužitelj model s lokalnim poslužiteljem

3.2.3. Peer-to-Peer (vršnjak-vršnjak)

Alternativno klijent/poslužitelj modelu navedenom gore, kod peer-to-peer modela poslužitelj ne postoji. Umjesto toga svaki „vršnjak“ prima sirov dotok podataka od ostalih igrača i sam procesira rezultate. Peer-to-peer model se smatra zastarjelim za akcijske igre no još se može naći u strateškim igrama zbog velikog broja stanja a malog broja igrača. Najveće su mu mane: da je teško sve peerove sinkronizirati, da je teško podržati priključenje novih peerova usred igre, zbog potrebe za komunikacijom svih peerova međusobno je broj igrača ograničen te su svi igrači osušeni na latenciju igrača s najgorom konekcijom.

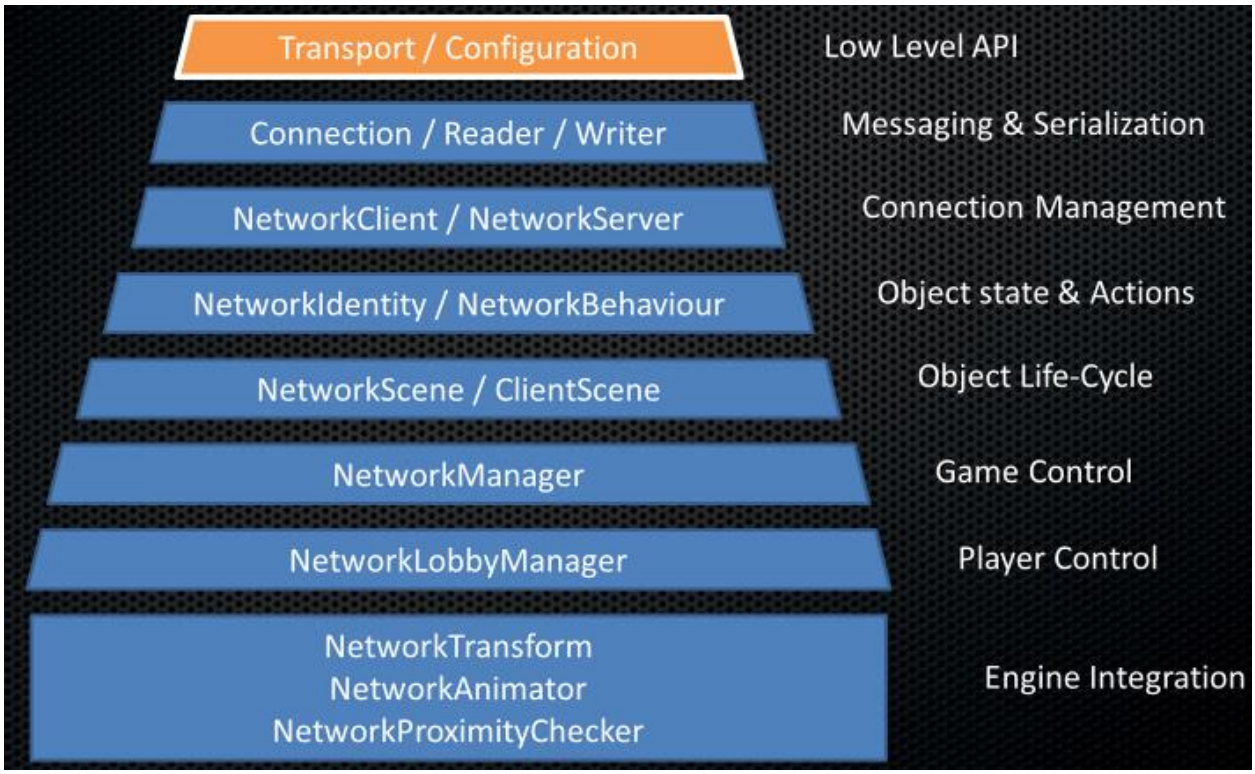


Slika 3.3 - Peer-to-peer model

3.3. Unity Multiplayer (UNET)

Unity Multiplayer servis je najjednostavniji način za izradu online multiplayer igara u Unityu. Unet je Unityevo rješenje klijent/poslužitelj arhitekture za umrežavanje te nudi korisniku širok spektar mogućnosti od automatske sinkronizacije objekata u igri do direktnog slanja bajtova kroz mrežu u igri. Radi tako da svaki stroj pokreče svoju inačicu igre s istim kodom i stanjima i izgledom. Kod zna vrti li se na poslužitelju ili klijentu i ima li autoritet ili ne te na temelju toga vrši radnju. Za igrale objekte klijent tog igrača ima autoritet dok poslužitelj ima prava nad objektima koji nisu.Sve radnje se procesiraju na poslužitelju te zatim sinkroniziraju sa svim klijentima. Network Manager objekt u Unityu je način pristupa Unet tehnologiji te se preko njega hosta i pridružuje igri i igraćoj sesiji. Unity putem uneta nudi visoko-razinsko aplikacijsko programsko sučelje (HLAPI) kojim se dobiva pristup kontrolama koja obuhvaćaju većinu

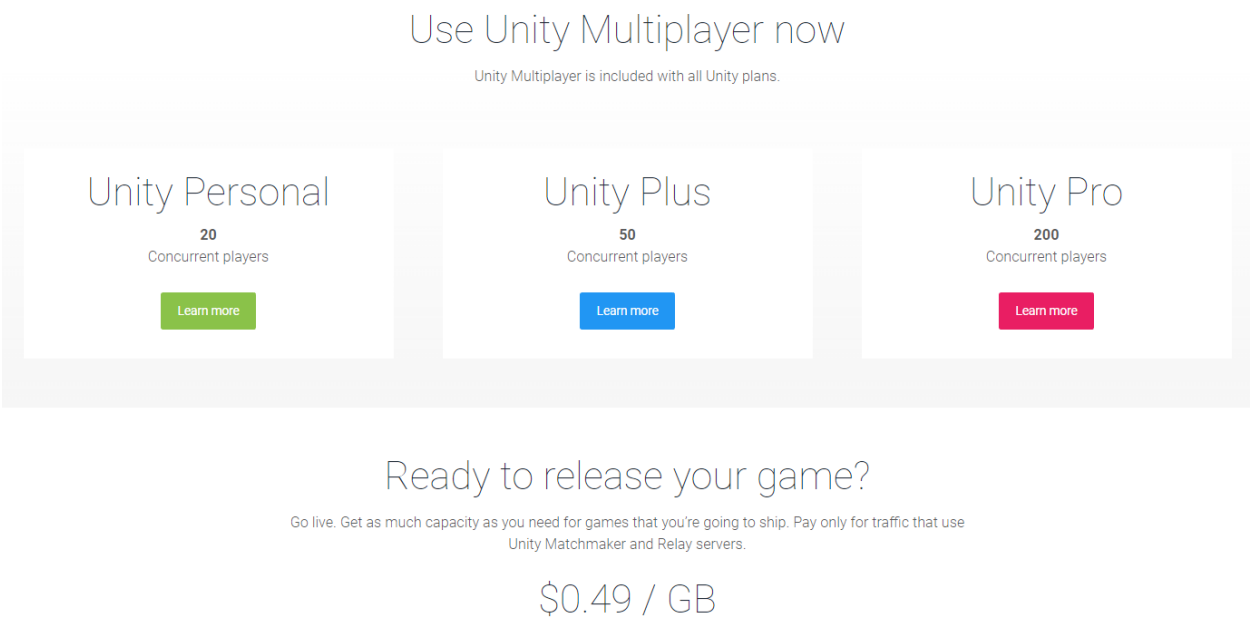
potreba za izradu igara za više igrača bez potrebe za brigom o detaljima provedbe na „nižoj razini“. HLAPI omogućuje: kontrolu mrežnog stanja igre putem Network Managera, rad putem lokalnog poslužitelja, slanje i primanje mrežnih poruka, slanje naredbi (command) od klijenta poslužitelju, pozivanje udaljenih postupaka (RPC) s poslužitelja klijentu te slanje mrežnih događaja s poslužitelja.



Slika 3.4 - Hlapi slojevi

Unet je integriran u sam razvojni sklop, omogućavajući lakšu izradu multiplayer igre. Nudi NetworkIdentity komponentu za mrežne objekte, NetworkBehavior za mrežne skripte i druge mrežne komponente i omogućuje rad s tim komponentama direktno u Unity scenama.

Unity također nudi internet servise tijekom razvoja i izdavanja igre koji uključuju: *matchmaking* servise, kreiranje i promoviranje mečeva, prikaz i priključivanje mečevima, igru putem interneta bez namjenskog poslužitelja i sl. Unity Multiplayer servis u igrinom razvoju dozvoljava limitiran broj istovremenih igrača na servisu koji u besplatnoj verziji Unitya iznosi dvadeset, a nakon što je igra izdana naplaćuje servis $0.49/GB za što je također potrebno posjedovati plaćenu verziju Unitya.



Slika 3.5 - Unity Multiplayer biznis model

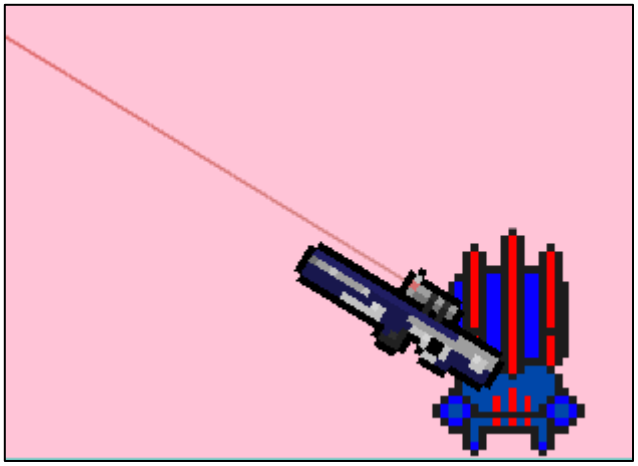
4. RAZVOJ IGRE

4.1. Ideja

Ideja projekta bila je napraviti multiplayer igru u 2D stilu s elementima akcijskih shootera. Za inspiraciju su uzete igre Showdown Effect i Tee Worlds. Igra je zamišljena da više igrača može kreirati igru te se međusobno napucavati na nivou s raznim puškama i kretati većim brojem opcija za mobilnost. U igri je svaki igrač za sebe i cilj mu je skupiti što veći broj ubojstava nad drugim igračima. Nivo se sastoji od platformerskih elemenata gdje igrač ima mogućnost kretnje po x i y osi nivoa. Igra je namijenjena za ima brz, akcijski *gameplay* te da se igrači konstantno kreću po nivou u potrazi za oružjem i izbjegavanju tuđih metaka i prepreka. Igra treba biti balansirana tako da svi igrači imaju iste uvijete za igru i uspjeh ovisi o njihovim vještinama a ne o sreći. Igra treba sakupljati podatke tokom igrače sesije poput ubojstva, smrti, rezultata i sl. te imati opciju za prikaz igračima kako bi imali uvid u trenutno stanje te na kraju sesije na temelju podataka prikazati pobjednika. Igra treba biti dostupna online svima na raspolaganju.

4.2. Likovi / kontroler

Igrači lik (player character) je osnovni element u igri, njime upravlja igrač i tako prisustvuje u igri. Osim igračevog lika, igru čine protivnički likovi, kojima upravljaju drugi igrači, s kojima se može uči u interakciju te je to osnova za igru s više igrača. Za kontroliranje lika potrebno je na objekt dodati kontroler skripte preko kojih igrač utječe na lika u 2D prostoru te ima interakciju s okolinom. Svi likovi imaju dvadeset životnih bodova.



Slika 4.1 - Izgled lika

4.2.1. Kontrole kretanja

Za izradu kontrolera potrebno je definirati osnovne mehanike s kojima se može upravljati likom. Pošto se radi o 2D igri potrebno je imati mogućnost kretanja po x i y osi. Ideja je igrač ima više opcija kretanja kako bi se mogao snaći u raznim situacijama te imao više fleksibilnosti u igri. Isto tako zamišljeno je da igra ima određene prepreke koje se može proći pomoću tih opcija. U kontroler su dodane osnovne kontrole trčanja lijevo-desno po horizontali te kontrola skoka za kretanje po vertikali. Osim toga dodane su opcije za skok od zida kako bi se moglo doći do viših dijelova mape, te *dash* kako bi mogao prijeći šire horizontalne prepreke poput rupa i dobiti mogućnost bržeg smanjenja udaljenosti s protivnikom. Radi kontrola kretnje bitno je imati definiranu interakciju s okolinom. Kod skoka bitno je da je lik prizemljen te da je točno definirano što je zemlja kako ne bi bilo neželjenog ponašanja poput dodatnih skokova u zraku ili nemogućnosti skoka kada se ponovno lik prizemlji. Isto tako kod skoka od zida potrebno je imati definirano što je zid kako bi željena funkcionalnost bila što bolje izvedena. Kontrole za kretnju se unose pomoću tipkovnice: A i D je trčanje, *space* i W su skokovi, shift je *dash*.

```
private void Movement()
{
    moveDirection = Input.GetAxisRaw("Horizontal");
    Move();
    if (Input.GetButtonDown("Jump")) {
        Jump();
    }
    if (Input.GetButtonDown("Dash") && !dashCD) {
        Dash();
    }
}

1 reference | mrabar, 146 days ago | 1 change
private void Move()
{
    if (isGrounded) {
        rigidBody.AddForce(new Vector2(moveDirection, 0) * acceleration);
    } else {
        rigidBody.AddForce(new Vector2(moveDirection, 0) * (acceleration - 10));
    }

    if (moveDirection > 0 && !facingRight) {
        Flip();
    } else if (moveDirection < 0 && facingRight) {
        Flip();
    }
}


4 references | mrabar +3, 146 days ago | 5 changes
private void Flip()
{
    facingRight = !facingRight;
    characterBody.transform.localScale
        = new Vector2(characterBody.transform.localScale.x * -1, characterBody.transform.localScale.y);
    CmdScale(characterBody.transform.localScale.x);
}
```

Slika 4.2 - Osnovne kontrole kretanja

```
private void Jump()
{
    if (isGrounded) {
        RegularJump();
    } else if (wallRide) {
        WallJump();
    } else if (fallJump) {
        RegularJump();
    }
}

2 references | mrabar, 38 days ago | 3 changes
private void RegularJump()
{
    AudioSource.PlayClipAtPoint(jumpClip,transform.position);
    rigidBody.velocity = new Vector2(rigidBody.velocity.x, 0);
    rigidBody.AddForce(Vector2.up * jumpVelocity, ForceMode2D.Impulse);
    fallJump = false;
}

1 reference | mrabar, 146 days ago | 1 change
private void WallJump()
{
    isInMotion = true;
    rigidBody.velocity = new Vector2(0, 0);
    if (facingRight) {
        Flip();
        rigidBody.AddForce(new Vector2(-wallJumpVelocity, wallJumpVelocity - 5), ForceMode2D.Impulse);
    } else {
        Flip();
        rigidBody.AddForce(new Vector2(wallJumpVelocity, wallJumpVelocity - 5), ForceMode2D.Impulse);
    }
    StartCoroutine(WallJumping(wallJumpDuration));
}

1 reference | mrabar, 146 days ago | 1 change
private IEnumerator WallJumping(float walljumpTime)
{
    yield return new WaitForSeconds(wallJumpDuration);
    isInMotion = false;
}

0 references | mrabar, 136 days ago | 1 change
private void FallJump()
{
    rigidBody.velocity = new Vector2(rigidBody.velocity.x, 0);
    rigidBody.AddForce(Vector2.up * jumpVelocity, ForceMode2D.Impulse);
    fallJump = false;
}
```

Slika 4.3 - Kontrole skakanja

```
private void Dash()
{
    dashCD = true;
    isInMotion = true;
    rigidBody.velocity = new Vector2(0, rigidBody.velocity.y);
    if (facingRight) {
        rigidBody.AddForce(Vector2.right * dashVelocity, ForceMode2D.Impulse);
    } else {
        rigidBody.AddForce(Vector2.left * dashVelocity, ForceMode2D.Impulse);
    }
    StartCoroutine(Dashing(dashTime));
}

1 reference | mrabar, 146 days ago | 2 changes
private IEnumerator Dashing(float dashTime)
{
    yield return new WaitForSeconds(dashTime);
    isInMotion = false;
    yield return new WaitForSeconds(1);
    dashCD = false;
}
```

Slika 4.4 - Kontrole za ubrzanje

4.2.2. Kontrole oružja

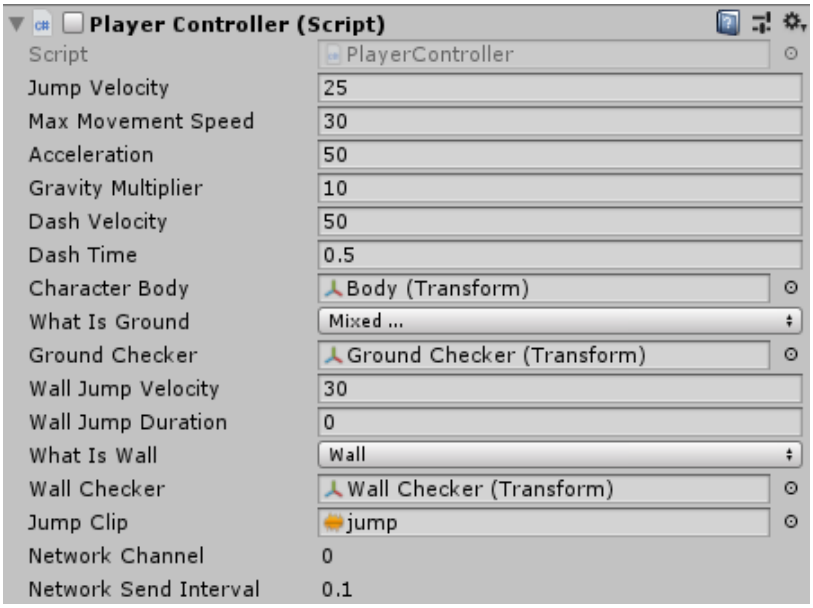
Osim kontrola za kretanje u ovoj igri su bitne kontrole za upravljanje oružjem. Nakon što igrač pokupi oružje mora moći nišaniti s njim te pucati iz njega te imati opciju mijenjanja municije. Igrač nišani pomoću pomicanja miša te puca pritiskom na lijevu tipku miša. Metak se ispucava u smjeru kursora miša u odnosu na lika te je nišanjenje dodatno prikazano kamerom koja se izduljuje u smjeru kojem se nišani. Nakon istrošenih metaka u magazinu ili po potrebi moguće je pomoću tipke R ili automatski zamijeniti municiju. Igrač može kod sebe držati najviše dva oružja te je u mogućnosti mijenjati između njih pomoću tipki 1 i 2 na tipkovnici. Oružje je moguće i baciti pomoću tipke F.

```
void Rotate()
{
    Vector2 distance = Camera.main.ScreenToWorldPoint(Input.mousePosition) - pivot.transform.position;
    distance.Normalize();
    float rotation = Mathf.Atan2(distance.y, distance.x) * Mathf.Rad2Deg;
    pivot.transform.rotation = Quaternion.Euler(0, 0, rotation);

    if (rotation > 90 || rotation < -90) {
        weaponLoc.transform.localScale = new Vector2(1, -1);
    } else {
        weaponLoc.transform.localScale = new Vector2(1, 1);
    }
    CmdFlip(weaponLoc.transform.localScale.y);
}
```

Slika 4.5 - Kontole nišanjenja

4.2.3. Fizika

Fizika je bitan element u igri koji direktno utječe na lika i njegovu interakciju s okolinom. Fizika u igri simulira fiziku stvarnog svijeta poput gravitacije trenja i sl. te je dodana kako bi kretnje i igra u cijelosti imale realističniju dinamiku. Kako bi fizika djelovala na lika potrebno mu je u Unityu na objekt dodati kruto tijelo (rigid body). Kada igrač skoči simulirana gravitacija ga prizemljuje pod definiranom razinom akceleracije. Kada igrač trči i stane trenje određene podloge mu diktira koliko brzo će stati. Stoga je bitno dobro namjestiti parametre fizike te parametre kontrole likova u odnosu na nju kako bi imali željeno ponašanje lika.



Slika 4.6 - Postavke Player Controller komponente

4.3. Okolina/ dizajn razine

Okolinu predstavlja mapa na kojoj se igra odvija te sve objekte s kojima igrač ima mogućnost interakcije.

4.3.1. Mapa

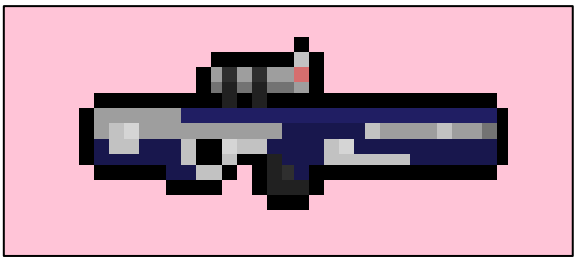
Mapa zauzima najveći dio scene i predstavlja statičnu okolinu igre na njoj se igra odvija i ona definira granice igre. Mapu predstavljaju zemlja te zidovi u igri kao i prepreke i platforme. Igračeva interakcija s mapom je takva da se uz pomoć kontrola i utjecaja fizike igrač može po njoj kretati, trčati po zemlji te odskakivati od zidova. Tlo sprječava da lik pod utjecajem gravitacije pada u nedogled. Svaka površina ima svoje parametre trenja kako bi se bolje definirala fizika interakcija te cijela mapa predstavlja granice kako bi igra bila fokusirana na određen prostor. Mapa je potpuno simetrična kako bi se stvori veći balans među igračima.

4.3.2. Protivnici

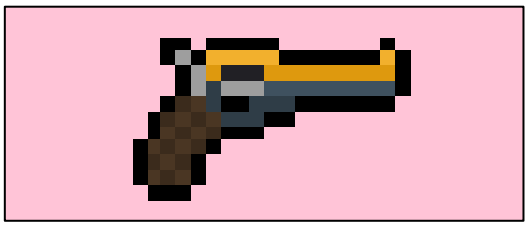
Protivnički likovi su likovi koji su pokretani od strane drugih igrača te predstavljaju neprijatelje koje treba nadjačati te pobijediti. Svi likovi imaju jednake parametre, mogućnosti upravljanja i kretanja. Osnovna verzija igre je zamišljena za dva igrača s mogućnosti nadogradnje za više igrača. Dok likovi među sobom nemaju fizičku interakciju poput sudaranja, mogu međudjelovati putem oružja i napucavanja. Svim likovima su početni životni bodovi postavljeni na 20 te međusobnim napucavanjem se životni bodovi smanjuju te kada dostignu 0 lik umire i protivnik osvaja bodove. Životni bodovi igrača su uvijek vidljivi na ekranu dok su životni bodovi protivnika vidljivi iznad protivničkog lika. Lik se oživljava nakon zadanog vremena. Igrač koji na kraju ima najviše bodova, u ovom slučaju ubojstava, pobjeđuje.

4.3.3. Oružje

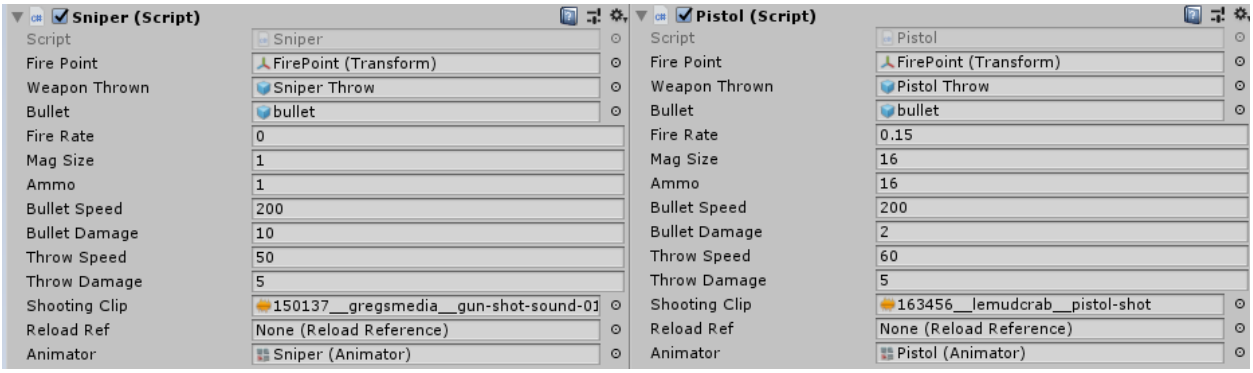
Kako bi igra bila *shooter* bitno je implementirati neku vrstu oružja. Oružjem se nanosi šteta protivniku te ga se dovodi do nula životnih bodova kako bi se ostvarili poeni. Kako većina oružja ima iste ili slične atribute napravljena je skripta *Weapon* koja sadrži sve potrebne atribute te preko koje se može modelirati potreban broj oružja s vlastitim postavkama parametara. Atributi koji su potrebni za izradu oružja su brzina pucanja, brzina metka, veličina magazina, nanošenje štete i sl. U igru su dodane dvije vrste oružja: pištolj i snajper. Dok je pištolj versatilnije oružje s kojim se može brže pucati i ima veći magazin, nanosi manje štete za razliku od snajpera koji je sporiji s jednim metkom ali nanosi puno veću štetu i teže ga je skupiti. Snajperu je također dodan laser radi lakšeg nišanjenja. Mjesta stvaranja oružja su postavljena simetrično po mapi: dva mjesta za pištolje kod početnih pozicija likova te jedno mjesto za snajper na sredini mape. Igrač u svakom trenutku može imati najviše dva oružja kod sebe u bilo kojoj kombinaciji te može mijenjati između njih. Nakon što je oružje pokupljeno nestaje na mapi te se ponovno stvara nakon određenog vremena. Metci se ispucavaju u smjeru kojem igrač nišani. Ako metak pogodi protivničkog lika nanosi mu štetu i ubija ga ako ga dovede do nula. Implementacija pucanja dodatno navedena u nastavku rada. Broj metaka u magazinu za oružje u korištenju je uvijek vidljivo igraču na ekranu te kada dosegne nulu ili pritiskom tipke R počinje zamjenjena magazina za što je napravljen indikator za prikaz vremena do kraja zamjene. Osim pucanja igrač može baciti oružje u smjeru kojem gleda te pogotkom protivnika također nanijeti štetu.



Slika 4.7 - Oružje: snajper



Slika 4.8 - Oružje: pištolj



Slika 4.9 - Postavke oružja

```
private void Aim()
{
    projectileDirection = Camera.main.ScreenToWorldPoint(Input.mousePosition) - player.transform.position;
    projectileDirection.Normalize();
    projectileRotation = Mathf.Atan2(projectileDirection.y, projectileDirection.x) * Mathf.Rad2Deg;
}

1 reference | mrabar +1, 31 days ago | 3 changes
private void InputManager()
{
    if (Input.GetButtonDown("Fire1") && (Time.time > (fireRate + lastShot)) && ammo > 0 && !reloading && ready) {
        Shoot();
        Debug.Log(Time.time);
    }
    if (Input.GetButtonDown("Throw")) {
        Throw();
    }
    if (Input.GetButtonDown("Reload") && ammo < magSize && !reloading) {
        StartCoroutine(Reload());
    }
}

1 reference | mrabar +2, 164 days ago | 3 changes
void Shoot()
{
    shootingNet.CmdShoot(firePoint.position, projectileDirection, projectileRotation);
    lastShot = Time.time;
    ammo--;
    if (ammo <= 0) {
        StartCoroutine(Reload());
    }
}

1 reference | mrabar +2, 164 days ago | 3 changes
void Throw()
{
    shootingNet.CmdThrow(firePoint.position, projectileDirection, projectileRotation);
}

3 references | mrabar +3, 31 days ago | 4 changes
IEnumerator Reload()
{
    reloadRef.Reload();
    reloading = true;
    yield return new WaitForSeconds(2);
    ammo = magSize;
    reloading = false;
}

1 reference | mrabar +1, 147 days ago | 2 changes
IEnumerator Setup()
{
    yield return new WaitForSeconds(0.14f);
    ready = true;
}
```

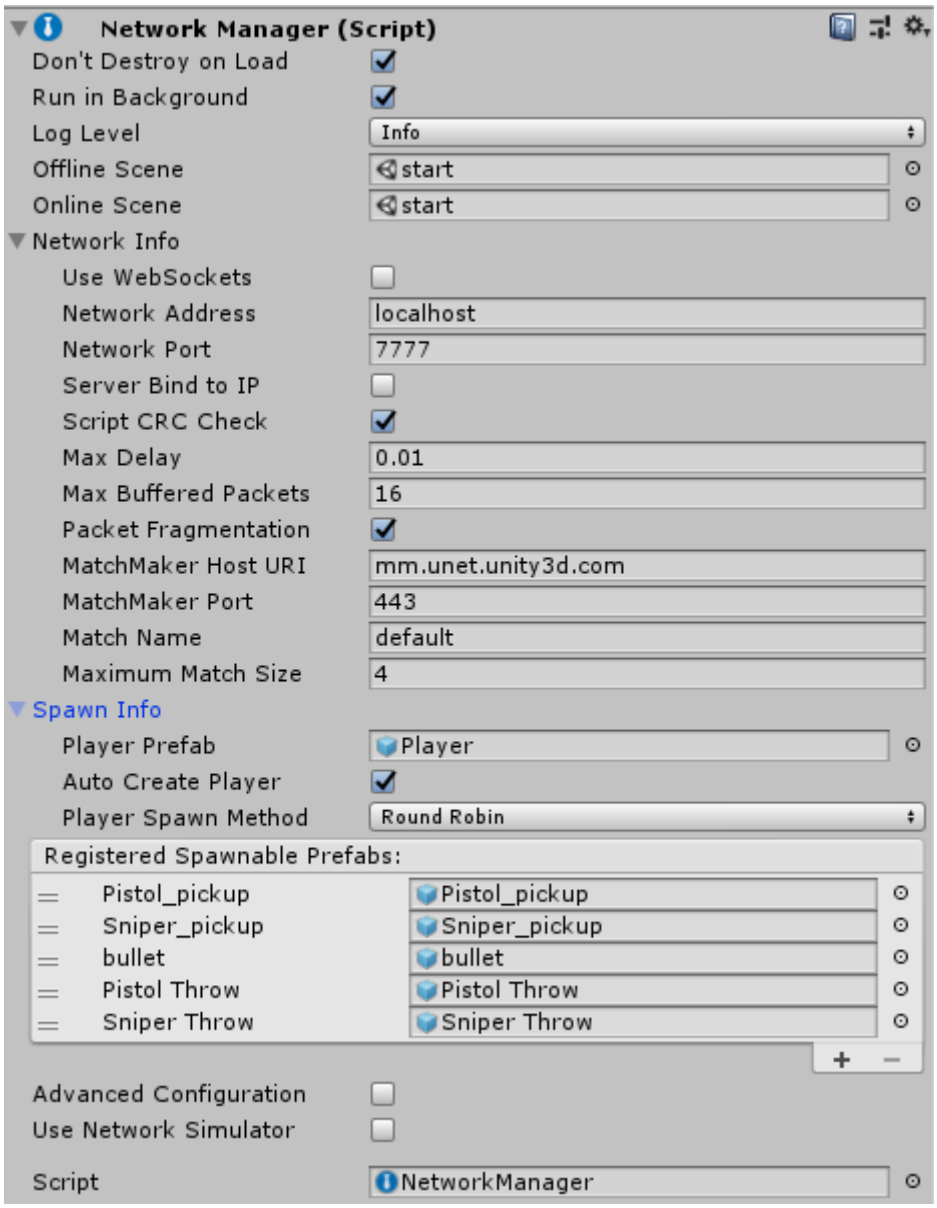
Slika 4.10 - Kontrole za upravljanje unosima za oružje

4.4. Mrežna implementacija

Kada bi se radila offline inačica igre moralo bi se paziti više igrača može pristupiti igri na istom uređaju te definirati različite unose za pojedinog igrača ili više perifernih uređaja poput kontrolera za upravljanje svakim pojedinim likom. Kod Online igre svi igrači imaju jednak način kontrola za igru i pri inicijalizaciji igre dobiju svako svoga lika koji je jednak i kojim mogu

upravljati te je bitno brinuti o tome da tra kontrola ne prelazi na tuđe likove. Također je bitno voditi računa o prisutnosti poslužitelja koji je naime zasebna inačica igre koja u sebi drži i sinkronizira stanja klijenata. U Unityu je za to zadužen Network Manager.

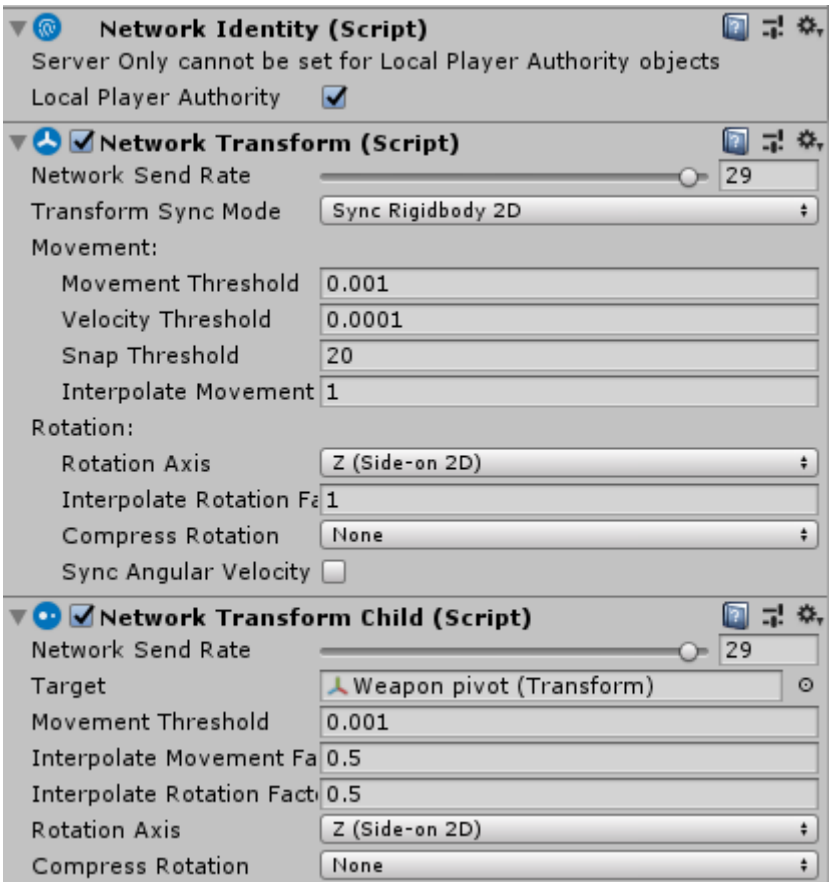
Network Manager je pristupna točka prije navedenom Unetu. Brine se kreiranju igračevog lika za svakog klijenta te se njime hosta i spaja igri. Putem njega se konfigurira *player prefab* (igračev lik), startne pozicija, svi objekti koji trebaju biti inicijalizirani i sinkronizirani na serveru te koji se stvaraju na serveru, stanje scena da li su offline ili online te naprednije stvari. U njemu su registrirani objekti poput samog igrača ali i drugi objekti koji se stvaraju od strane servera: pištolj i snajper za pokupiti, metak te pištolj i snajper kao bačeni objekti. Definiran je maksimalan broj igrača i scene te ostale opcije nisu dirane.



Slika 4.11 - Network Manager komponenta

Svi Unet mrežni objekti na sebi moraju imati pridodan Network Identity. Njime se identificira da je neki objekt isti na različitim klijentima i da ga je potrebno sinkronizirati.Na njemu definiramo tko upravlja tim objektom: poslužitelj ili klijent te tko ima autoritet. U slučaju igračevog lika lokalni igrač ima autoritet. Network Transform komponenta služi za sinkronizaciju kretanja i rotacije objekata na mreži. Na njoj se nalaze mnogi parametri kojima se može upravljati kako bi se poboljšale performanse igre te smanjio *lag* poput količine podataka koji se šalju odjednom granice kretanja nakon koje se vrši slanje. Korištena je objektu lika uz Network Transform Child komponentu koja upravlja djecom objekata poput oružja u ovom slučaju te su postavke

postavljene da optimalno sinkroniziraju likove među klijentima te bi u većim projektima trebalo razmišljati o vlastitim implementacijama interpolacije radi daljnjeg smanjenja *laga*.



Slika 4.12 - Network Identity i Network Transform komponente

4.4.1. Unity Networking

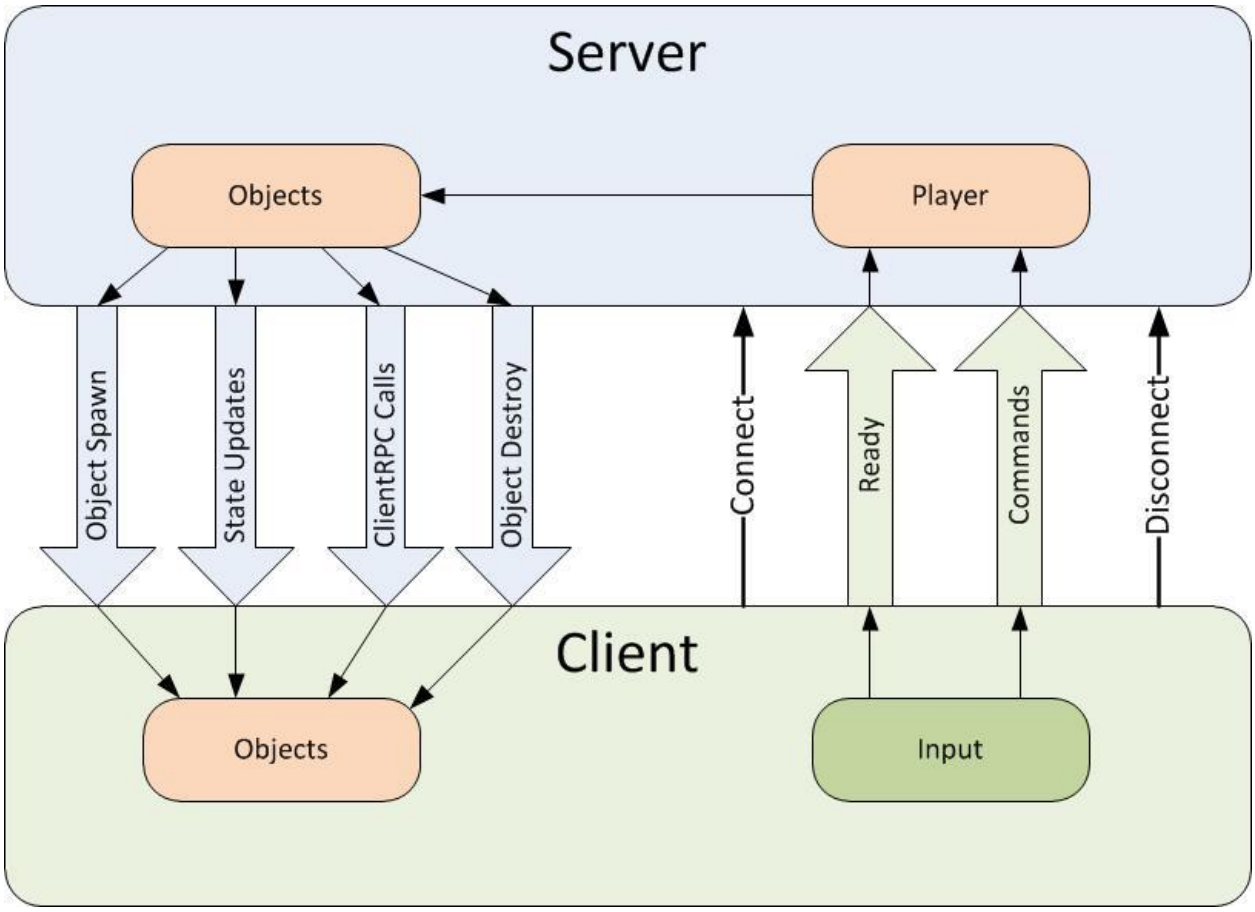
NetworkBehaviour je osnovna klasa koja se koristi kako bi komponenta znala za rad na mreži. Pomoću nje njenih potklasa u skriptama možemo imati uvid u mrežne događaje, automatsku sinkronizaciju polja, pozivati metode na serveru i/ili klijentu te znati ima li autoritet nad objektom ili ne.

Na slici 4.13 je prikazan kod koji vodi računa o životnim bodovima igrača te u njemu imamo prikazane neke koncepte koje obuhvaća NetworkBeahviour. Svojstvo isLocalPlayer govori o tome da li je objekt lokalni igrač i tako sprječava neželjeno upravljanje drugim igračima. U gornjem slučaju skripta se prekida izvoditi ako nije na lokalnom uređaju tog igrača. Slično tomu se koriste svojstvo isServer koje kazuje na to pripada li objekt serveru i svojstvo hasAuthority koje provjerava autoritet nad objektom. Anotacijom SyncVar nad poljem specificiramo da polje mora biti sinkronizirano na svim instancama igre, primjer u ovom slučaju je nad samim životnim

bodovima koji su inicijalno postavljeni na dvadeset te kad bi se ta brojka smanjila na poslužitelju za neki broj morala bi se na svim klijentima također smanjiti za isti broj. Command anotacija se koristi na funkcijama koje se pozivaju na klijentu a izvršavaju na poslužitelju. Funkcije stom anotacijom moraju počinjati slovima „Cmd“. U našem slučaju CmdDie funkcija šalje zahtjev od strane klijenta na poslužitelj da provjeri dali je igrač na nula životnih bodova te ako je poziva funkciju RpcDie. Anotacija ClientRpc se koristi na funkcijama koje se pozivaju od strane poslužitelja na svim klijentima i funkcija mora počinjati slovima „Rpc“. Nakon što poslužitelj provjeri i potvrdi da je igrač na nula životnih bodova, obavještava sve klijente pomoću RpcDie koji tada na svim inačicama igre tog igrača zabilježavaju mrtvim i uništavaju objekt te dolazi do sinkroniziranog stanja. Naredbe (command) i pozivi udaljenih postupaka (RPC) su osnovni način komunikacije među poslužiteljem i klijentima. ServerCallback i ClientCallback anotacije su jednostavniji način izvršenja koda samo na poslužitelju ili klijentu bez potrebe za provjerom isServer i isClient.

```
public class Health : NetworkBehaviour
{

    public int maxHealth = 20;
    [SyncVar]
    public int currHealth = 20;
    private PlayerNet player;

    [SerializeField]
    private PlayerStats playerStats;

    0 references | mrabar +2, 147 days ago | 3 changes
    private void Awake()
    {
        player = GetComponent<PlayerNet>();
    }

    [ServerCallback]
    0 references | mrabar +1, 147 days ago | 2 changes
    private void OnEnable()
    {
        currHealth = maxHealth;
    }

    0 references | mrabar +2, 147 days ago | 4 changes
    void Update()
    {
        if (!isLocalPlayer) {
            return;
        }
        CmdDie();
    }

    [Command]
    1 reference | mrabar +1, 147 days ago | 2 changes
    void CmdDie()
    {
        if (currHealth <= 0) {
            RpcDie();

        }
    }

    [ClientRpc]
    1 reference | mrabar +2, 53 days ago | 5 changes
    void RpcDie()
    {
        GetComponent<WeaponSync>().slot1 = 0;
        GetComponent<WeaponSync>().slot2 = 0;
        player.Die();
    }
}
```

Slika 4.13 - Funkcionalnost kontrole životnih bodova



Slika 4.14 - Klijent/pšoslužitelj komunikacija

NetworkServer.Spawn funkcija je bitna kod sinkronizacije objekata stvorenih od strane poslužitelja na klijentima. Radi tako da stvara objekt na svim klijentima s istim mrežnim identifikacijskim brojem. U ovoj igri se tako stvara oružje za pokupiti te bačeno oružje i u ranijim verzijama igre metci.

```
public class ItemSpawner : NetworkBehaviour {

    public GameObject pistol;
    public GameObject sniper;


    0 references | mrabar +1, 122 days ago | 2 changes
    public override void  OnStartServer() {
        foreach(Transform weapon in transform) {
            if(weapon.tag=="PistolPickup") {
                GameObject item = Instantiate(pistol, weapon.transform.position, Quaternion.identity) as GameObject;
                NetworkServer.Spawn(item);
            }
            if(weapon.tag=="SniperPickup") {
                GameObject item = Instantiate(sniper, weapon.transform.position, Quaternion.identity) as GameObject;
                NetworkServer.Spawn(item);
            }
        }
    }
}
```

Slika 4.15 - NetworkServer-Spawn funkcionalnost unutar klase ItemSpawner

Pucanje, osnovni dio *shooting* žanra, je originalno zamišljeno da kada igrač klikne mišem i time pokrene pucanje da se metak stvori sa strane poslužitelja pomoću NetworkServer.Spawn metode te ga ispucava u određenom smjeru nakon čega metak na poslužitelju registrira da li je došlo do kolizije s igračem te nakon toga oduzimati životne bodove i sinkronizirati to na svim klijentima. Poslužitelj bi trebao sinkronizirati sve pozicije metka tijekom leta kako bi na svim klijentima ispucavanje metka izgledalo glatko i bez poteškoća. Zbog velike brzine ispucavanja i leta metka te drugih faktora poput rada pod utjecajem gravitacije i brzine same igre, prevelik broj podataka je potrebno slati u svakom trenutku te su dobiveni su loši rezultati: putanja metka nije bila dobro sinkronizirana na svim klijentima i metci bi letjeli na sve strane te bi metci nekad zbog svoje brzine proletjeli kroz protivnike bez da stignu očitati koliziju. Zbog toga je novo pucanje u igri „lažirano“ *hitscan* metodom. *Hitscan* je često korišten u igrama žanra pucačine, gdje program detektira u kojem smjeru nišani oružje, te ispucava zraku (ray) u tom smjeru na ograničenu udaljenost navedenu u programu. Ako zraka nešto pogodi zabilježava te se klijenti sinkroniziraju. Ovaj način rada je puno efikasniji jer nema potrebe za sinkronizacijom objekta u kretnji već se sve dogodi u jednom trenutku te se dobivaju bolji rezultati a zbog velike brzine igre se ne primjećuje „nerealnost“. Na slici dole vidimo korištenje Physics2D.Raycast funkcije koja se izvršava na poslužitelju te izračunava da li se nešto nalazi u smjeru nišanjenja na udaljenosti od 100. Rezultati su odmah zabilježeni te se poziva Rpcshoot na svim klijentima koji obavještava o rezultatu pucanja i svi klijenti znaju novo stanje. Za kraj je dodana funkcija BulletTrail koja iscrtava crtu na klijentima na djelić sekunde te time simulira putanju metka i daje bolji osjećaj igri.

```
[Command]
1 reference | mrabar +2, 53 days ago | 5 changes
public void CmdShoot(Vector2 firePoint, Vector2 direction, float rotation)
{
    RaycastHit2D hit = Physics2D.Raycast(firePoint, direction, 100);

    if (hit.transform.gameObject.tag == "Player") {
        if (hit.transform.gameObject.GetComponent<Health>().currHealth <= bulletDamage) {
            playerStats.kills++;
            hit.transform.gameObject.GetComponent<PlayerStats>().deaths--;
        }
        hit.transform.gameObject.GetComponent<Health>().currHealth -= bulletDamage;
    }
    if (hit.collider == null) {
        RpcShoot(firePoint, direction, hit.point, false);
    } else {
        RpcShoot(firePoint, direction, hit.point, true);
    }
}

[ClientRpc]
2 references | mrabar, 31 days ago | 3 changes
private void RpcShoot(Vector2 firePoint, Vector2 direction, Vector2 hitPoint, bool targetHit)
{
    AudioSource.PlayClipAtPoint(weapon.shootingClip, firePoint);
    weapon.animator.Play("Recoil");
    if (targetHit) {
        lineRenderer.SetPosition(0, firePoint);
        lineRenderer.SetPosition(1, hitPoint);
    } else {
        lineRenderer.SetPosition(0, firePoint);
        lineRenderer.SetPosition(1, firePoint + direction * 100);
    }
    StartCoroutine(BulletTrail());
}

1 reference | mrabar, 130 days ago | 1 change
IEnumerator BulletTrail()
{
    lineRenderer.enabled = true;
    yield return new WaitForSeconds(0.02f);
    lineRenderer.enabled = false;
}
```

Slika 4.16 - Funkcionalnost pucanja

Izbacivanje oružja ima sličnu funkcionalnost kao originalno pucanje no zbog malih brzina projektila nije bilo potrebe za mijenjanjem funkcionalnosti.

```
[Command]
1 reference | mrabar +2, 38 days ago | 4 changes
public void CmdThrow(Vector2 firePoint, Vector2 direction, float rotation)
{
    if (weaponSync.weapon1 != null && weaponSync.weapon1.activeSelf) {
        weaponSync.slot1 = 0;
    } else if (weaponSync.weapon2 != null && weaponSync.weapon2.activeSelf) {
        weaponSync.slot2 = 0;
    }

    GameObject thrown = Instantiate(weaponThrown, firePoint, Quaternion.Euler(0, 0, rotation)) as GameObject;
    Physics2D.IgnoreCollision(thrown.GetComponent<Collider2D>(), GetComponent<Collider2D>());
    thrown.GetComponent<Rigidbody2D>().velocity = direction * throwSpeed;
    thrown.GetComponent<Projectile>().damage = throwDamage;
    if (rotation > 90 || rotation < -90) {
        thrown.transform.localScale = new Vector2(thrown.transform.localScale.x, thrown.transform.localScale.y * -1);
    }
    NetworkServer.Spawn(thrown);
    Destroy(thrown, 1);
    RpcThrow(firePoint);
}

[ClientRpc]
1 reference | mrabar, 38 days ago | 1 change
private void RpcThrow(Vector2 firePoint){
    AudioSource.PlayClipAtPoint(thrownClip, firePoint);
}
```

Slika 4.17 - Funkcionalnost bacanja oružja

```
public class Projectile : NetworkBehaviour {

    public int damage;

    0 references | mrabar +2, 164 days ago | 4 changes
    private void OnTriggerEnter2D(Collider2D coll)
    {
        if (!isServer) {
            return;
        }

        if (coll.tag == "Player") {
            coll.GetComponent<Health>().currHealth -= damage;
            Destroy(gameObject);
        } else {
            Destroy(gameObject);
        }
    }
}
```

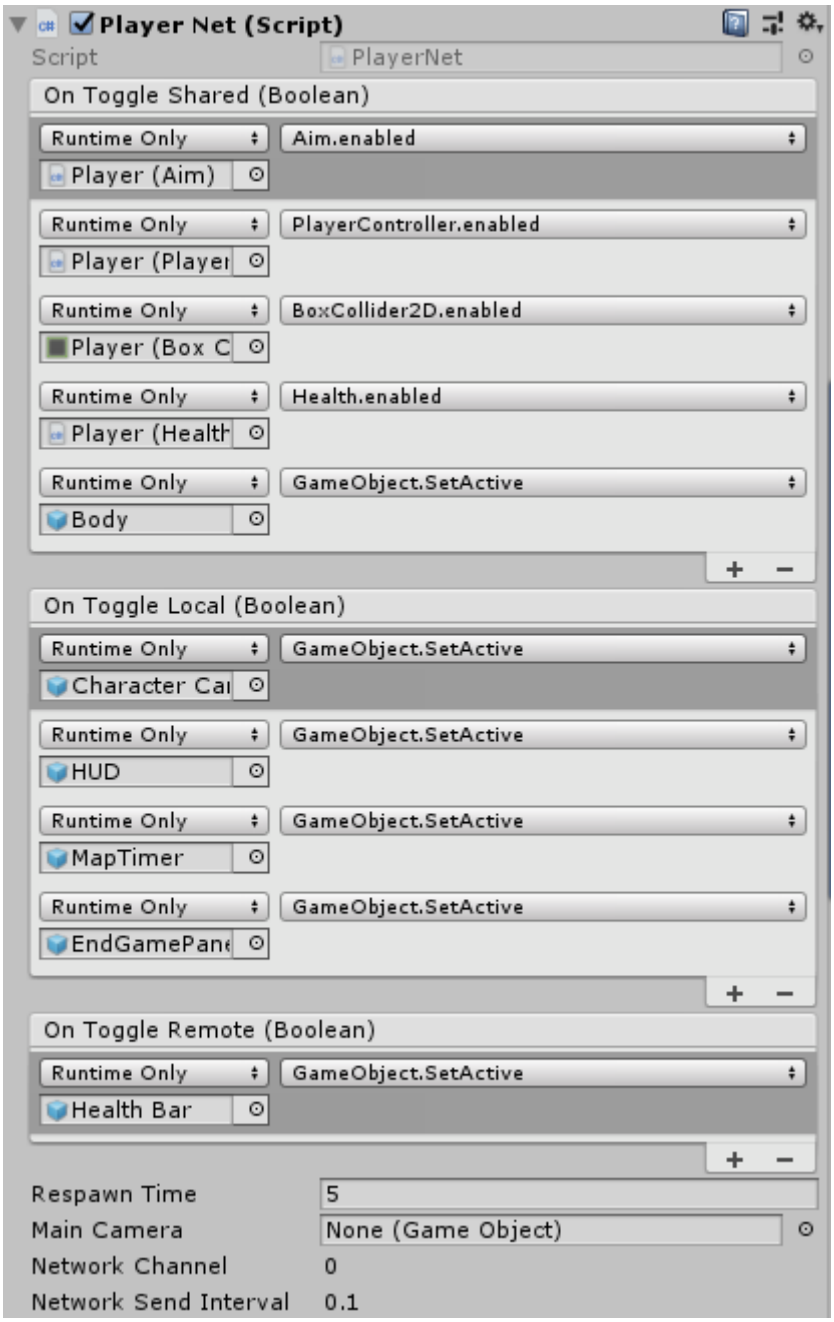
Slika 4.18 - Funkcionalnost bačenog projektila

Za kraj, kako su svi igrači likovi zapravo instance istog objekta s velikim brojem funkcionalnosti koje obuvaćaju i lokalnog lika i neprijateljeve likove moramo moći definirati koje komponente želimo pokretati samo na lokalnom klijentu a koje na ostalima ili svima. Npr. likovima tuđih klijenta želimo imati aktiviran Health Bar koji se prikazuje iznad neprijateljskih likova dok na

lokalnom želimo imati health bar koji je prikazan u samom HUDu. Lokalnost provjeravamo ponovno pomoću svojstva isLocalPlayer te na temelju toga aktiviramo i deaktiviramo komponente po potrebi.

```
[SerializeField]
ToggleEvent onToggleShared;
[SerializeField]
ToggleEvent onToggleLocal;
[SerializeField]
ToggleEvent onToggleRemote;
[SerializeField]
float respawnTime = 5f;

public GameObject mainCamera;


0 references | mrabar +1, 147 days ago | 2 changes
void Start()
{
    mainCamera = Camera.main.gameObject;
    EnablePlayer();
}

1 reference | mrabar +2, 147 days ago | 4 changes
void DisablePlayer()
{
    if (isLocalPlayer) {
        mainCamera.SetActive(true);

    }
    onToggleShared.Invoke(false);

    if (isLocalPlayer)
        onToggleLocal.Invoke(false);
    else
        onToggleRemote.Invoke(false);
}

2 references | mrabar +2, 147 days ago | 4 changes
void EnablePlayer()
{
    GetComponent<Rigidbody2D>().velocity = new Vector2(0, 0);
    if (isLocalPlayer) {
        mainCamera.SetActive(false);
    }
    onToggleShared.Invoke(true);

    if (isLocalPlayer)
        onToggleLocal.Invoke(true);
    else {
        onToggleRemote.Invoke(true);
        name = "Dummy";
    }
}
```

Slika 4.19 - Klasa za upravljanje komponentama lokalnih i udaljenih igračih likova

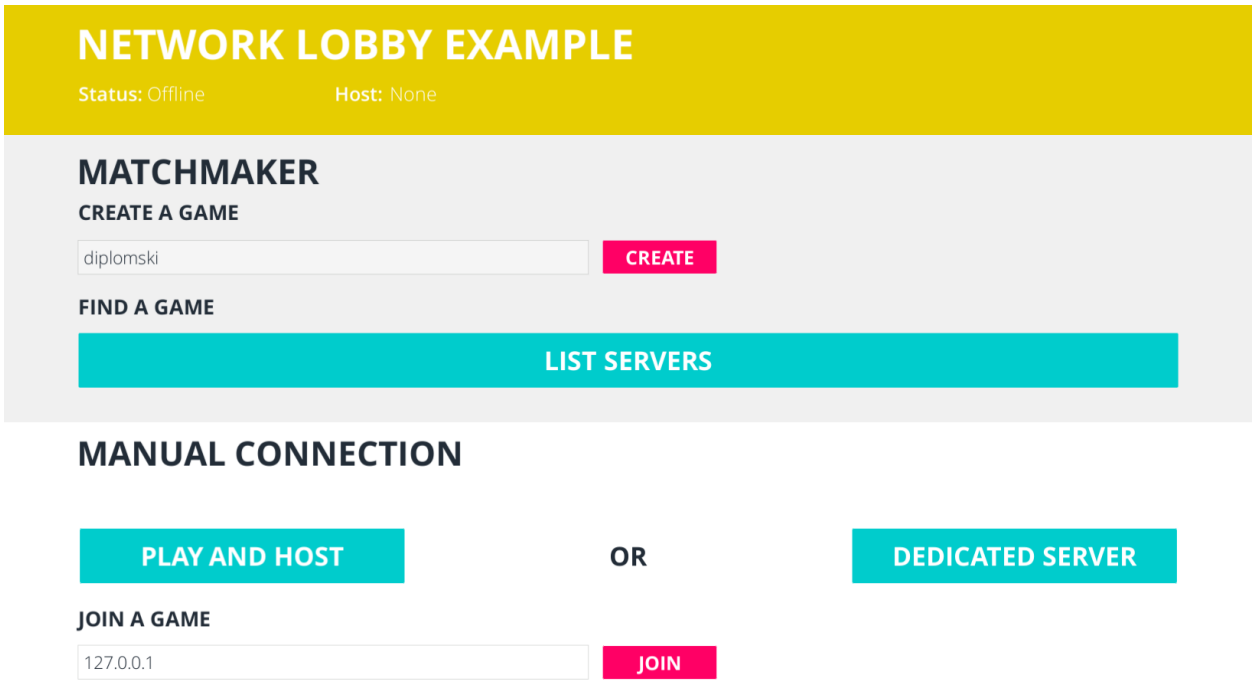


Slika 4.20 - Postavke za upravljanje lokalnim i udaljenim komponentama

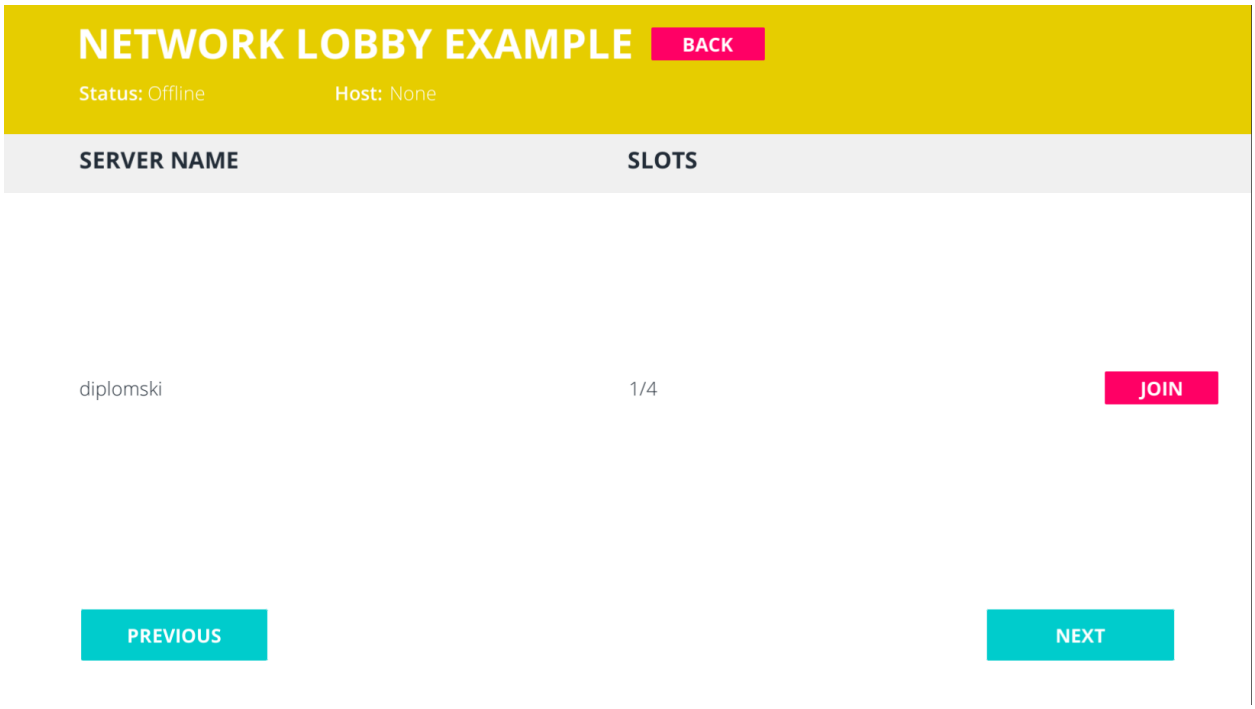
4.4.2. Player Lobby

Player lobby ili predvorje je scena u igri prije same igre. Funkcionalnost scene je dana od strane Unitya te koristi funkcionalnosti Network Managera za svoj rad. Predvorje je dio igre di igrači mogu hostati poslužitelj te klijenti mogu pronaći postojeće poslužitelje i pridružiti se. To je mjesto di se igrači sreću i zajedno ulaze u novu igraču sesiju. U *lobbyu* je moguće pokrenuti poslužitelj kao namjenski te s drugih uređaja se pridružiti isključivo kao klijent ili napraviti

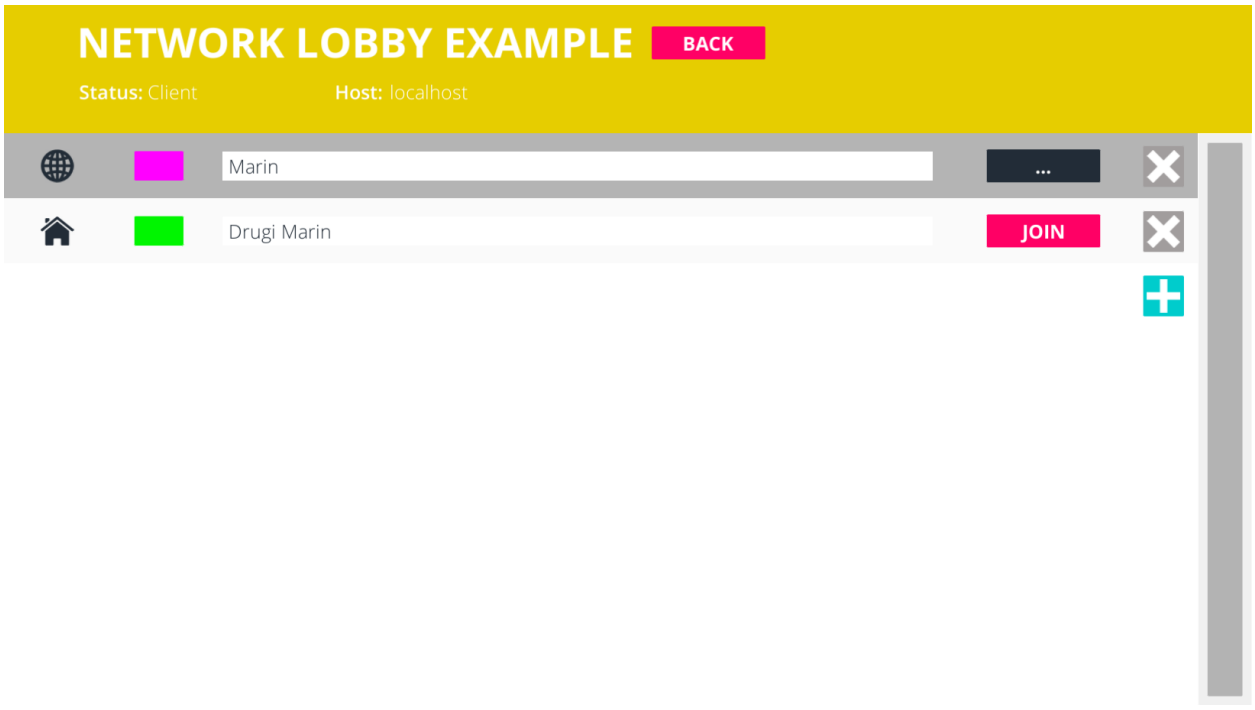
lokalni poslužitelj paralelno sa spajanjem u obliku klijenta. Pri hostanju, poslužitelju je moguće dati ime. Kao klijent je moguće pronaći sve kreirane poslužitelje s prikazanim imenima te se pridružiti nekom od njih ili se direktno pridružiti putem IPa. Nakon što su se svi igrači skupili u predvorju mogu si pridodati svoje ime odabrati boju koja ih reprezentira te pokrenuti igru kada su svi spremni.



Slika 4.21 - Network lobby opcije



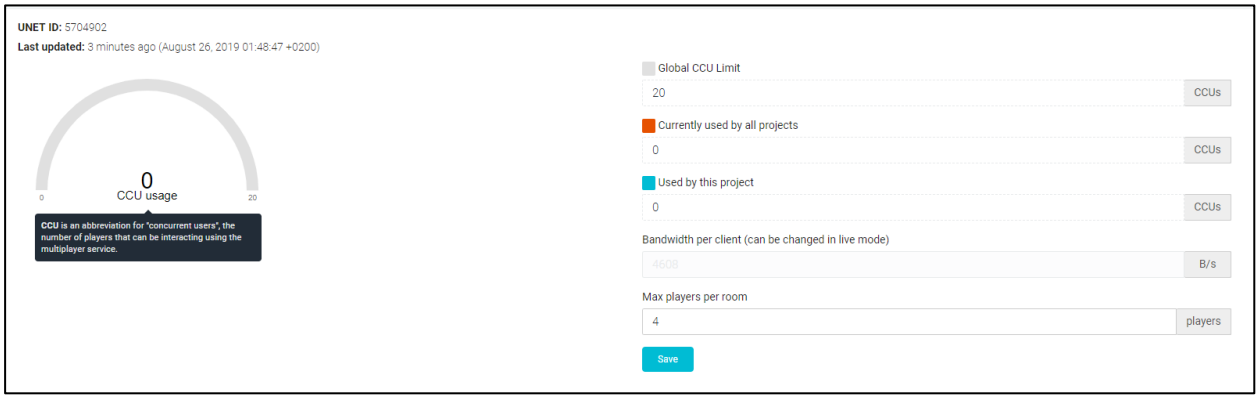
Slika 4.22 - Network lobby pretraživanje poslužitelja



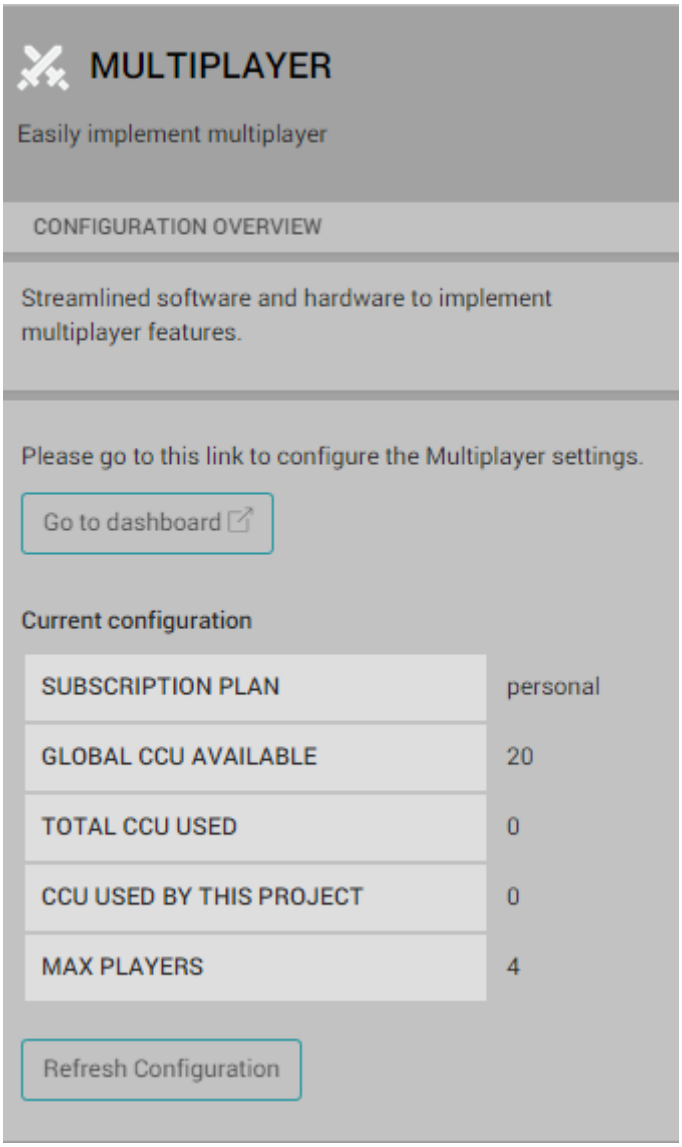
Slika 4.23 - Predvorje

4.4.3. Unity Multiplayer servis

Kako bi se igrači uopće mogli pronaći te kako bi cijela mrežna implementacija bila funkcionalna potrebno je aktivirati Unity Multiplayer servis. Unutar Unitya na service tabu je potrebno aktivirati Multiplayer opciju koja nas vodi na Unity dashboard gdje možemo namjestiti postavke za maksimalan broj igrača po poslužitelju i vidjeti neke druge informacije kao i prilagoditi postavke u igri u izdanoj verziji kao npr. maksimalan broj poslanih paketa. Nakon par klikova aplikacija je spremna za umreženu igru.



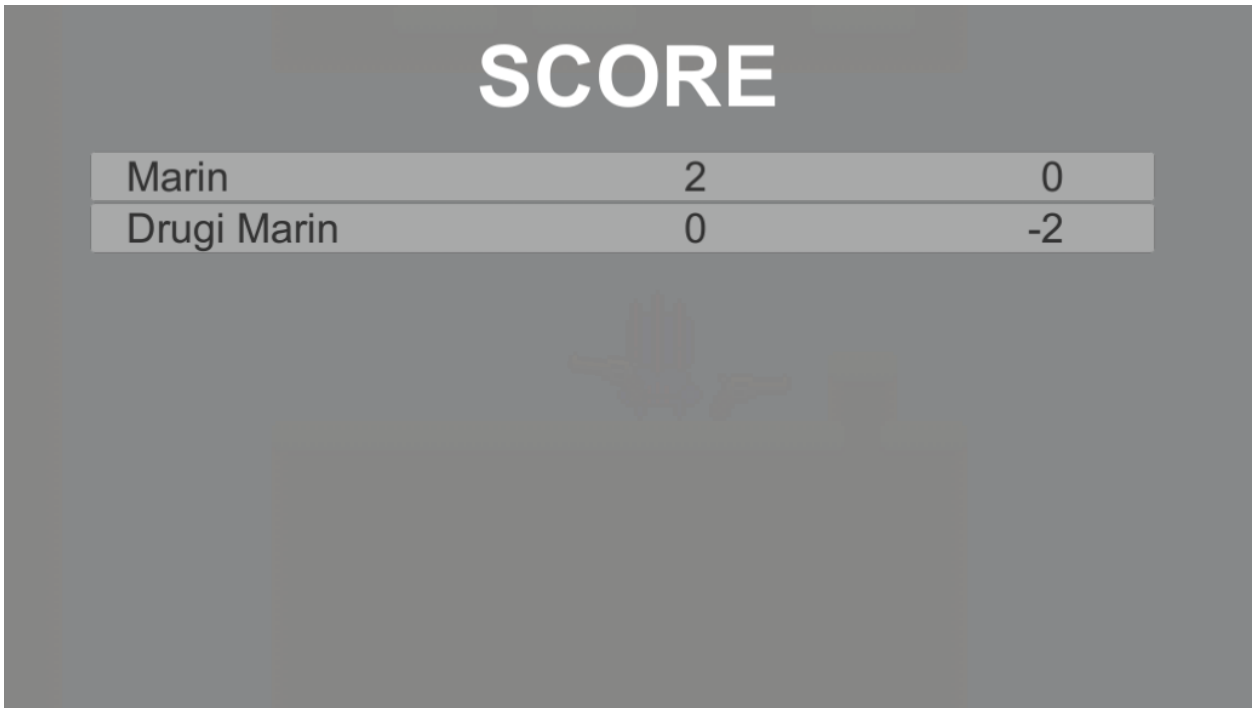
Slika 4.24 - Unity multiplayer postavke na dashboardu



Slika 4.25 - Unity Multiplayer postavke unutar Unitya

4.5. Natjecateljska igra

Kako je igra zamišljena da bude natjecateljskog duha potrebno je imati neku vrstu parametra koji su može uspoređivati kako bi se vidjelo tko je bolji. Nakon pokretanja igre, svaki igrač iz predvorja je pod svojim imenom dodan u tablicu te je napravljena funkcionalnost za zabilježavanje željenih podataka koja služi kao baza podataka. Pošto je igra dosta mala dovoljno nam je skupljati podatke o ubojstvima i smrtima. Bod u ubojstvo se dobiva za svako dovođenje neprijateljevih životnih bodova na nula dok se bod u smti dobiva za svaku vlastitu smrt nanesenu ili od neprijatelja ili padanjem u rupu. Trenutno stanje bodova je moguće vidjeti u svakom trenutku putem tipke 'tab' koja prikazuje tablicu s imenima igrača pored kojih se nalaze ubojstva i smrti te je poredana po ubojstvima od najviše prema manje.



Slika 4.26 - Ploča s rezultatima

4.5.1. Pobjeda

Trajanje jedne igrače sesije je postavljeno na četiri minute te se nakon isteka vremena pregledavaju rezultati te se odabiru prva tri mjesta na temelju ubojstva i smrti. Pobjednik je onaj s najviše ubojstva i najmanje smrti. Igra nakon toga završava i podatci se brišu.

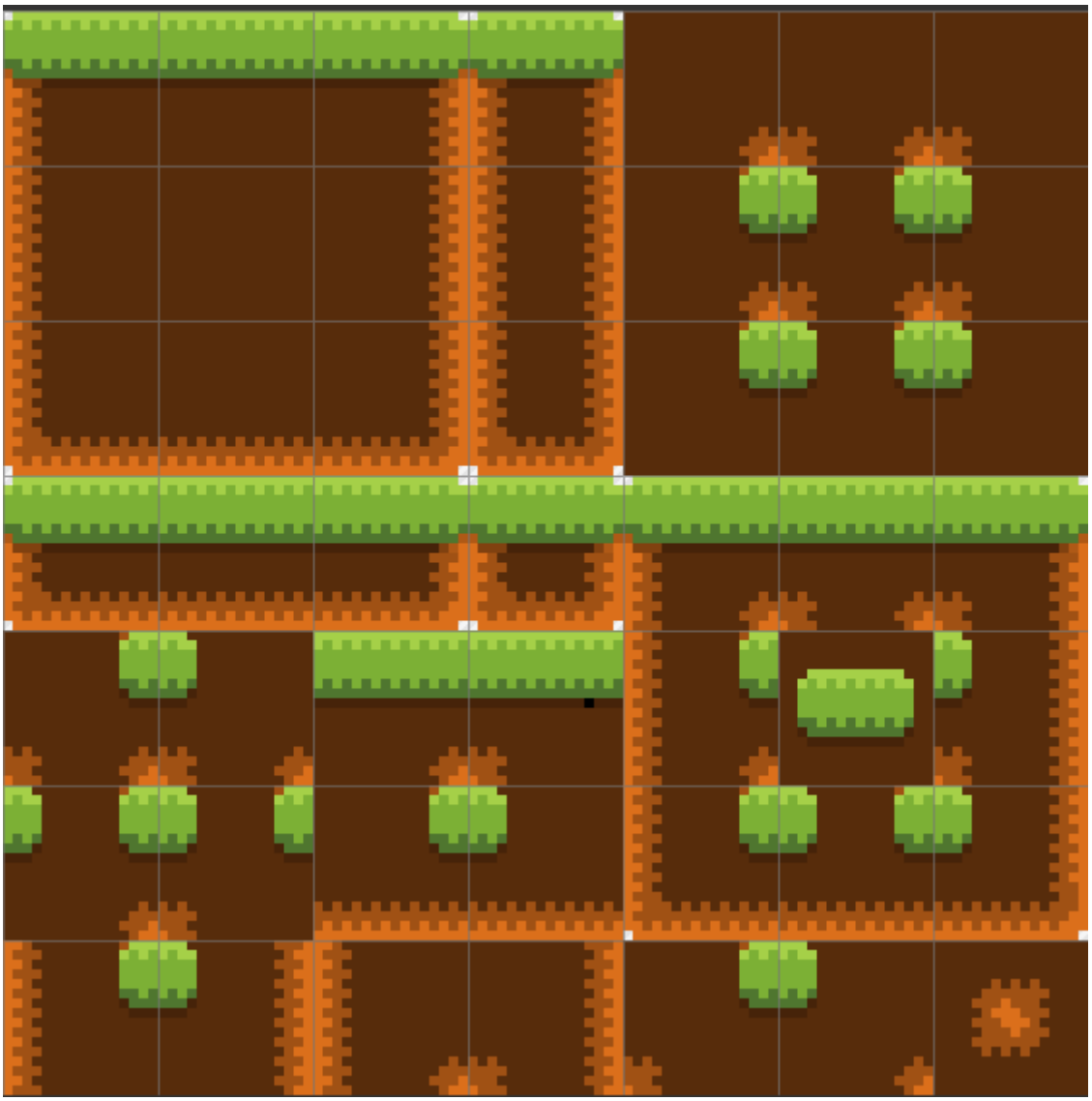
Slika 4.27 - Prikaz pobjednika na kraju igre

4.6. Grafika

Kako bi igra sveukupno malo bolje izgledala dodana je jednostavna 2D grafika. Većina grafike je izrađena pomoću alata Pyxel edit za izradu Pixel art grafike jer je odabran taj stil za igru.

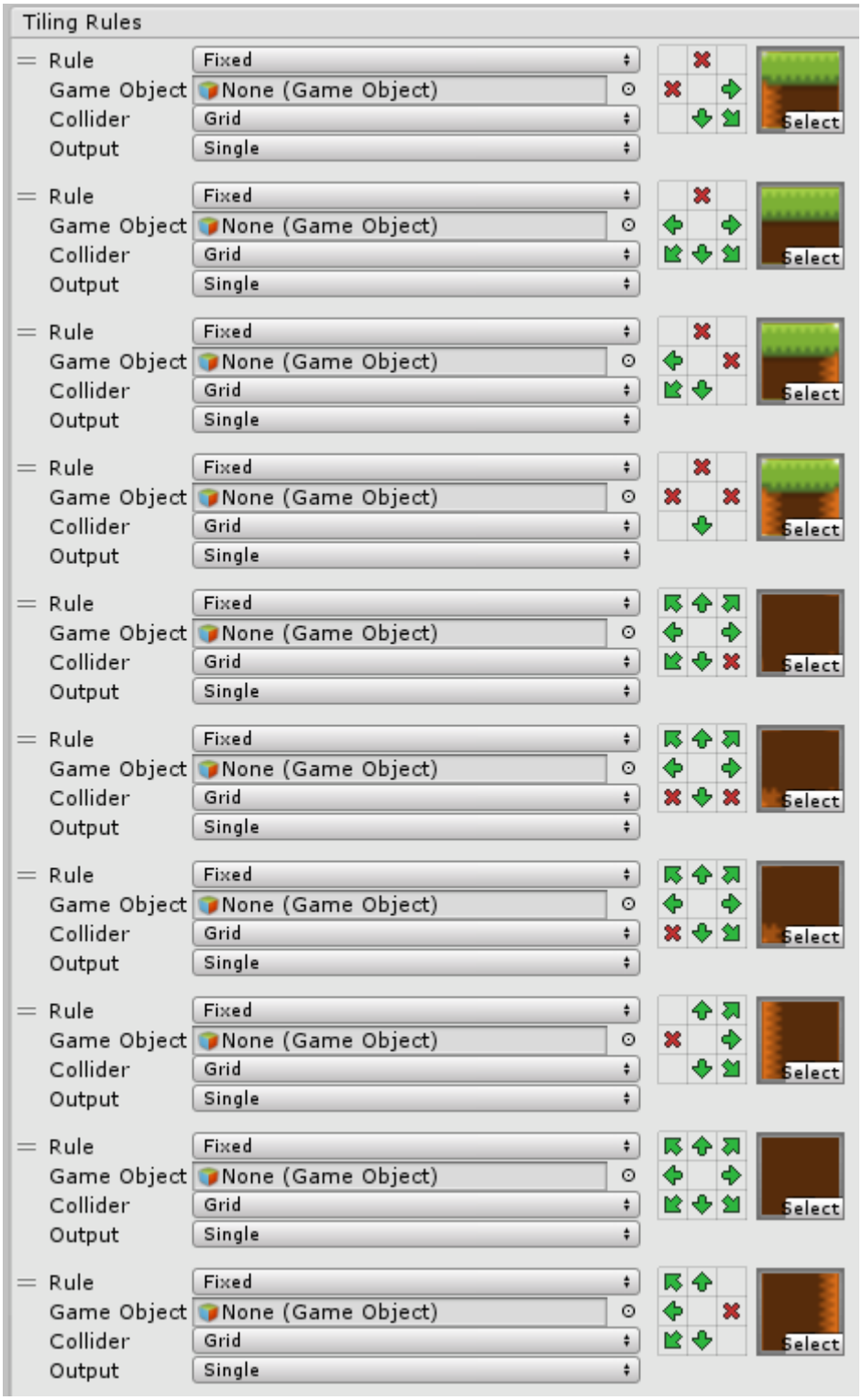
4.6.1. Mapa

Za izradu mape korištena je *tiling* metoda. Napravljeno je četrdesetosam (48) različitih *spriteova* istih dimenzija koji predstavljaju pločice (tileove) te se međusobno slažu po određenim pravilima kako bi stvorili mapu. Svaka pločica se može u svakom smjeru samo s određenim drugim pločicama spajati kako ne bi došlo do neprirodnog izgleda mape. Kod izrade svake pločice treba paziti na to hoće li se sa svake strane pločice u svih 8 strana dodavati druga pločica ili ne. Za izradu svih kombinacija potrebno je četrdesetsedam (47) pločica.



Slika 4.28 - Tileset za izradu mape

Unutar Unitya pomoću alata za *tiling* definiramo pravila za kada koja pločica treba biti iscrtana tako da ne moramo sami postavljati uvijek ispravnu pločicu već pomoću istog objekta dok ga postavljamo po sceni program na temelju zadanih pravila postavlja pločicu koja paše na to mjesto. Pravila se odnose na to da li pored postavljene pločice se nalazi druga pločica ili praznina u svih 8 smjerova.

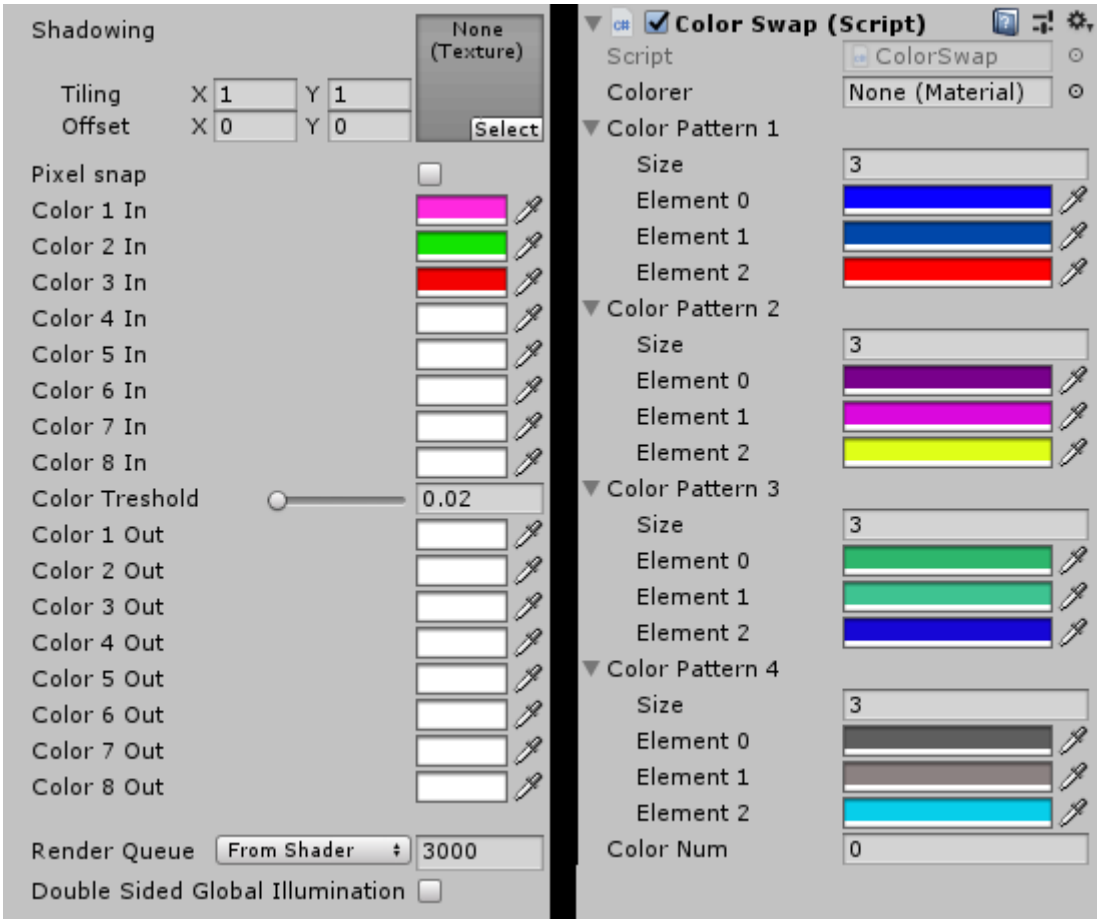


Slika 4.29 - Pravila za odnos između pločica

4.6.2. Likovi

Kako više igrača upravlja istovremeno svojim likovima bitno je da se likovi razlikuju kako ne bi došlo do zbunjenosti igrača. Odlučeno je da će se likovi razlikovati u paleti boja. Dok je osnovni oblik lika napravljen u Pyxel editu te su se i sve varijante palete mogle izvesti na isti način i

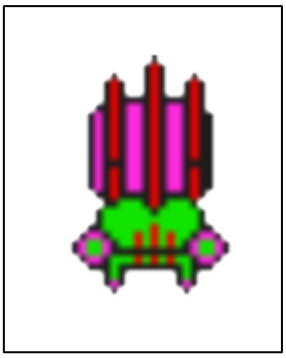
ubaciti u igru, odlučeno je koristiti se shaderima kako bi se istražila ta tehnologija. Shader je program koji se vrti direktno na GPU kao već navedeno te ga mi koristimo kako bi manipulirali izlaznim bojama pojedinih Pyxela. Osnovni oblik lika je obojan u žestoke međusobno veoma različite boje kako se ne bi miješale te je u shaderu definirana svaka od tih boja kao ulazna vrijednost te četiri palete boja kao izlazna vrijednost gdje se paleta boja za igrača odabire u ovisnosti o boji odabranoj u predvorju. Svaka originalna boja je u odnosu 1:1 s bojama iz paleta. Kada igra pokrenuta GPU dok iscrtava igru na zaslon pretražuje na objektu ulaznu vrijednost boje te ju iscrtava izlaznom vrijednosti.



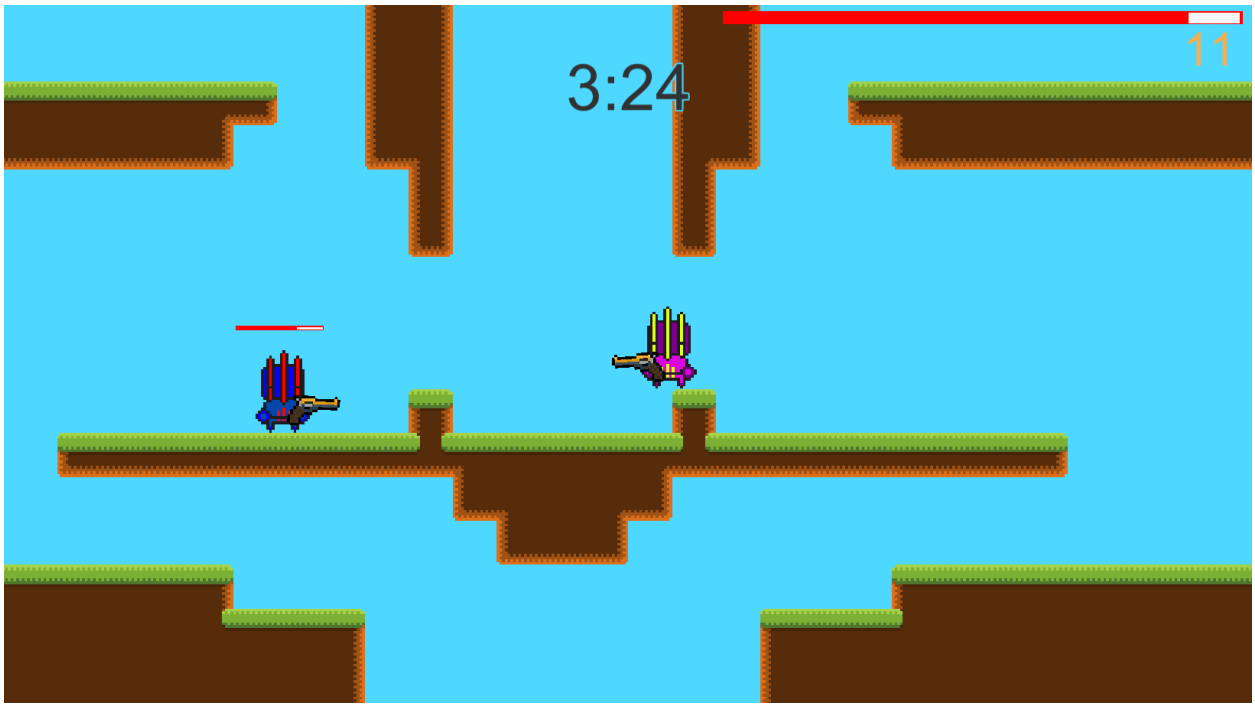
Slika 4.30 - Ulazna i izlazne palete za bojanje likova

```
fixed4 frag(v2f i) : COLOR
{
    half4 col = tex2D(_MainTex, i.uv1.xy);

    col = all(abs(col.r - _Color1in.r)
    <= _ColorTresh && abs(col.g - _Color1in.g)
    <= _ColorTresh && abs(col.b - _Color1in.b)
    <= _ColorTresh) ? _Color1out : col;

    col = all(abs(col.r - _Color2in.r)
    <= _ColorTresh && abs(col.g - _Color2in.g)
    <= _ColorTresh && abs(col.b - _Color2in.b)
    <= _ColorTresh) ? _Color2out : col;

    col = all(abs(col.r - _Color3in.r)
    <= _ColorTresh && abs(col.g - _Color3in.g)
    <= _ColorTresh && abs(col.b - _Color3in.b)
    <= _ColorTresh) ? _Color3out : col;

    col = all(abs(col.r - _Color4in.r)
    <= _ColorTresh && abs(col.g - _Color4in.g)
    <= _ColorTresh && abs(col.b - _Color4in.b)
    <= _ColorTresh) ? _Color4out : col;

    col = all(abs(col.r - _Color5in.r)
    <= _ColorTresh && abs(col.g - _Color5in.g)
    <= _ColorTresh && abs(col.b - _Color5in.b)
    <= _ColorTresh) ? _Color5out : col;

    col = all(abs(col.r - _Color6in.r)
    <= _ColorTresh && abs(col.g - _Color6in.g)
    <= _ColorTresh && abs(col.b - _Color6in.b)
    <= _ColorTresh) ? _Color6out : col;

    col = all(abs(col.r - _Color7in.r)
    <= _ColorTresh && abs(col.g - _Color7in.g)
    <= _ColorTresh && abs(col.b - _Color7in.b)
    <= _ColorTresh) ? _Color7out : col;

    col = all(abs(col.r - _Color8in.r)
    <= _ColorTresh && abs(col.g - _Color8in.g)
    <= _ColorTresh && abs(col.b - _Color8in.b)
    <= _ColorTresh) ? _Color8out : col;

    half4 mask = tex2D(_MaskTex, i.uv2.xy);
    col.rgb = col.rgb*mask.rgb;
    return col * i.color;
}
```

Slika 4.31 - Dio shader skripte za bojanje likova



Slika 4.32 - Osnovni izgled lika ulaznom paletom boja



Slika 4.33 - Konačni izgled igre

4.7. Posljednji detalji i build

Za kraj su u igru dodane odskoka oružja kod pucanja, glazba i zvučni efekti za neke kretnje i pucanje kako igra ne bi bila monotona. Igri je dan naziv Boulette (metak na francuskom) te se time iz imena može vidjeti da se možda radi o pucačini. Igra je time završena te je napravljen build za Windows platformu putem Unitya koji je zatim uploadan na stranicu Itch.io. Igra je dostupna na Itch.io stranici pod nazivom Boulette na adresi https://reybar.itch.io/boulette. Igru je potrebno downloadati kako bi se igrala.

5. ZAKLJUČAK

Cilj ovog diplomskog rada je bio napraviti online natjecateljsku igru za više igrača gdje svi igrači imaju jednake mogućnosti. U radu su istraženi načini i tehnologije za izradu online natjecateljske igre za više igrača. Za izradu igre je korišteno Unity razvojno okruženje te rješenja za online implementaciju koji isti nudi. Unet je Unityevo rješenje klijent/poslužitelj arhitekture za umrežavanje te putem visoko razinskog API-a (HLAPI) nudi širok spektar kontrola koje pokrivaju osnovne i napredne potrebe za izradu online igre. Osnovna prepreka izrade projekta bile je upoznavanje s radom klijenta i poslužitelja, kako komunicirati između njih, njihova sinkronizacija te koje komponente je potrebno pokretati na poslužitelju, koje na lokalnom a koje na udaljenom klijentu. Tijekom izrade projekta je izučeno da izrada online igre nije samo izrada offline inačice i dodavanje online komponenti na kraju već je od početka izrade potrebno imati na umu koji je cilj te koristiti Unet komponente i slagati Unity projekt po pravilima. Kako se radi o natjecateljskoj igri, bilo je bitno da svi igrači imaju jednake šanse za pobjedom te da su međusobno balansirani. Najjednostavniji način za postizanje toga je bio da su svi igrači isti te da imaju jednake mogućnosti. Tomu u korist je mapa izrađena u potpunoj simetriji te je oružje isto tako postavljeno po njoj. Napravljen je sistem za sakupljanje broja ubojstava i smrti kako bi se moglo prikazati tko je bolji igrač na kraju sesije. Osim izrade samog projekta bilo je potrebno omogućiti da je igra negdje dostupna te da se igrači mogu u njoj pronaći i započeti igraču sesiju. Za mogućnost hostanja i pronalaženja poslužitelja je korišten Unity Multiplayer servis koji tijekom razvoja projekta dozvoljava istovremeno korištenje od dvadeset igrača. Igra je postavljena na itch.io stranicu te je tamo dostupna svima na korištenje. Dok je ovime pokazano osnovno rješenje projekta, postoje mnogi dijelovi koji bi se u daljnjoj izradi mogli poboljšati. Počevši od dodatnog rada na sinkronizaciji i ublažavanju među klijentima, preko razrade samog dizajna igre, izrade više mogućnosti oružja, mapa, prepreka te razrade grafike, sve do pribavljanja namjenskih poslužitelja radi stabilnosti i pouzdanosti.

LITERATURA

[1] Unity - https://unity3d.com/unity

[2] Unity multiplayer - https://unity3d.com/unity/features/multiplayer

[3] Unet - https://www.youtube.com/watch?v=Z3f0CYArn34

[4] HLAPI - https://docs.unity3d.com/Manual/UNetUsingHLAPI.html

[5] Shader - https://unity3d.com/de/learn/tutorials/topics/graphics/gentle-introduction-shaders

[6] Game server - https://en.wikipedia.org/wiki/Game_server

[7] Network lobby - https://assetstore.unity.com/packages/essentials/network-lobby-41836

[8] Tiling - https://www.youtube.com/watch?v=KGrjZQ2qUoo

SAŽETAK

U ovom diplomskom radu bilo je potrebno proučiti i izraditi online natjecateljsku igru za više igrača s mogućnosti rangiranja. Za izradu projekta je korišten Unity game engine te njegova Unet tehnologija za mrežnu implementaciju. Istražen je način rada game servera te klijent/poslužitelj mrežne arhitekture. U radu je opisan razvoj cijele igre. Objašnjeno je kako radi lik te kako igrač njime upravlja te kako je izrađena okolina, od mape do protivnika i kako ona utječe na lika. Opisan je rad s Unet komponentama te kako se pomoću njih igra radi an mreži u odnosu na druge klijente i poslužitelj. Kroz implementaciju player lobbya i korištenjem Unity Multiplayer servisa omogućeno je igračima hostati i pronalaziti poslužitelje te započinjati igraču sesiju. Objašnjeno je prikupljanje podataka u svrhu rangiranja te odabiranje pobjednika na temelju istih. Na kraju je prikazana izrada grafike i njena implementacija te postavljanje igre na internet putem itch.io servisa kako bi bila svima dostupna.

ABSTRACT

In this thesis, it was necessary to study and create an online multiplayer competitive game with a ranking system. The Unity game engine and its Unet technology for network implementation were used to create the project. The game server and the client / server network architecture are explored in this as a basis for the work. The paper describes the development of the entire game. It explains how the character works and how the player controls it, how the environment is created, from the map to the opponent, and how it affects the character. It describes how to work with Unet components and how the game client works on the network in comparison to other clients and the server. Through the implementation of the player lobby and the use of the Unity Multiplayer service, players are allowed to host and find servers and start a game session. Collecting data for ranking purposes and selecting winners based on them is explained. Finally, the development of graphics and its implementation, as well as the placement of the game on the Internet via the itch.io service, are presented to make it accessible to all.

Keywords: Unity, Unet, client, server, player lobby, Unity Multiplayer, ranking, winners, itch.io

ŽIVOTOPIS

Marin Rabar rođen je 27.04.1994 u Osijeku. Od 2001. do 2009. pohađa OŠ Frana Krste Frankopana. Nakon toga je upisuje Gaudeamus, prvu privatnu srednju školu u Osijeku s pravom javnosti koju je završava 2013. godine. Iste godine redovno upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. 2016. godine završava preddiplomski studij te na istom fakultetu upisuje diplomski studij kao redovan student na kojem i danas studira.