Izrada demonstracijske videoigre koristeći Fusion programsku biblioteku

Matanović, Filip

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:434362

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 443

IZRADA DEMONSTRACIJSKE VIDEOIGRE KORISTEĆI FUSION PROGRAMSKU BIBLIOTEKU

Filip Matanović

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 443

IZRADA DEMONSTRACIJSKE VIDEOIGRE KORISTEĆI FUSION PROGRAMSKU BIBLIOTEKU

Filip Matanović

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 443

Pristupnik:	**Filip Matanović (0036508118)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Mirko Sužnjević

Zadatak: **Izrada demonstracijske videoigre koristeći Fusion programsku biblioteku**

Opis zadatka:

Videoigre za više igrača danas predstavljaju sve veći dio sveukupnog tržišta igara. Implementacija stvarnovremenske distribuirane simulacije virtualnog svijeta koja je potrebna za ovakve igre je vrlo složen problem. Stoga su programske biblioteke koje omogućuju povezivanje igrača u višekorisničkoj igri najčešći način implementacije višekorisničkih igara. Vaš zadatak je proučiti prethodno razvijenu biblioteku Fusion za razvoj višekorisničkih videoigara. Na temelju proučene biblioteke potrebno je implementirati višekorisničku igru s osnovnim funkcionalnostima u svrhu pomoći pri učenju o razvoju videoigara. Igra mora podržavati više igrača, njihovu interakciju u stvarnom vremenu te demonstrirati glavne mogućnosti Fusion biblioteke za razvoj višekorisničkih igara. Navedena igra mora biti detaljno dokumentirana, te ključni odsječci koda s temeljnim funkcionalnostima moraju biti jasno označeni te pojašnjeni u dokumentaciji.

Rok za predaju rada: 28. lipnja 2024.

Zahvala

Želio bih se zahvaliti mentoru, izv. prof. dr. sc. Mirku Sužnjeviću na znanju koje sam stekao pod njegovim mentorstvom, danim prilikama i pomoći pri izradi ovog rada.

Sadržaj

Uvod 1
1. Pregled područja 2
   1.1. Photon Fusion 2 2
   1.2. Photon Quantum 2 2
   1.3. Mirror Networking 3
   1.4. Unity Netcode for GameObjects 3
   1.5. Vlastita rješenja 3
3. Programska izvedba 6
   3.1. Photon Fusion 2 biblioteka 6
      3.1.1. Upute za instalaciju 6
      3.1.2. Ključne metode FusionConnection klase 7
      3.1.3. Komunikacija između igrača u arhitekturi klijent-poslužitelj 10
      3.1.4. InputManager 12
   3.2. ScriptableObject komponente 14
      3.2.1. LevelDataScriptable 14
      3.2.2. WeaponDataScriptable 14
      3.2.3. SensitivitySettingsScriptable 15
   3.3. Početni ekran 15
   3.4. Scena igre 19
      3.4.1. Mapa za igranje 19
      3.4.2. Korisničko sučelje 21
      3.4.3. PlayerManager 22
      3.4.4. PlayerStats 23
      3.4.5. PlayerCharacterController 25
      3.4.6. Projektili 28
      3.4.7. Objekti ozdravljenja 31
      3.4.8. GameManager 32
   3.5. Izrada potrebnih vizualnih sredstava 33
4. Izrada uputa za laboratorij 34
5. Rezultati 35
Zaključak 38
Literatura 39

Sažetak 40
Summary 41
Privitak 42

Uvod

Razvoj video igara je kompleksno područje. Ako se želi postići da više korisnika iz više različitih instanci aplikacije igre mogu igrati zajedno i komuniciraju jedan sa drugim, to još znatno otežava već zahtjevan zadatak. Komunikacija između korisničkih aplikacija se odvija preko internetske mreže. Za razvoj umreženih igara programer uz druga znanja potrebna za razvoj igara također mora znati algoritme i tehnike umreženog programiranja. Te tehnike se koriste za prijenos podataka o stanju igre između igrača, sinkronizacije stanja, kompenzacije kašnjenja mreže, autorizacija akcija za sprječavanje varanja i mnoge druge. Potreba za tim dodatnim funkcionalnostima znatno usporava i komplicira razvoj igara. Za olakšanje i ubrzanje razvoja umreženih igara može se koristiti jedno od gotovih rješenja za implementaciju mrežnih algoritama.

Na tržištu postoje razna rješenja za izradu umreženih višekorisničkih igara, sa svojim prednostima i manama. Zadatak ovog diplomskog rada je napraviti pregled dostupnih biblioteka za povezivanje igrača u višekorisničkoj igri, te napraviti pokaznu igru u Unity programskom alatu. Za implementaciju mrežnih algoritama odabrana je biblioteka Photon Fusion 2, te opisana njena upotreba. Igra treba biti žanra pucanja iz trećeg lica. Također je potrebno izraditi materijale za učenje i vježbu izrade umreženih višekorisničkih igara na temelju te pokazne igre. U sklopu materijala za učenje potrebno je izraditi verziju igre kojoj nedostaju dijelovi koda koju učenici trebaju nadopunjavati koristeći upute za vježbu, kako bi na praktični način stekli znanje o korištenju odabranog rješenja za izradu umreženih igara i proširiti svoje znanje o razvoju umreženih video igara.

1. Pregled područja

U ovom poglavlju razmatrati će se neka od dostupnih programskih rješenja za razvoj umreženih igara u Unity programskom alatu za razvoj igara (engl. *Game Engine*).

1.1. Photon Fusion 2

Photon Fusion 2 [1] je biblioteka za sinkronizaciju stanja preko mreže u Unity programskom alatu. Ona sinkronizira stanja objekata na način kojim je programerima jednostavno implementirati u svoj projekt. Također omogućuje programerima slanje podataka između klijentima pozivima udaljenih procedura (engl. *Remote Procedure Calls)* i umreženim svojstvima (engl. *Networked Properties)*. Komunikacija se odvija između klijenata preko Photon Fusion servera u oblaku, no jedan od klijenata se može smatrati domaćinom (engl. *Host)* čime je podržana i arhitektura klijent-poslužitelj. Klijenti se pridružuju u sobe i tako povezuju. Korisnik koji je napravio sobu se smatra glavnim klijentom (engl. *Master Client)*, no to ne znači da mora biti domaćin. Stvorene sobe se nalaze na Photon Fusion serveru u oblaku dok god je netko povezan u sobu, te se soba uništava ako nema niti jednog klijenta povezanog. Biblioteka šalje podatke u povezanim binarnim podatkovnim paketima koje šalje preko UDP veze. Photon Fusion 2 koristi algoritame kompresije kako bi smanjio zahtjeve za propusnošću s minimalnim opterećenjem CPU-a. Podaci se prenose u parcijalnim dijelovima, slijedeći model eventualne konzistentnosti (engl. *Eventual consistency)* koji osigurava da se promjena podataka na jednom računala propagira na sva ostala računala koji su dio te komunikacije. Jedna od korisnih funkcionalnosti ove biblioteke je izvođenje fizičkih zraka s kompenzacijom kašnjenja (engl. *Lag Compesantion Raycast)*, koje omogućuju precizno izračunavanje pogotka u igrama pucanja jer rješavaju problem kašnjenja u mreži. Prijenos stanja ove biblioteke nije deterministički, što znači da u jednom trenutku dva igrača mogu vidjeti različit prikaz stanja dok se ne sinkronizira. Pošto se biblioteka zasniva na prijenosu stanja potrebna je serverska autorizacija za igre u kojima nam je bitno spriječiti varanje, jer maliciozni akteri mogu slati netočna stanja igre.

1.2. Photon Quantum 2

Photon Quantum 2 [2] je programsko rješenje iste tvrtke kao i Fusion 2. Za razliku od Fusion 2 koji je biblioteka, Photon Quantum 2 je radni okvir za izradu igara. Cijela logika igre se piše u

Quatum ECS (engl. *Entity Component System)* sistemu koji se temelji na prijenosu korisničkih unosa umjesto stanja, a koristi Unity samo za vizualnu prezentaciju. Quantum 2 je deterministički, što znači da svi korisnici uvijek vide isto stanje. Pošto se prenose samo korisnički unosi nije potrebna serverska autorizacija, što je prednost u odnosu na Fusion 2. Radni okvir podržava predikciju stanja, tehniku povratka stanja (engl. *Rollback),* te je izrađen s namjerom da podržava veliki broj entiteta koje sinkronizira. Nedostatak u odnosu na Fusion 2 je to što programer mora naučiti koristiti Quantum ECS sistem koji se razlikuje od Unity razvojnog alata, što uzrokuje dodatni vremenski trošak dok se ne stekne znanje rada u tom sistemu.

1.3. Mirror Networking

Mirror Networking [3] je biblioteka otvorenog koda za razvoj umreženih igara u Unity programskom alatu. Za razliku od Photon Fusion 2 biblioteke, arhitektura igara razvijenih s Mirror bibliotekom je isključivo klijent-poslužitelj, gdje jedan od klijenata može biti server. Klijent i server se razvijaju u istom Unity projektu za povećanje produktivnosti. Podržava veliki broj konekcija i projekte velikog obujma. Dostupan je na Unity dućanu sredstava (engl. *Unity Asset Store)*, te je besplatan. Korisnici imaju mogućnost postavljanja svojih servera na Mirror servise u oblaku, no to se naplaćuje.

1.4. Unity Netcode for GameObjects

Unity Netcode for GameObjects [4] je biblioteka za razvoj umreženih igara od strane same tvrtke Unity. Iz tog razloga jako lagano se integrira u Unity projekt i omogućava lakšu implementaciju mrežnih algoritama. Dokumentacija biblioteke je opsežna i sadrži puno primjera projekata za isprobavanje i učenje. Podržava arhitekturu klijent-poslužitelj s autoritativnim serverom. Neki nedostatci su što ne podržava učitavanje podscena, klijentsku predikciju, interpolaciju animacija i namijenjen je za manje kooperativne igre do dvanaest igrača.

1.5. Vlastita rješenja

Za projekte umreženih igara nije potrebno koristiti gotova rješenja, nego razvojni tim može izraditi vlastita. Razvoj vlastitog rješenja zahtjeva dobro poznavanje mrežnih algoritama poput

razmjene poruka, sinkronizacije stanja, klijentske predikcije, serverskog pomirenja (engl. *Server Reconciliation)*, kompenzacije kašnjenja (engl. *Lag Compensation)* i serverska autorizacija. Razvijanje vlastitog rješenja je vremenski zahtjevan posao koji zahtjeva puno rada i znanja [5] što je nedostatak ovakvog pristupa. Također programer mora paziti na sigurnosne parametre sustava, jer za razliku od službenih rješenja koja su testirana sam mora brinuti da sustav nema sigurnosnih propusta koje maliciozni akteri mogu iskoristiti da nanesu novčanu štetu organizaciji i korisnicima. No, ono što je primamljivo kod ovakvog pristupa je nezavisnost o organizaciji koja razvija rješenje, takvo rješenje se puno bolje može prilagoditi potrebama projekta i jednom stečeno znanje o području mrežnih algoritama je svakako korisno za budućnost takvog programera.

2. Specifikacija zadatka

Za projekt je potrebno razviti umreženu igru za više igrača pucanja u trećem licu. Pri ulasku u igru igrač prvo treba unijeti svoje ime. Nakon toga se igraču prikazuje izbornik s popisom aktivnih soba kojima se igrač može pridružiti, izbornik za odabir vrste čarolija koje će igrač koristiti kao oružje u igri, te gumb koji otvara izbornik za izradu sobe. U izborniku za izradu sobe igrač mora unijeti ime sobe, odabrati hoće li se igra igrati u timovima ili u načinu svatko protiv svakoga, te odabir mape na kojoj će se igra odvijati. Nakon što se igrač pridruži sobi učitava mapu sobe, te mu se stvara lik kojega može kontrolirati. Igrač se s likom može kretati u trodimenzionalnom prostoru, pucati čarolije onoga tipa kojega je odabrao u početnom izborniku, te vidjeti druge igrače u prostoru. Ako igračev projektil pogodi protivničkog igrača u timskom načinu igre, ili bilo kojeg drugog igrača u načina igre svatko protiv svakoga, pogođenom igraču se čini šteta koja mu smanjuje životne bodove. U mapi se također nalaze predmeti ozdravljenja (engl. *Healing point)* koji igraču pri doticaju vraćaju životne bodove, te se onesposobljuju na kratko vrijeme. Ako igračevi životni bodovi dostignu nula, igračev lik se onesposobljuje na kratko vrijeme, nakon kojega se postavlja na nasumično odabranu jedno od unaprijed određenih pozicija. Igraču koji mu je zadnji napravio štetu dodaje se bod. U timskom načinu igre bodovi svih igrača se zajedno zbrajaju. Prikaz bodova se nalazi na vrhu ekrana, te također igrač zadavanjem komande može otvoriti prikaz bodova za svakog pojedinačnog igrača. U timskom načinu igre prvi tim koji dostigne unaprijed zadani broj bodova je pobjednik. U načinu svatko protiv svakoga vrijedi isto ali za svakog igrača zasebno. Nakon što je proglašen pobjednik, svi igrači su na kratko onesposobljeni, prikazano im je ime pobjednika ili pobjedničkog tima, te se nakon toga svi igrači postavljaju na nasumično mjesto odabrano od unaprijed određenih mjesta i svi bodovi se postavljaju na nulu. Igrač na ekranu ima gumb koji mu otvara izbornik s postavkama za upravljanje osjetljivosti okretanja kamere i gumbom za povratak na početni izbornik.

Potrebno je napraviti dvije verzije Unity projekta. Jednu cjelovitu i jednu kojoj fale dijelovi koda. Necjelovita verzija projekta je namijenjena kao materijal za učenje, u kojoj učenici moraju nadopuniti dijelove koji nedostaju. Za tu verziju trebaju biti napravljene upute sa zadacima.

3. Programska izvedba

Za diplomski rad je napravljena aplikacija u Unity radnom okviru za razvoj igara. Kod je pisan jezikom C#. U Unity projekt je dodana biblioteka Photon Fusion 2 [1] koja je korištena za implementaciju povezivanja više igrača preko interneta.

3.1. Photon Fusion 2 biblioteka

3.1.1. Upute za instalaciju

Za korištenje Photon Fusion servisa u Unity projekt je potrebno dodati Photon Fusion 2 Unity paket preuzet s njihovih službenih stranica. Potom je potrebno napraviti račun na Photon Fusion stranici ili se ulogirati u postojeći račun. Na stranici je potrebno otići na stavku Dashboard, te izraditi novu aplikaciju. Nakon što je aplikacija izrađena potrebno je kopirati podatak koji se nalazi pod App ID. U Unity projektu potrebno je otvoriti Photon Fusion 2 Hub izbornik pritiskom na stavku Tools u Unity alatnoj traci, odabirom Fusion stavke te pritiskom na Fusion Hub stavku. U Photon Fusion 2 Hub izborniku potrebno je zalijepiti kopirani podatak s Photon Fusion stranice u Fusion App Id polje za unos teksta. Nakon toga se može zatvoriti izbornik te je omogućeno korištenje Photon Fusion 2 biblioteke.

Povezivanje sa Photon Fusion 2 Cloud servisima i komunikacija sa drugim igračima ostvarena je u skripti *FusionConnection*. Ta skripta je postavljena kao komponenta *FusionConnection* game objekta, te je napravljena u Singleton obliku. Singleton oblik znači da im se može pristupiti preko statičke reference i da uvijek postoji samo jedna instanca svake skripte. *FusionConnection* skripta je također napravljena kao *DontDestroyOnLoad* objekt, što znači da se mijenjanjem scene ne uništava objekt na kojoj je ta skripta postavljena. *FusionConnection* nasljeđuje Photon Fusion 2 sučelje *INetworkRunnerCallbacks*. To sučelje definira metode putem kojih se komunicira s Photon Fusion servisima. U Unity projektu ovog diplomskog rada od definiranih metoda korištene su *OnSessionListUpdated* i *OnPlayerJoined*. U skriptu su također dodane metode *ConnectToLobby, CreateSession, JoinSession i LeaveSession* koji nisu dio *INetworkRunnerCallback*s sučelja.

3.1.2. Ključne metode FusionConnection klase

U Unity metodi *Awake* klase *FusionConnection* inicijaliziraju se komponente *NetworkSceneManagerDefault* i *NetworkRunner* Photon Fusion 2 biblioteke koje su potrebne za pravilan rad *FusionConnection* klase. *NetworkRunner* [6] služi za simulaciju stanja i sinkronizaciju udaljenih objekata, a *NetworkSceneManagerDefault* služi za automatsku promjenu scene u scenu igre nakon uspješnog spajanja u sobu.

OnSessionListUpdated metoda se poziva klijent dobije osvježenu listu otvorenih soba s Photon Cloud servisa. Preko njega se detektiraju promjene kada se otvori nova soba, zatvori stara soba ili promjeni broj igrača u sobama. Argument ove metode je lista soba u obliku *SessionInfo* klase, iz kojih se mogu izvući potrebne informacije o sobama. Neke od osnovnih podataka koje *SessionInfo* sadrži su ime sobe, broj igrača, maksimalni broj igrača, i listu prilagođenih svojstava u *Dictionary* obliku gdje je ključ ime svojstva u formatu string a vrijednost je u *Object* obliku kojega je potrebno pretvoriti u oblik željene klase.

ConnectToLobby metoda se poziva pri ulasku u glavni izbornik. Metoda sprema odabrano ime igrača u statičku referencu i poziva *JoinSessionLobby* metodu *NetworkRunner* instance. *JoinSessionLobby* prima kao argument *SessionLobby* enum. SessionLobby enum označava grupu soba u kojoj svi članovi imaju pristup informacijama i mogu pristupiti sobama drugih članova u istoj grupi. Vrijednosti *ClientServer* i *Shared* tog enuma se koriste za inicijalno dane grupe soba, a *Custom* za prilagođene grupe soba, uz koje je potrebno i dati ime grupe.

CreateSesion (Kôd 3.1) metodu poziva igrač koji stvara sobu. Metoda kao argument prima ime sobe, *GameModeType* enum koji predstavlja hoće li se igra igrati u timovima ili svatko za sebe i *LevelType* enum koji predstavlja na kojoj mapi će se igra odvijati. U metodi *CreateSession* se poziva metode *StartGame* instance *NetworkRunner*. Metoda *StartGame* kao argument prima strukturu *StartGameArgs* koja sadrži način komunikacije klijenata, ime sobe, indeks scene mape koja će se učitati nakon spajanja u sobu i koja se nalazi u build postavkama Unity-a, maksimalnog broja igrača i liste prilagođenih svojstava. Arhitektura komunikacije klijenata se određuje preko enuma *GameMode*. Vrijednosti *GameMode* enuma su:

- *Single* koji označava da će se soba igrati u načinu jednog igrača. Igrač se spaja na Photon Fusion server, ali konekcije drugih igrača u sobu nisu dopuštene.
- *Shared* u kojemu su svi igrači ravnopravni. Svaki igrač svim drugim igračima šalje svoje stanje, no svaki igrač može slati stanja o objektima kojima je on postavljen kao StateAuthority u instanci NetworkRunner skripte,

- *Host* i *Client* koji se koriste za hijerarhijsku klijent-poslužitelj arhitekturu. Prvi igrač koji pristupa sobi označava kao domaćin (engl. *Host*). Druge instance se smatraju klijentima, te oni domaćinu šalju samo svoje unose komandi. Domaćin je autoritet za stanje igre. Domaćin obrađuje poruke klijenata, preko njih simulira nova stanja, te šalje klijentima podatke o novom stanju. Klijentu lokalno izvode isti kod, kako ne bi morali čekati nove podatke stanja od domaćina nego pretpostaviti iduće stanje i tako smanjiti kašnjenje. Po dolasku podataka s domaćina klijent provjerava točnost svog stanja, te ga po potrebi ažurira. Ova tehnika se zove predikcija klijentske strane (engl. *Client side prediction),*
- *Server* koji je sličan slučaju s *Host* vrijednosti, s jedinom razlikom gdje u *Server* slučaju ta instanca igre se ne smatra igračem, nego namjenskim poslužiteljem (engl. *Dedicated server)*, te samo se brine za ažuriranje stanja drugih igrača i
- *AutoHostOrClient* koji prvog igrača koji pristupa sobi automatski postavlja kao domaćina, a sve iduće igrače za klijente.

U sklopu rada odabran je način klijent-poslužitelj s opcijama *Host* i *Client,* jer olakšava implementaciju logičkih elemenata koji zahtijevaju centralni autoritet poput objekata koje igrači mogu skupiti u mapi, npr. objekta za ozdravljenje. U ravnopravnom načinu komunikacije svaki igrač je autoritet za objekte koje on stvara, te se ti objekti uništavaju u trenutku kada igrač napusti sobu, što bi dovelo do neželjenog ponašanja da objekti za skupljanje koji bi trebali biti permanentni u mapi nestanu kada neki od igrača izađu iz sobe. U načinu klijent-poslužitelj taj problem je riješen prisustvom centralne logike jer je domaćin autoritet za sve objekte, te time oni postaju permanentni dok soba traje. Klijenti nisu autoritet stanja za svog lika igrača, nego samo autoritet unosa naredbi. Još jedan nedostatak arhitekture ravnopravne komunikacije je manjak centralnog autoriteta koji bi provjeravao da su stanja koje igrači šalju konzistentna, pa zlonamjerni igrač može slati netočna stanja i tako varati. Arhitektura gdje domaćin igra ulogu poslužitelja ne rješava taj problem u potpunosti, jer se poslužitelj izvršava na jednom od računala klijenta, te on može izmijeniti ponašanje poslužitelja na nelegitimni način. Jedini način koji osigurava zaštitu od varanja je korištenje arhitekture sa namjenskim poslužiteljem koji ne sudjeluje u igri, nego samo izvršava ulogu poslužitelja i izvršava se na računalu za kojeg znamo da nije kompromitirano.

JoinSession metoda se poziva kada se klijent želi spojiti u postojeću sobu. Kao argument metoda prima ime sobe i *GameMode* način igre. Metoda također poziva metodu *StartGame NetworkRunner-a. GameMode* argument metode *StartGame* se postavlja na *Client.*

OnPlayerJoined metoda se poziva kada se lokalni ili udaljeni igrač priključi sobi. Informacije o igraču koji se priključio se dobivaju u obliku *PlayerRef* strukture. Jedna od informacija koje se mogu koristiti iz te strukture je je li igrač koji se priključio lokalni ili udaljeni igrač. U slučaju kada se metoda pozove na računalu domaćina sobe instancira se novi game objekt igrača koristeći referencu na *PlayerCharacterController* [3.4.5] prefab. Tom objektu se kao autoritet unosa naredbi postavlja referenca igrača koji je pristupio igri. Potom se u slučaju timske igre klasi *PlayerStats* [3.4.4] tog igrača postavlja tim kojem pripada, a za odabir se uzima tim s trenutno manjim brojem igrača. Ako je igrač za kojeg se stvara objekt ujedno i lokalni igrač domaćin, zaključuje se da je to prvi poziv te metode te se stvaraju točke ozdravljenja [3.4.7] i instancira se objekt *GameManager* [3.4.8] klase.

OnPlayerLeft metoda se poziva u slučaju da je igrač izašao iz sobe. Kada je ova metoda pozvana na računalu domaćina, uništava se objekt njegovog lika u igri.

LeaveSession metoda poziva *ShutDown* metodu instance klase *NetworkRunner* koja je stvorena u metodi *ConnectToLobby* te učitava scenu početnog glavnog izbornika.

```
public async void CreateSession(string sessionName, GameModeType gameMode, 
LevelType level)
{
    _runner.ProvideInput = true;

    Dictionary<string, SessionProperty> properties = new Dictionary<string, 
SessionProperty>();
    properties.Add("level", SessionProperty.Convert((int)level));
    properties.Add("gameMode", SessionProperty.Convert((int)gameMode));
    _gameModeType = gameMode;

    await _runner.StartGame(new StartGameArgs()
    {
        GameMode = GameMode.Host,
        SessionName = sessionName,
        Scene = SceneRef.FromIndex((int)level),
        PlayerCount = _playerCount,
        SceneManager = _networkSceneManager,
        SessionProperties = properties,
    });
}
```

Kôd 3.1 *CreateSession* Metoda skripte *FusionConnection*

3.1.3. Komunikacija između igrača u arhitekturi klijent-poslužitelj

U arhitekturi klijent-poslužitelj domaćin ima postavljenu zastavicu *HasStateAuthority* postavljenu na istinito stanje za sve umrežene objekte u sceni. To znači da jedino on ima ovlaštenja mijenjati stanje tih objekata u sceni, te mijenjati stanje njihovih umreženih svojstava.

Umrežena svojstva (engl. *Network Properties*) su svojstva u koje Photon Fusion 2 sinkronizira preko mreže na svim klijentima, to znači da promjene napravljene na tim svojstvima se propagiraju svim ostalim igračima i oni ih vide. Umrežena svojstva se mogu koristiti kao svojstva klasa koje nasljeđuju klasu *NetworkBehaviour* [9] biblioteke *PhotonFusion*. Kako bi se svojstvo označilo kao umreženo svojstvo potrebno je dodati [*Networked*] atribut prije svojstva. Promjene takvih svojstava Photon Fusion 2 automatski propagira drugim igračima. Umrežena svojstva je potrebno mijenjati u *FixedUpdateNetwork* pozivima kako bi se osiguralo pravilno mijenjanje vrijednosti.

FixedUpdateNetwork metoda je implementacija Unity metode *FixedUpdate* biblioteke PhotonFusion gdje biblioteka očekuje promjene simulacije stanja *NetworkObject* [10] objekata i promjene umreženih svojstava. Ta metoda se poziva u vremenskim razmacima otkucaja igre, koji ne moraju odgovarati jednom prikazu igre u Unity razvojnom okruženju. Detekcija promjene se provodi dodavanjem *CallBack* metode unutar atributa [*Networked*] dodavanjem atributa *OnChangedRender* kojemu se predaje ime metode kao parametar. Igrač može mijenjati vrijednosti udaljenih svojstava samo onih *NetworkBehaviour* objekata nad kojima ima vrijednost varijable *HasStateAuthority* postavljeno na istinito, što je u slučaju arhitekture klijent-poslužitelj domaćin, a u ravnopravnoj arhitekturi je to klijent koji instancira taj objekt.

U arhitekturi klijent-poslužitelj klijenti koji nisu domaćin mogu samo imati postavljenu zastavicu *HasInputAuthority* postavljenu na istinito stanje za dijeljene objekte u sceni, što znači da samo njihov unos naredbi može biti registriran za te objekte. Kada žele mijenjati stanja objekata, klijenti to mogu zatražiti od domaćina na dva načina:

- Koristeći pozive udaljenih procedura [8] i
- Slanjem stanja naredbi.

Pozivi udaljenih procedura se koriste za izvršavanje metoda nad objektima nad kojima neki drugi klijent ima autoritet mijenjanja stanja. Ispred definicija tih metoda je potrebno dodati [*Rpc*] atribut. Tom atributu je također moguće dodati parametre za dozvoljena ishodišta i odredišta poziva. Ishodišta se određuju enum parametrom *RpcSources*, a odredišta enum parametrom *RpcTargets*. Metode je potrebno pozvati u lokalnoj instanci tog objekta, te će time Photon Fusion 2 izvršiti tu metodu na računalima klijenata koji odgovaraju odabranom *RpcTargets* atributu. Oba dva enuma imaju četiri moguće vrijednosti:

- All: označava da poziv može biti poslan od bilo kojeg klijenta ili da može biti obrađen od stane bilo kojeg klijenta,
- Proxies: označava da poziv može biti poslan ili obrađen samo od strane klijenata koji nema parametar StateAuthority ili parametar InputAuthority postavljen na istinit za taj objekt,
- InputAuthority: označava da poziv može biti poslan ili obrađen od strane klijenta koji ima parametar InputAuthority postavljen na istinit za taj objekt i
- StateAuthority: označava da poziv može biti poslan ili obrađen od strane klijenta koji ima parametar StateAuthority postavljen na istinit za taj objekt.

U arhitekturi klijent-poslužitelj *Rpc* atribut također kao parametar prima i enum *RpcHostMode* koji određuje ponašanje *Rpc* poruka koje dolaze s domaćina. Ako je taj parametar postavljen na vrijednost *SourceIsServer* smatra se da poruke koje dolaze ne dolaze od domaćina u ulozi poslužitelja, a ako je postavljen na *SourceIsHostPlayer* smatra se da poruke dolaze od domaćina u ulozi klijenta. Definicija poziva udaljenih procedura mora za prefiks ili sufiks imati „RPC", inače se kod neće moći kompajlirati i Unity će javljati grešku. Metode udaljenih procedura biblioteke Photon Fusion 2 mogu imati parametre, no ne mogu imati povratnu vrijednost, nego uvijek moraju vraćati void. Popis dopuštenih parametara se mogu pronaći na službenoj dokumentaciji biblioteke Photon Fusion 2 [11].

3.1.4. InputManager

Slanje stanja naredbi u projektu ostvareno je kroz skriptu *InputManager*. Photon Fusion 2 biblioteka skuplja podatke o korisnikovim naredbama u pozivima metoda *OnInput* koja je dio sučelja *INetworkRunnerCallbacks*. Ta metoda kao parametar prima *NetworkInput* objekt kojem je pri pozivu metode *OnInput* potrebno postaviti novo stanje pozivom njegove metode *Set* koja kao parametar prima strukturu koja nasljeđuje sučelje *INetworkInput.* U ovom projektu to je struktura *NetworkInputData* i u njoj su definirani svi podatci koji su potrebni za upravljanje vlastitim likom: smjer kretanja, rotacija, smjer pucanja i pritisnuti gumbi za pucanje i skok. Photon Fusion 2 za prijenos podataka od jednog bita, poput pritiska gumbi, nudi strukturu podataka *NetworkButtons.* Vrijednost za pritiske gumbi se postavlja pozivom metode *Set* koja kao parametre prima cjelobrojnu vrijednost indeksa gumba i bool vrijednost istinitosti. Metoda *OnInput* poziva se za svaki logički otkucaj biblioteke PhotonFusion, što se ne događa za svaki okvir igri, te bi neke od naredbi mogle biti izgubljene. Kako bi se to izbjeglo, naredbe se akumuliraju u metodi *BeforeUpdate* (Kôd 3.2) sučelja *IBeforeUpdate,* te se pri pozivu metode *OnInput* postavljaju za trenutne naredbe i resetiraju na početne vrijednosti.

```
public void BeforeUpdate()
{
    if (_reset)
    {
        _reset = false;
        _accumulatedInput = default;
    }

    Vector2 direction = Vector2.zero;
    NetworkButtons buttons = default;

    if (Input.GetKey(KeyCode.W))
        direction += Vector2.up;

    if (Input.GetKey(KeyCode.S))
        direction += Vector2.down;

    if (Input.GetKey(KeyCode.D))
        direction += Vector2.right;

    if (Input.GetKey(KeyCode.A))
        direction += Vector2.left;

    _accumulatedInput.Direction += direction;

    buttons.Set(NetworkInputData.JUMP, Input.GetKey(KeyCode.Space));

    buttons.Set(NetworkInputData.SHOOT, Input.GetMouseButton(0));
    if (Input.GetMouseButton(0)) _accumulatedInput.ShootTarget =
FusionConnection.Instance.LocalCharacterController.GetShootDirection();

    _accumulatedInput.Buttons = new
NetworkButtons(_accumulatedInput.Buttons.Bits | buttons.Bits);

    _accumulatedInput.YRotation += Input.GetAxis("Mouse X") *
SensitivitySettingsScriptable.Instance.LeftRightSensitivity;
}
```

Kôd 3.2 Metoda *BeforeUpdate* klase *InputManager*

3.2. ScriptableObject komponente

Unity projekt sadrži nekoliko *ScriptableObject* komponenti. One služe za perzistentan način spremanja podataka koji ne zavisi o instancama skripte u sceni. Kada se objekt na kojemu je skripta uništi ili se igra završi podatci se gube na tim skriptama, na *ScriptableObject* instance postoje kao objekti u hijerarhiji projekta i tako zadržava podatke van konteksta samih scena.

Sve *ScriptableObject* skripte u projektu su napravljene u *Singleton* obliku. Referenca za svaku instancu tih objekata treba biti dodana instanci *ScriptableHolder* skripte. Ta skripta je također u *Singleton* obliku, ali je napravljena i kao *DontDestroyOnLoad* objekt. *ScriptableHolder* također poziva *Init* funkciju svake *Singleton ScriptableObject* instance koja inicijalizira statičku referencu na samog sebe.

3.2.1. LevelDataScriptable

LevelDataScriptable sadrži podatke o mapama na kojima igrači mogu igrati. Svaka mapa mora biti dodana kao zasebna vrijednost u *LevelType* enum-u. Također, svaka mapa mora biti dodana u popis scena za build u build postavkama Unity-a. *LevelDataScriptable* za svaku mapu sadrži vrijednost njenog *LevelType* enum-a, referencu na sliku koja će se prikazivati za izbor te mape, te referencu na scenu koja se treba učitati za tu mapu.

3.2.2. WeaponDataScriptable

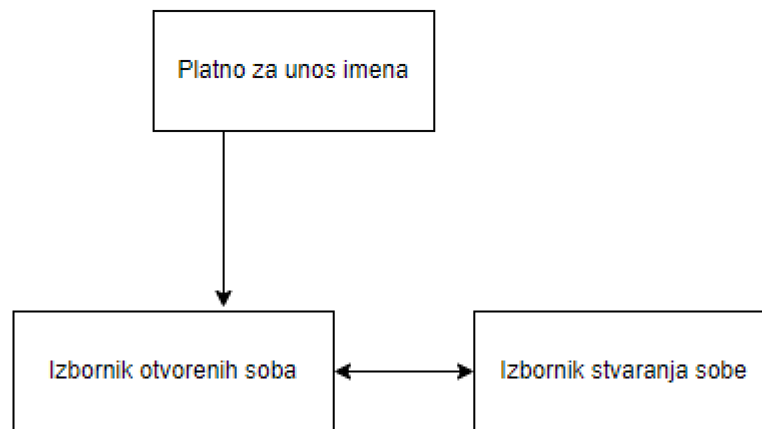
WeaponDataScriptable sadrži podatke o borbenim čarolijama koje igrači mogu odabrati. Svaka čarolija mora biti dodana kao zasebna vrijednost u *WeaponType* enum-u. *WeaponDataScriptable* za svaku čaroliju sadrži vrijednost njenog *WeaponType* enum-a, slike koja će se prikazivati za izbor te čarolije, referencu na prefab projektila te čarolije po kojem će se stvarati nova instanca objekta svaki puta kada se ispali projektil, referencu na prefab za rukavice koje će se postaviti na ruke igrača koji je odabrao tu čaroliju kao oružje, pomak rukavice u odnosu na ruku u Vector3 obliku koji poravnava vizualnu reprezentaciju rukavice s rukom igrača, te brzinu ispaljivanja koja određuje koliko brzo može pucati igrač koji je odabrao taj tip čarolije.

3.2.3. SensitivitySettingsScriptable

SensitivitySettingsScriptable sadrži podatka za upravljanje osjetljivošću okretanja kamere u igri. Sadrži podatke za brzinu okretanja kamere u smjeru gore-dolje i lijevo-desno.

3.3. Početni ekran

Početni ekran se nalazi u *LoginSceen* sceni. Sastoji se od objekta platna na kojem se nalazi izbornik za unos imena, izbornik otvorenih soba i izbornik izrade sobe (Sl. 3.1 Dijagram toga promjene izbornika početne scene). Na početnom ekranu je prvo prikazano samo polje za unos teksta preko kojega igrač unosi svoje ime i gumb za potvrdu unosa. Njihovo ponašanje se kontrolira kroz skriptu *LoginView*. Prilikom pritiska na gumb za potvrdu unosa izvršava se provjera je li ime dovoljne duljine, te se potom gasi game Login game objekt i uključuje *SessionExplorer* game objekt. Login game objekt je u hijerarhiji scene dijete glavnog UI platna i na njemu je stavljena *LoginView* skripta kao komponenta. *SessionExplorer* sadrži izbornik za odabir borbenih čarolija, izbornik otvorenih igara, te izbornik za izradu igara.



Sl. 3.1 Dijagram toga promjene izbornika početne scene

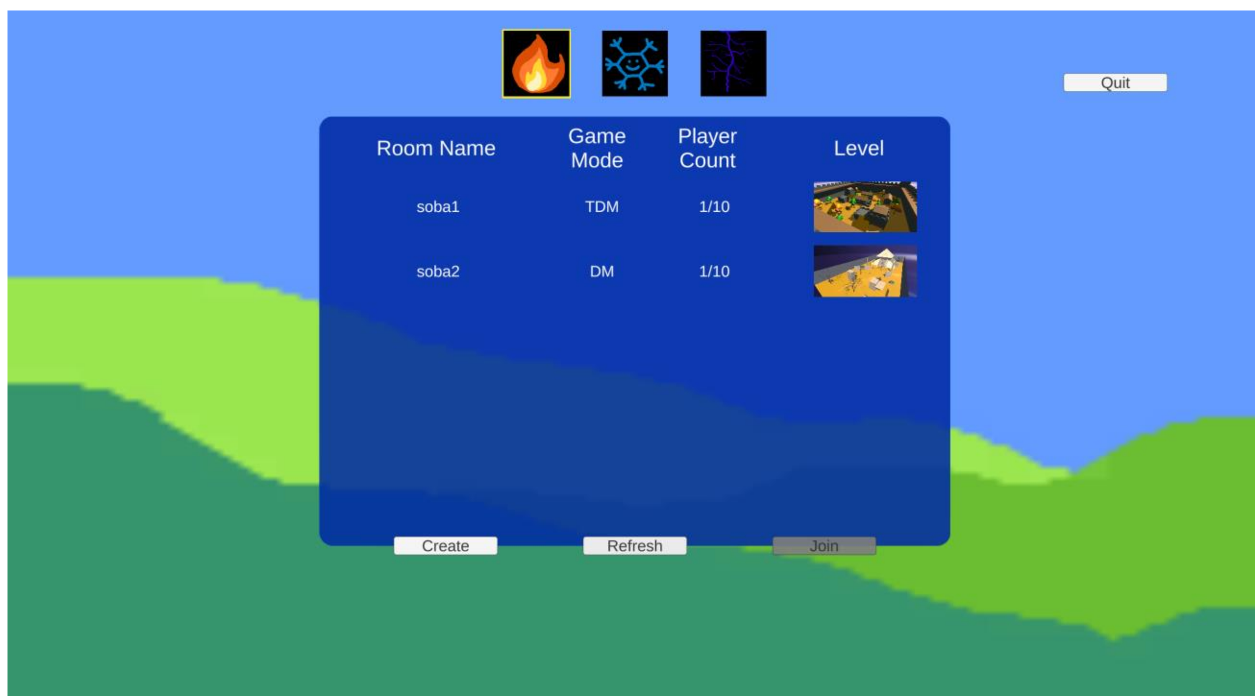
Izbornik za odabir čarolija se sastoji od elemenata s *Toggle* komponentom. Skripta *SessionView* na game objektu *Sessions* pri prvoj aktivaciji objekata inicijalizira *Toggle* elemente izbornika. *SessionView* pri prvoj aktivaciji svog objekta instancira novi game objekt za svaki unos u listi čarolija koje sadrži *WeaponDataScriptable* instanca (Kôd 3.3). Kao roditelj tih objekata u hijerarhiji se postavlja *WeaponSelectionToggle* objekt koji sadrži *HorizonatlLayoutGroup* komponentu koja automatski formatira pozicije tih objekata da su poredani jedni pored drugih i *ToggleGroup* komponentu koja osigurava da je uvijek barem jedna *Toggle* komponenta postavljena

kao upaljena. Za stvaranje tih game objekata koristi referencu na *WeaponSelectionToggle* prefab. Svaki *WeaponToggle* objekt referencu na svoju sliku i *WeaponType* varijablu koja govori na koju čaroliju se taj *WeaponToggle* odnosi. Ti podatci se popunjavaju iz *WeaponDataScriptable* informacije za sliku te čarolije, te referencu na svoju *Toggle* komponentu. Po aktivaciji svoje *Toggle* komponente, *WeaponToggle* poziva svoju *CallBack* metodu koju instanci *WeaponToggleSkriptable* skripte postavlja kao trenutno odabrani *WeaponType* onaj koji pripada toj *WeaponSelectionToggle* komponenti.

```
private void Awake()
 {
     base.Awake();

     . . .

     foreach (var data in WeaponDataScriptable.Instance.Weapons)
     {
         WeaponSelectionToggle weaponToggle =
              Instantiate(_weaponSelectionTogglePrefab,
               _weaponSelectionContainer.transform);

         weaponToggle.ShowWeapon(data.WeaponType, _weaponSelectionContainer,
            data.WeaponImage, (weaponType) =>
            WeaponDataScriptable.SetSelectedWeaponType(weaponType));
     }

     . . .
 }
```

Kôd 3.3 Inicijalizacija *WeaponToggle* komponenti u *SessionView* skripti

Izbornik otvorenih igara prikazuje listu svih igara koje su trenutno u tijeku i kojima se igrač može pridružiti. Osvježavanje liste igara se odvija kroz metodu *UpdateSessionList* skripte *SessionView*. Ta metoda se poziva prvi puta kada s Photon Cloud servisa dođe nova lista soba kroz metodu *OnSessionListUpdates* klase *FusionConnection* ili kada korisnik pritisne na gumb Refresh koji se nalazi u tom izborniku i na koji klasa *SessionView* ima referencu. Metoda *UpdateSessionList* prvo počisti stare prikaze iz popisa soba, te potom za svaku sobu iz nove liste instancira game objekt koristeći referencu na *SessionDataView* prefab. Kao roditelj u hijerarhiji se postavlja game objekt *SessionList* koji sadrži komponentu *VerticalLayoutGroup* koja vertikalno rasprostranjuje prikaze i koja je dijete game objekta ScrollView koji omogućuje korisniku listanje prikaza otvorenih soba u slučaju da ih ima previše. *SessionDataView* prikazuje ime sobe, način igre, broj igrača, sliku mape na kojoj se odvija soba i sadrži referencu na vlastitu *Toggle* komponentu. Pri aktivaciji *Toggle* komponente poziva se metoda *SessionToggle* klase *SessionView* koji aktivira Join gumb za pristup i postavlja podatke o toj sobu u privatnu varijablu tipa touple, koji u sebi sadrži potrebne informacije, kao trenutno odabrane. Ako se ponovno pritisne na isti element i deaktivira se *Toggle* komponenta, Join gumb se postavlja kao neaktivan. Pritiskom na Join gumb poziva se metoda *JoinSession* klase *FusionConnection* šaljući joj kao argumente spremljene informacije o odabranoj sobi.

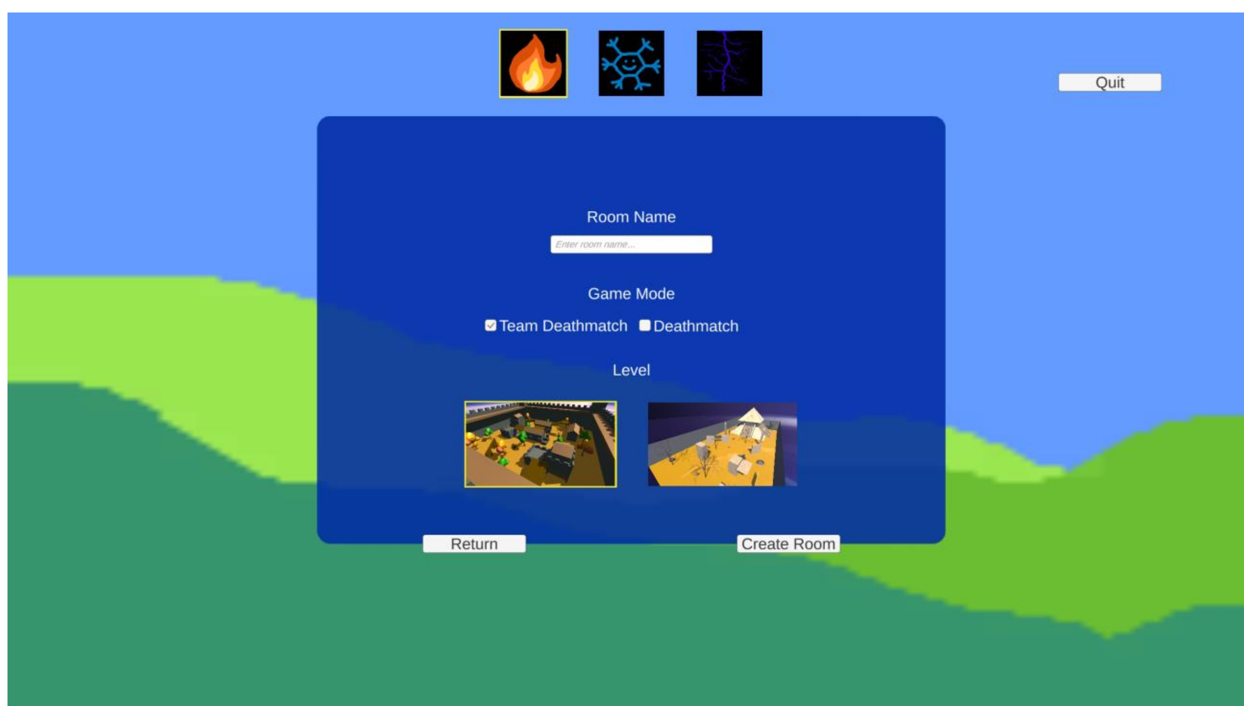


Sl. 3.2 Izbornik otvorenih igara i izbornih borbenih čarolija

Pritiskom na gumb Create Room gasi se game objekt za prikaz izbornika otvorenih igara i pali se game objekt za prikaz izbornika izrade igara. Izbornik za izradu igre sadrži polje za unos imena sobe, gumbi za odabir načina igre i gumbi za odabir mape koristeći *Toggle* komponente, gumb za potvrdu stvaranja sobe i gumb za povratak na izbornik otvorenih igara. Ponašanje izbornika se kontrolira kroz *RoomCreationView* skriptu. Na prvom uključivanju game objekta *RoomCreationView* skripta dohvaća popis podataka o svim mapa postavljenim u instanci *LevelDataScriptable* skripte, te za svaku od njih stvara novu instancu objekta za izbor mape i inicijalizira ju s podatcima o mapi (Kôd 3.4). Za roditelja tih instanci u hijerarhiji se postavlja LevelToggle game objekt koji sadrži HorizontalLayoutGroup komponentu koja horizontalno rasprostranjuje instance i ToggleGroup komponentu koja osigurava da je uvijek barem jedna od njih postavljena kao odabrana. Za stvaranje instanci se koristi referenca na LevelSelectionToggle prefab. LevelSelectionToggle sadrži prikaz slike mape i referencu na vlastitu Toggle komponentu. Po pritisku na Create Room gumb poziva se metoda JoinSession klase FusionConnection, te joj se kao argumenti šalju podatci o unesenom imenu sobu, odabranom načinu igre i odabranoj mapi. Pritiskom na Return gumb gasi se game objekt za prikaz izbornika izrade igara i pali se game objekt za prikaz izbornika otvorenih igara.

```
private void Awake()
 {
     foreach(var data in LevelDataScriptable.Instance.Levels)
     {
         LevelSelectionToggle selectionToggle = Instantiate(
            levelSelectionTogglePrefab, _levelSelectionContainer.transform);
         selectionToggle.ShowLevel(data.LevelType, _levelSelectionContainer,
            data.LevelImage, (levelType) => _selectedLevelType = levelType);
     }

     . . .
 }
```

Kôd 3.4 Inicijalizacija *LevelSelectionToggle* komponenti u *RoomCreationView* skripti



Sl. 3.3 Izbornik izrade soba

3.4. Scena igre

Scena igre se sastoji od terena po kojemu se igrači mogu kretati, platna za prikaz korisničkog sučelja i servisa za izbor lokacije stvaranja igrača. Platno korisničkog sučelja se sastoji od prikaza trenutnih života, ciljnika, prikaza timskih bodova, prikaza dobivenih i danih bodova svakog igrača i izbornika pauze. U ovom Unity projektu svaka scena mape je scena igre. Za diplomski rad dodane su dvije scene igre: srednjovjekovni grad i pustinja.

3.4.1. Mapa za igranje

Svaka scena za igranje sadrži teren po kojemu se igrači kreću. Za diplomski rad izrađeni su potrebni modeli i dizajnirana je mapa s temom srednjovjekovnog grada. Svi elementi terena koji sadrže *Collider* komponentu trebaju imati Tag komponentu postavljenju na vrijednost *Ground* kako bi igračev lik mogao detektirati je li prizemljen i trebaju imati *Layer* komponentu postavljenu također na vrijednost *Ground*, jer skripta za kontroliranje kamere pri detektiranju kolizije koristi *LayerMask* postavljen samo na vrijednost *Ground*. Podatci za svaku izrađenu mapu moraju biti dodani u instancu *LevelDataScriptable* [3.2.1] skripte kako se tu mapu moglo odabrati, te svaka scena mape mora biti dodana u build postavke Unity-a.

Sl. 3.4 Izrađena mapa sa temom srednjovjekovnog grada

3.4.2. Korisničko sučelje

Korisničko sučelje scene igre se kontrolira kroz skriptu UIManager. U sredini korisničkog sučelja se nalazi ciljnik koji igraču omogućava precizno pogađanje željene mete. U donjem desnom dijelu nalazi se tekst i klizajući prozor koji prikazuju trenutne životne bodove igrača. U gornjem desnom kutu ekrana se nalazi Pause gumb, pritiskom na koji se prikazuje izbornik pauze. Izbornik pauze se sastoji od klizajućih izbornika za promjenu osjetljivosti okretanja kamere u smjeru gore-dolje i okretanja lika u smjeru lijevo-desno, Resume gumb za ponovno skrivanje izbornika pauze i Leave gumb za napuštanje igre. Pritiskom na Leave gumb poziva se *LeaveSession* metoda klase *FusionConnection* gasi instancu *NetworkRunner* skripte i učitava scenu početnog izbornika. Promjene klizajućih izbornika za osjetljivost skripta *UIManager* sprema u instancu skripte *SensitivitySettingsScriptable* [3.2.3], čije vrijednosti se kasnije koriste pri okretanju kamere i lika. Korisničko sučelje također sadrži game objekt koji se uključuje kada životi lokalnog igrača dostignu 0 i koji odbrojava sekunde do ponovne aktivacije lika, te game objekt koji se uključuje kada jedan od timova dostigne broj bodova dovoljnih za pobjedu te pokazuje ime pobjedničkog tima u načinu timske igre ili kada jedan od igrača postigne broj bodova dovoljan za pobjedu te pokazuje ime pobjedničkog igrača u načinu igre svatko protiv svakoga. Na vrhu ekrana prikazuje se broj bodova koji svaki tim posjeduje u timskom načinu igre ili broj bodova koje posjeduje lokalni igrač u načinu igre svatko protiv svakoga. Pritiskom na tipku tab uključuje se game objekt za prikaz dobivenih i danih bodova za svakoga igrača. Za svakog igrača koji se priključi sobi instancira se novi game objekt koristeći referencu na *PlayerScoreboardData* prefab. *PlayerScoreboardData* prefab prikazuje ime igrača, broj dobivenih i danih bodova, te sprema privatnu varijablu tipa *TeamType* enum u kojemu pamti kojem timu pripada lokalni igrač. *TeamType* enum ima tri vrijednosti:

- *None* koji označuje da igrač ne pripada ni jednom timu te da se igra odvija u načinu svatko protiv svakoga,
- *TeamA* koji označuje da igrač pripada timu A i
- *TeamB* koji označuje da igrač pripada timu B.

Ovisno o toj varijabli, svaki puta kada se inicijaliziraju podatci klase *PlayerScoreboardData* prikazani tekstovi se boje u prijateljsku boju ako se radi o podatcima za lokalnog igrača ili igrača koji pripada istom timu, ili u neprijateljsku boju ako se radi o igrači iz protivničkog tima ili se igra odvija u načinu svatko protiv svakoga. Boje se dobivaju iz instance skripte *PlayerManager*.

Sl. 3.5 Prikaz danih i dobivenih bodova svakog igrača

3.4.3. PlayerManager

PlayerManager skripta služi za upravljanje referencama o igračima i informacijama o timskoj pripadnosti. Nakon što se u sceni stvori game objekt igrača i inicijaliziraju sve potrebne stavke od strane Photon Fusion 2 biblioteke skripta *PlayerStats* [3.4.4] koja pripada objektu tog igrača poziva metodu *RegisterPlayer* (Kôd 3.5) klase *PlayerManager*. Metoda *RegisterPlayer* dodaje referencu na tog igrača u lokalnu listu svih igrača. *PlayerManager* također sadrži metodu *SetPlayerColor* koja igraču postavlja timsku boju ovisno o tome podudara li se ili ne tim tog igrača s timom lokalnog igrača. Klasa PlayerStats po uništenju svog game objekta poziva metodu UnregisterPlay klase PlayerManager koja ga miče iz liste igrača, jer se game objekt klase *PlayerStats* uništava samo kada taj igrač napusti sobu. Klasa također sadrži metode za postavljanje prijateljskog tima, nakon kojega se ažuriraju timske boje svih do sada registriranih igrača, te metode za emitiranje poziva *SendGameEndRpc* i *SendGameStartRpc* za sve igrače koja im javlja da je jedan od timova ili jedan od igrača postigao dovoljni broj bodova za pobjedu. Te metode se pozivaju samo sa strane domaćina, te one pozivaju metode udaljenih procedura, kako bi svaki igrač lokalno dobio informaciju da je igra gotova i da se prikaže ekran kraja igre.

```
public void RegisterPlayer(PlayerStats player)
{
    _playerStats.Add(player);
    if(_friendlyTeam == TeamType.None || player.Team != _friendlyTeam)
                  player.SetTeamMaterial(_enemyMaterial, _enemyColor);
    else player.SetTeamMaterial(_friendlyMaterial, _friendlyColor);
}
```

Kôd 3.5 Metoda za registraciju igrača klase *PlayerManager*

3.4.4. PlayerStats

PlayerStats skripta sadrži podatke o igraču koji su dio logike igre. Ti podatci u obliku umreženih svojstava koje biblioteka Photon Fusion 2 sinkronizira s ostalim igračima. Skripta sadrži umrežena svojstva o podatcima igrača. Podatci koji se spremaju u ovu skriptu su ime igrača, tim kojem igrač pripada, odabrana borbena čarolija, trenutni životni bodovi, broj osvojenih, broj danih bodova i preostalo vremensko trajanje usporenog kretanja. Ta umrežena svojstva ažurira isključivo domaćin, dok drugi klijenti detektiraju promjene preko *CallBack* metoda. Detektirane promjene služe za ažuriranje elemenata korisničkog sučelja kako bi prikazivali točno stanje. Klasa *PlayerStats* nasljeđuje klasu *NetworkBehaviour* biblioteke Photon Fusion 2. Kada je instanciran game objekt s klasom *PlayerStats* i Photon Fusion 2 inicijalizira sve potrebne vrijednosti poziva se metoda *Spawned*. U toj metodi klasa *PlayerStas* sprema referencu na *PlayerCharacterController* [3.4.5] komponentu s vlastitog game objekta i poziva metodu *RegisterPlayer* klase *PlayerManager* [3.4.3] kako bi taj igrač bio dodan u lokalni popis igrača i kako bi mu se postavile timske boje. Kad je metoda *Spawned* pozvana na klijentu koji ima autoritet davanja naredbi za tog igrača on poziva metodu udaljenog poziva *RPC_InitializeData* koja se izvršava na domaćinu i kojoj prosljeđuje ime igrača i odabrano oružje jer su to podatci koje zna samo klijent, a domaćinu su potrebni za inicijalizaciju i ažuriranje umreženih svojstava. Nakon postavljanja inicijalnih vrijednosti *PlayerStats* stvara novu instancu game objekta koristeći referencu na *PlayerScoreboardData* prefab za prikaz podataka o bodovima igrača. Također ažurira tekst koji stoji iznad igrača s igračevim imenom, te ga za svaki novi prozor igre rotira prema kameri.

Klasa sadrži metode *ApplySlow, Heal* i *DealDamage* koji se pozivaju na strani domaćina i koji služe za ažuriranje podataka o igraću prilikom pogotka projektilom, skupljanja objekta ozdravljenja i

započinjanja nove runde. Domaćin svakom novom prozoru igre smanjuje preostalo vrijeme usporenog kretanja ako je trajanje veće od nula. U metodi *DealDamage* (Kôd 3.6) životi igrača se smanjuju za vrijednost štete koju nanosi projektil koji je pogodio igrača. Ako životi igrača padnu na 0 povećava se broj bodova igraču koji je nanio štetu, a igraču koji primio štetu povećava podatak o broju danih bodova. Pri povećanju broja dobivenih ili danih bodova pozivaju se metode za ažuriranje prikaza broja bodova klase *UiManager* i instance klase *PlayerScoreboardData* koja pripada igraču. Ako je novi broj dobivenih bodova veći ili jednak broju bodova za pobjedu koji je definiran u klasi *GameManager* [3.4.8], poziva se metoda *GameEnd* klase *GameManager* koja šalje poziv završava rundu i obavještava klijente o kraju runde pozivima udaljenih metoda.

Klasa također sadrži metode za postavljanje materijala timske boje na vizualni prikaz igrača, metodu za resetiranje podataka o igri i metode za ponovno postavljanje životnih bodova na maksimalnu vrijednost koji se pozivaju kada započinje nova runda.

```
public void DealDamage(int damage, PlayerStats attacker)
{
    if (Health - damage > 0)
    {
        Health -= damage;
        return;
    }

    if (Health == 0) return;

    Deaths++;
    attacker.Kills++;
    _playerCharacterController.PlayerKilled();

    if (FusionConnection.GameModeType == GameModeType.DM &&
            attacker.Kills >= GameManager.Instance.SoloKillsForWin)
    {
        GameManager.Instance.GameEnd(attacker.PlayerName.Value);
    }
    else if (FusionConnection.GameModeType == GameModeType.TDM)
    {
        GameManager.Instance.AddTeamKill(attacker.Team);
        if (GameManager.Instance.GetTeamKills(attacker.Team) >=
                    GameManager.Instance.TeamKillsForWin)
            GameManager.Instance.GameEnd(attacker.Team);
    }

    Health = 0;
}
```

Kôd 3.6 *DealDamage* metoda klase *PlayerStats* za nanošenje štete igraču

3.4.5. PlayerCharacterController

Klasa *PlayerCharacterController* je glavna klasa koja upravlja fizikom kretanja lika igrača. Nasljeđuje klasu *NetworkBehaviour* biblioteke Photon Fusion 2 te koristi njene *Spawned* i *FixedUpdateNetwork* metode.

Spawned metoda se poziva nakon što se instancira igrajući objekt kojemu je *PlayerCharacterController* dodan kao komponenta, te nakon što Photon Fusion 2 biblioteka inicijalizira sve potrebne vrijednosti. Na poziv *Spawned* metode klasa *PlayerCharacterController* poziva metodu za inicijalizaciju klijentskih varijabli na klijentu koji je autoritet za unos naredbi tog objekta i metodu za inicijalizaciju serverskih varijabli na domaćinu. Klijent inicijalizira poziciju kamere, postavlja statičku referencu na lokalnog igrača klase *FusionConnection* [3.1.2] i inicijalizira klasu za kontrolu animacija lika *PlayerAnimationController.* Domaćin inicijalizira *PlayerAnimationController* klasu na svojoj strani te postavlja početnu poziciju liku. Početna pozicija lika postavlja se pozivanjem metode *Teleport* klase *NetworkCharacterController* Photon Fusion 2 biblioteke. *NetworkCharacterController* klasa dolazi uz biblioteku, te je namijenjena za pomicanje umreženih objekata, te rekalkulaciju pozicije nakon dobivanja podataka o poziciji objekta s poslužitelja. *NetworkCharacterController* zahtjeva komponentu *CharacterController* Unity razvojnog okruženja i preko njega pomiče lika i postiže kolizije. U projektu svo kretanje je postignuto kroz *NetworkCharacterController* kojemu je izmijenjena logika kretanja za potrebe projekta, kako bi se postiglo željeno korisničko iskustvo pri kontroliranju lika. Koordinate početne pozicije dohvaćaju se iz klase *SpawnLocationManager*. *SpawnLocationManager* je klasa u singleton obliku koja sadrži reference na prazne game objekte u sceni. Te objekte je potrebno postaviti na mjesta gdje se želi omogućiti postavljanje lika. Lokacija se dohvaća preko metode *GetRandomSpawnLocation* koja vraća koordinate nasumično odabranog game objekta iz popisa.

U *FixedUpdateNetwork* metodu izvršavaju se akcije mijenjanja pozicije lika u sceni. Mijenjanje pozicije se izvršava i na domaćinu koji je autoritet stanja i na klijentu za klijentsku predikciju. Na početku se provjerava je postavljena zastavica za ponovno postavljanje pozicije. Ako je postavljena, zastavica se miče i pozicija se postavlja na novu nasumičnu vrijednost iz klase *SpawnLocationManager.* Za simulaciju kretanje su potrebne posljednje naredbe dane od klijenta koji ima autoritet davanja naredbi za taj objekt. Naredbe se dobivaju pozivanjem metode *GetInput NetworkBehaviour* klase, i dobivaju se u formatu *NetworkInputData* strukture. Sljedeće se izvršava provjera je li igrač u doticaju s tlom koristeći varijablu *isGrounded* Unity *CharacterController* komponente. Ako je dana naredba za skok, a igrač je u doticaju s tlom ili ako igrač nije u doticaju s tlom, ali varijabla duplog skoka je postavljena na aktivnu i od prošlog skoka je prošlo dovoljno vremena poziva se metoda *Jump NetworkCharacterController.* Igraču se ažuriraju animacije koristeći varijable smjera prikupljene iz Update funkcije i varijable doticaja s tlom, te se potom poziva metoda *Move* klase *NetworkCharacterController* za promjenu pozicije. Metoda *Move* dobiva horizontalne parametre vektora kretanja koristeći varijable smjera i varijablu brzine kretanja. Ako je varijable

usporenja pripadne klase *PlayerStats* postavljena kao aktivna vektor kretanja se množi faktorom usporenja. Za vertikalni parametar smjera kretanja koristi se lokalna varijabla vertikalnog smjera kretanja. Od te varijable se oduzima vrijednost gravitacije. U slučaju da je lik u doticaju s tlom varijabla vertikalnog smjera kretanja se postavlja na nula i varijabla duplog skoka se postavlja na aktivno stanje. Potom se varijabla smjera kretanja šalje u metodu *AdjustVelocityToSlope* koju smjer kretanja prilagođava nagnutim površinama poput krovova. To se izvodi korištenjem metode *Raycast* Unity *Physics* klase kako bi se dobio kut nagiba plohe na kojoj lik stoji, te pomnožio tu vrijednost sa smjerom kretanja u *Quaternion* obliku. Na kraju se varijabli smjera kretanja dodaje varijabla vertikalnog smjera kretanja, te se poziva metoda *Move* *CharacterController* komponente igračevog lika. Metodi *Move* se kao parametar šalje varijabla smjera kretanja, te ona pomiče lika za taj vektor. U *Move* metodi se također izvodi okretanje lika ovisno o podatku kretanju miša iz dobivenih naredbi kretanja i osjetljivosti okretanja lijevo-desno iz instance klase SensitivitySettingsScriptable [3.2.3]. Na kraju se vrijednost smjera kretanja i varijabla doticaja s tlom spremaju u varijablu *Data* tipa *NetworkCCData* klase *NetworkCharacterController.* Ta varijabla se koristi kako bi klijent mogao provjeriti točnost svoje simulacije i popraviti svoju poziciju po potrebi.

Sl. 3.6 Borba dva igrača

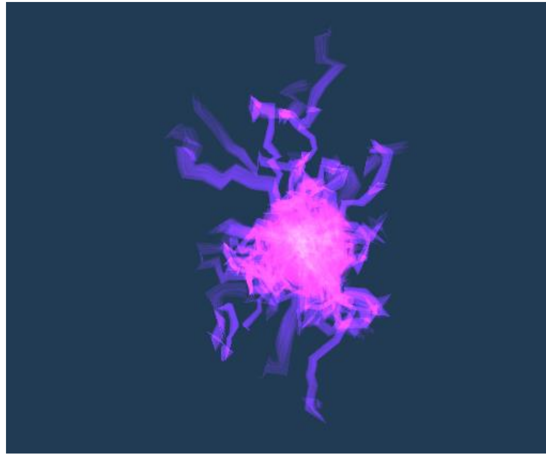
U *FixedUpdateNetwork* metodi se također izvodi akcija pucanja. Akcija pucanja se izvodi na pritisak miša ako je proteklo vrijeme od prošlog pucanja veće od vrijednosti brzine pucanja za tu

borbenu čaroliju iz *WeaponDataScriptable* [3.2.2] instance. Pri izvođenju pucanja, prvo se vrijeme zadnjeg pucanja postavlja na trenutno vrijeme te se izvodi animacija pucanja. Na klijentu se samo izvodi animacija pucanja, dok se na domaćinu uz animaciju i instancira novi game objekt projektila koristeći referencu na *Projectile* prefab dobiven iz *WeaponDataScriptableInstance*. Nakon stvaranja projektila postavlja mu se smjer podatkom iz dobivenih naredbi kretanja koji se dobiva iscrtavanjem zrake od kamere kroz sredinu ekrana, te uzimanjem pogotka kao odredištem smjera. Smjer pucanja se predaje projektilu u pozivu metode *Throw*, gdje joj se također predaje referenca na igrača koji je stvorio taj projektil.

Klasa *PlayerCharacterController* sadrži metodu *PlayerKilled* koju poziva pripadna instanca klase *PlayerStats* kada životni bodovi igrača padnu na 0. Ovaj poziv se obavlja na strani domaćina. Domaćin započinje korutinu odbrojavanja dok se igračev lik ponovno ne osposobi, te poziva metodu *RPC_PlayerKilled* udaljenog poziva koja se izvodi na klijentu koji ima autoritet davanja naredbi tom liku. Ta metoda izvodi animaciju umiranja lika, poziva metodu klase UiManager koja prikazuje ekran gubitka bodova, te onesposobljuje kontroliranje lika i kamere. Klijent ažurira vrijednost odbrojavanja u ekranu za prikaz gubitka korištenjem *CallBack* metode umreženog svojstva *RemainingRespawnTime* . Nakon isteka odbrojavanja metoda postavlja vrijednost lokalne varijable koja označava da je potrebno ponovno postaviti nasumičnu poziciju lika, te osposobljuje kontroliranje lika. PlayerCharacterController također sadrži metode *RPC_GameEnd* i *RPC_GameStart* koja izvršava slično ponašanje. Metoda GameEnd prvo deaktivira ekran gubitka bodova ako je aktivan, te aktivira ekran kraja igre na kojemu se ispisuje ime pobjedničkog tima ili igrača pozivima metoda klase UiManager. Metoda *Rpc_GameStart* skriva ekran kraja igre i aktivira korištenje kamere i pomicanje lika.

3.4.6. Projektili

Klase za sve projektile u Unity projektu nasljeđuju apstraktnu klasu *Projectile.* Apstraktna klasa *Projectile* nasljeđuje klasu *NetworkBehaviour* biblioteke Photon Fusion 2 kako bi se ponašanje svih projektila koju ju nasljeđuju moglo sinkronizirati između klijenata. Klasa *Projectile* definira varijable koje će koristiti svaki projektil: varijablu za brzinu kretanja, za količinu štete koju čini na pogodak, umreženu varijablu smjera kretnja, te umreženu varijable koje sadrži reference na instancu klase *PlayerStats* [3.4.4] i *PlayerRef* strukturu Photon Fusion 2 biblioteke igrača koji je ispucao projektil. Klasa također definira metodu *Throw* koja treba biti pozvana nakon stvaranja projektila i koja započinje njegovu putanju i postavlja mu rotaciju. Za projekt su napravljeni projektili za tri čarolije napada: vatreni, ledeni i električni projektil.

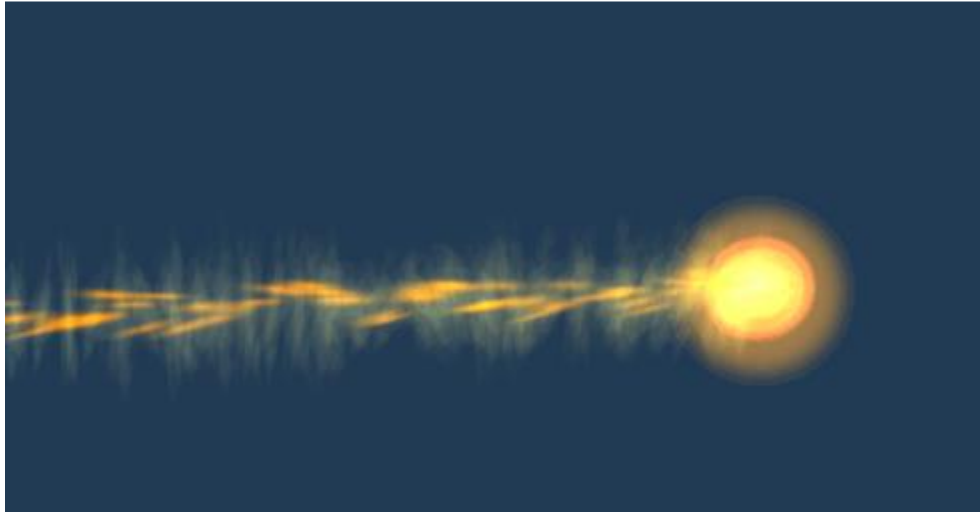


Sl. 3.7 Izgled efekta električnog projektila

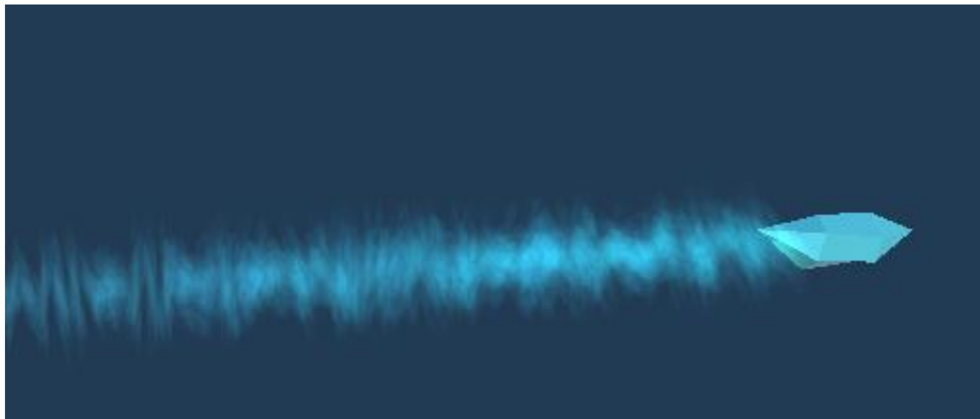
U metodi *FixedUpdateNetwork* koju su projektili naslijedili iz klase *NetworkBehaviour* projektili pomiču svoju poziciju u smjeru kretanja koji im je dan pri pozivu metode *Throw* za vrijednost vlastite varijable brzine. Ako se metoda izvršava na klijentu domaćinu koji ima zastavicu *HasStateAuthority* postavljenu na true provjerava se je li projektilu došao neki objekt u njegov domet. To se izvršava koristeći metodu *OverlapSphere Unity Phsysics* klase. Ta metoda kao rezultat vraća listu *Collider* komponenti pogođenih objekata. Za svaki pogodak se provjerava je li u pitanju drugi igrač različit od vlasnika projektila, te pripada li neprijateljskom timu. Ako je to slučaj poziva se *DealDamage* metoda *PlayerStats* [3.4.4] klase tog igrača kako bi mu se počinila šteta. Nakon toga ako je bilo koji od pogođenih objekata bio igrač ili dio terena uništava se game objekt projektila. Električni projektil ima samo osnovno ponašanje, dok ledeni i vatreni projektil imaju još svoja dodatna ponašanja. Ledeni projektil prilikom pogotka ne uništava odmah svoj projektil, nego nakon vremenske odgode. To se na računalima klijenata postiže pozivom *RPC_LockModelToPlayer* metode udaljene procedure, koja kao roditelj modela projektila postavlja pogođeni objekt. Također, prilikom pogotka protivničkog igrača poziva *ApplySlow* metodu pogođenog igrača koja uključuje usporeni način kretanja tog igrača, te kao parametar šalje vremensko trajanje usporenja. Vatreni projektil pri pogotku izvršava dodatnu eksploziju. Eksplozija (Kôd 3.7) ima svoj zasebni doseg koji je veći od dosega projektila, te izvršava istu logiku detekcije pogotka projektila. Detekcija se izvršava samo istom okviru igre kao originalni pogodak. Vizualni prikaz eksplozije se izvršava gašenjem game objekta vizualnog prikaza projektila i paljenjem game objekta vizualnog prikaza eksplozije. Pošto se detekcija pogotka izvršava samo na računalu igrača koji je stvorio projektil, kako bi se propagirala informacija o potrebi uključivanja vizualnog prikaza eksplozije poziva se *RPC_ExplodeEffect* metoda udaljenog poziva koja kao parametar odredišta svog *Rpc* atributa ima sve klijente kako bi se izvršila na računalima svih igrača u sobi.

```
private void Explode()
{
    _exploded = true;
    Debug.Log("Exploded");

    _projectileEffect.SetActive(false);
    _explosionEffect.SetActive(true);
    RPC_ExplodeEffect();

    Collider[] hitColliders = Physics.OverlapSphere(transform.position,
_explosionRange);

    foreach (Collider collider in hitColliders)
    {
        if (collider.tag != "Player") continue;

        PlayerStats player = collider.GetComponent<PlayerStats>();

        if (player.Object.InputAuthority ==
            OwnerPlayerStats.Object.InputAuthority) continue;
        if (FusionConnection.GameModeType == GameModeType.TDM && player.Team
            == OwnerPlayerStats.Team) continue;
        if (player == _hitPlayer) continue;

        player.DealDamage(_damage, OwnerPlayerStats);
    }

    Destroy(gameObject, _explosionDuration);
}
```

Kôd 3.7 *Explode* metoda klase *FireballProjectile* koja se poziva pri pogotku sa vatrenim projektilom



Sl. 3.8 Izgled efekta vatrenog projektila

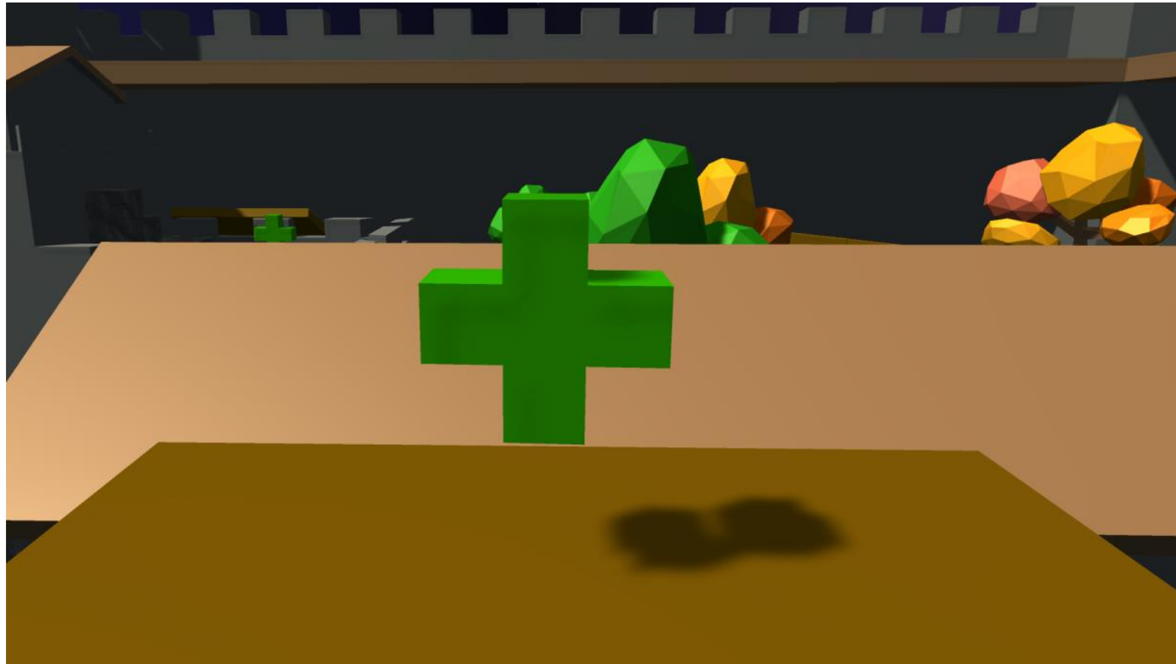


Sl. 3.9 Izgled efekta ledenog projektila

3.4.7. Objekti ozdravljenja

Objekte ozdravljenja stvara domaćin nakon stvaranja vlastitog lika za igranje. Klasa *HealingPointSpawner* tipa singleton sadrži listu lokacija na kojima treba stvoriti objekte ozdravljenja. Po pozivu metode *SpawnHealingPoints HealingPointSpawner* na svakoj lokaciji instancira novi objekt koristeći referencu na *HealingPoint* prefab pozivom metode *Spawn* instance *NetworkRunner* klase. *HealingPoint* objekt pri doticaju s igračem mu vraća životne bodove i onesposobljuje se na kratko vrijeme. Detekcija kolizije se odvija samo na domaćinu. Kako bi se ažurirao vizualni prikaz objekta ozdravljenja ovisno o stanju osposobljenosti koristi se umreženo svojstvo *RespawnTimeLeft*. *RespawnTimeLeft* pri promjeni vrijednosti poziva *CallBack* metodu *RespawnHealItemSequence* koji

ažurira osposobljenost objekta ovisno o preostalom vremenu. Preostalo vrijeme *RespawnTimeLeft* se ažurira od strane domaćina.



Sl. 3.10 Objekt ozdravljenja u mapi igre

3.4.8. GameManager

Klasa *GameManger* je umreženi objekt koji nasljeđuje klasu *NetworkBehaviour* Photon Fusion 2 biblioteke. Objekt ove klase se instancira samo jednom od strane domaćina, nakon što stvori vlastitog lika za igranje. Klasa *GameManager* se brine o bodovima timova, završetku runde nakon postignuća dovoljnog broja bodova i pamćenja pobjednika. Pobjednik se pamti kroz umrežena svojstva, što je važno za slučaj kada se novi igrač pridruži sobi za vrijeme završetka igre kako bi mogao prikazati točno ime pobjednika. Nakon što se detektira da je postignut dovoljan broj bodova od strane igrača ili tima za pobjedu u klasi *PlayerStats* [3.4.4] poziva se metoda *GameEnd* klase *GameManager*. Metoda *GameEnd* zabilježava ime pobjednika i započinje korutinu odbrojavanja kraja igre. Ta korutina na početku poziva metodu *SendGameEndRpc* klase *PlayerManager* [3.4.3] koja propagira poruke kraja igre metodama udaljenih procedura svim ostalim igračima, a na kraju odbrojavanja resetira podatke o bodovima i poziva metode *ResetPlayerStats* i *SendGameStartRpc* klase *PlayerManager* koje resetiraju borbene podatke igračima i obavještavaju ih o početku nove runde pozivima metoda udaljenih procedura.

3.5. Izrada potrebnih vizualnih sredstava

Za diplomski rad su izrađeni potrebni modeli i vizualni efekti. Modeli su izrađeni u programskom alatu Blender [12], dok su vizualni efekti izrađeni u sklopu Unity programa korištenjem *ParticleEffect* komponente. Izrađeni su modeli za zidine, kuće, drveće i bunare srednjovjekovne mape, te točke ozdravljenja. Izrađen je model viteza koji se koristi za prikaz igrača, te je animiran koristeći besplatne animacije iz Mixamo [13] biblioteke. Također su izrađeni vizualni efekti za vatreni, ledeni i električni projektil.

4. Izrada uputa za laboratorij

U sklopu projekta izrađeni su zadatci za učenje i vježbu, koji su namijenjeni za korištenje u laboratorijskim vježbama. Zadatci za vježbu se izvode na verziji projekta kojoj nedostaju dijelovi koda. Zadatak učenika je nadopuniti dijelove koda koji nedostaju. Početni zadatci služe za upoznavanje učenika sa kodom projekta i funkcionalnostima biblioteke Photon Fusion 2. Zadatci se sastoje od uvoda i tri zadatka za rješavanje.

U uvodu je dan pregled Photon Fusion 2 biblioteke, te su dane upute za postavljanje biblioteke u Unity projekt i registriranje na Photon Fusion servise. U prvom zadatku učenici nadopunjuju kod scene s izbornicima. Upoznaju se s klasama *RoomCreationView, SessionView* i *FusionConnection.* Kao rezultat bi trebali steći znanja o upravljanju elementima korisničkog sučelja scene s izbornicima i funkcionalnostima Photon Fusion 2 biblioteke za spajanje u sobe igre i povezivanje na Photon Fusion servise u oblaku. U drugom zadatku dan je pregled scene igre i potrebnih funkcionalnosti, te zadatci za nadopunjavanje koda. Učenici se u ovom zadatku upoznaju s klasama *GameManager, InputManager, FireballProjectile, PlayerManager, PlayerCharacterController i PlayerStats.* Kroz ovaj zadatak bi trebali steći znanja o upravljanju kretanja likova u umreženim igrama, pozivima metoda udaljenih procedura i korištenjem umreženih svojstava. U trećem zadatku se od učenika traži da implementiraju vlastiti tip borbene čarolije, kako bi na praktični način iskoristili i pokazali stečena znanja i kreativnost.

5. Rezultati

Kao rezultat ovog diplomskog rada izrađena je umrežena igra za više igrača pucanja u trećem licu. Igra ima početnu scenu glavnog izbornika. U glavnom izborniku korisnik unosi svoje ime, te može pregledati otvorene sobe koja se može pridružiti, može stvoriti novu sobu i može odabrati borbenu čaroliju. Pri izradi sobe bira između timskog načina igre i načina igre svatko protiv svakoga, te bira mapu na kojoj će se igra odvijati. Proširenje igre dodavanjem novih mapa i oružja je trivijalno, potrebno je samo napraviti vizualne prikaze za njih, te ih dodati u pripadnu *ScriptableObject* instancu.

U sceni igre učitava se odabrana mapa i stvara se lik igrača. Igrač se može kretati po terenu, vidjeti druge igrače te pucati projektile odabrane borbene čarolije. Prilikom pogotka projektila protivničkog igrača igraču se smanjuju životni bodovi. Igrač može skupiti objekte ozdravljenja koji mu vraćaju životne bodove. Ako životni bodovi dostignu 0 igrač se onesposobljuje na nekoliko sekundi, te se dodaju bodovi protivničkom igraču. U timskom načinu igre zbrajaju se bodovi svih igrača u timu. Ako jedan tim ili igrač dostigne dovoljni broj bodova za pobjedu igrači se onesposobljuju, prikazuje se ekran s imenom pobjednika, te se nakon par sekundi bodovi resetiraju, igrači postavljaju na novu nasumičnu poziciju i osposobljuju, te igra kreće ispočetka.

Sl. 5.1 Ekran pobjede tima

Pri izradi rada ključno je bilo naučiti funkcionalnosti poziva metoda udaljenih procedura i umreženih svojstava kako bi se moglo prenositi podatke unutar skripti između klijenata i domaćina i

mijenjati podatke na objektima nad kojima klijent nema autoritet stanja. Sinkronizacija pozicija objekata u sceni je trivijalna, jer je potrebno samo dodati komponentu *NetworkObject* i *NetworkTransform* čime Photon Fusion 2 biblioteka preuzima ulogu sinkronizacije, što je i glavni razlog korištenja takve biblioteke da makne s programera potrebu za implementacijom mrežnih algoritama i ubrza razvoj umreženih igara.

Projekt je razvijen u arhitekturi komunikacije klijent-poslužitelj. Ulogu poslužitelja ima domaćin, koji ujedno i sudjeluje u igri kao klijent. Domaćin služi kao centralni autoritet. Domaćin ima autoritet stanja nad svim objektima, a klijenti imaju autoritet naredbi nad objektom lika kojeg upravljaju, te domaćinu samo šalju svoje naredbe koje on potom izvršava. Prednosti ovog pristupa su prisustvo centralne logike i prisustvo centralnog autoriteta. Centralni autoritet osigurava od varanja sa strane ostalih klijenata, no ne osigurava od strane varanja klijenta koji igra ulogu domaćina. Centralna logika osigurava da permanentni mrežno dijeljeni elementi mape s kojima igraći mogu interaktirati neće nestati ako se neki od klijenata odspoji.

Photon Fusion 2 biblioteka za sinkronizaciju objekata klijentima šalje stanja, a domaćinu unose naredbi ostalih igrača. No zbog mrežnog kašnjenja stanja na klijentima nisu deterministička. To znači da fizička simulacija stanja ne mora biti ista na svim klijentima u svakom trenutku, što u kombinaciji s mrežnim dovodi do toga da prikaz stanja može biti različit za svakog klijenta, dok se ne sinkroniziraju sa stanjem poslužitelja. Problem postaje očiti što je mrežno kašnjenje veće. Prikaz razlike stanja na dva klijenta u *localHost* mreži se može vidjeti na priloženoj slici (Sl. 5.2). Za minimiziranja posljedica toga u igrama gdje je preciznost bitna, poput igara pucanja, Photon Fusion 2 biblioteka podržava izvođenje fizičkih zraka s kompenzacijom kašnjenja, koje unatoč kašnjenju mogu točno detektirati je li se trebao dogoditi pogodak ili ne. Kako bi se igračima poboljšalo korisničko iskustvo i minimizirale osjetne posljedice mrežnog kašnjenja koristi se tehnika predikcije klijentske strane. Klijenti lokalno izvršavaju naredbe korisnike, koje se također šalju na poslužitelja, uz pretpostavku da će primjenom istih naredbi simulacija stanja klijenta biti istog stanja kao izvođenje tih naredbi na poslužitelju. Poslužitelj nakon izvođenja naredbi šalje podatke o novom stanju klijentima. Klijenti nakon primanja podataka o novom stanju mogu ažurirati svoje trenutno stanje ako ne odgovara pristiglim podatcima s poslužitelja.

Sl. 5.2 Slika stanja igre na dva klijenta preko *localHost* mreže

Rezultat ovog diplomskog rada je i verzija Unity projekta igre sa nedostajućim dijelovima koda i uputama za učenje sa zadatcima za ispunjavanje istih.

Zaključak

Izrada umreženih video igara je zahtjevan posao. Uključuje kombinaciju različitih disciplina inženjerstva razvoja koda igara, dizajniranja elemenata igre i tehnika umreženih algoritama. Kako bi se olakšao razvoj takvih igara postoje biblioteke koje obavljaju mrežni dio razvoja igara. Te biblioteke izvode funkcionalnosti prenošenja podataka i sinkroniziranja stanja između igrača bez da programer koji razvija igru mora sam implementirati mrežne algoritme. Jedna od takvih biblioteka je Photon Fusion 2, namijenjena za implementaciju u Unity programskom alatu za izradu interaktivnih igara, koja je korištena u sklopu ovog rada.

Za ovaj diplomski rad je izrađena igra pucanja iz trećeg lica za više igrača. Igrači mogu unijeti svoje ime, te se spojiti u postojeću sobu iz izbornika otvorenih soba ili napraviti svoju sobu. Prije ulaska u sobu igrač bira između tri borbene čarolije čije projektile će koristiti za pucanje. Igre u sobama se mogu odvijati u timskom načinu ili svatko protiv svakoga. Igrač ili tim dobivaju bod kada spuste protivničkom igraču životne bodove na nula. Kada je dostignut dovoljni broj bodova, taj igrač ili tim se proglašavaju pobjednicima, te igra započinje ispočetka. Za igru su izrađeni svi potrebni modeli i efekti.

Korištenje Photon Fusion 2 biblioteke za razvoj umreženih igara je uvelike pomogao i ubrzao razvoj projekta. Biblioteka omogućava programerima da se fokusiraju na razvoj igre, a implementaciju mrežnih algoritama prepuste biblioteci. Photon Fusion 2 je preporučljiv za sve projekte kojima kompleksnost projekta spada u djelokrug biblioteke i kojima je razvoj igrajućih elemenata u većem fokusu neko razvoj mrežnih algoritama.

Literatura

[1] *Fusion 2 Introduction*, https://doc.photonengine.com/fusion/current/fusion-intro, pristupano 8.2.2024.

[2] *Quantum 2 Introduction*, https://doc.photonengine.com/quantum/current/quantum-intro, pristupano: 8.2.2024.

[3] *Unity Mirror*, https://assetstore.unity.com/packages/tools/network/mirror-129321, pristupano: 8.2.2024.

[4] *Unity Netcode for GameObjects*, https://docs-multiplayer.unity3d.com/netcode/current/about/, pristupano: 8.2.2024

[5] Nemec, F. *Development and application of a videogame multiplayer networking library*. Diplomski rad. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2022.

[6] *Fusion 2 Network Runner*, https://doc.photonengine.com/fusion/current/manual/network-runner, pristupano: 8.2.2024.

[7] *Fusion 2 Networked Properties*, https://doc.photonengine.com/fusion/current/manual/data-transfer/networked-properties, pristupano: 8.2.2024.

[8] *Fusion 2 Remote Procedure Calls*, https://doc.photonengine.com/fusion/current/manual/data-transfer/rpcs, pristupano: 8.2.2024.

[9] *Fusion 2 Network Behaviour*, https://doc.photonengine.com/fusion/current/manual/network-behaviour, pristupano: 8.2.2024.

[10] *Fusion 2 Network Object*, https://doc.photonengine.com/fusion/current/manual/network-object, pristupano: 8.2.2024.

[11] *Fusion 2 RPC Method Parameters*, https://doc.photonengine.com/fusion/current/manual/data-transfer/rpcs#rpc-method-parameters, pristupano: 8.2.2024.

[12] *Blender*, https://www.blender.org/, pristupano: 8.2.2024.

[13] *Adobe Mixamo*, https://www.mixamo.com/#/, pristupano: 8.2.2024.

Sažetak

Za diplomski rad je izrađena umrežena igra za više igrača u programskom alatu za izradu igara Unity. Za implementaciju mrežnih algoritama korištena je biblioteka Photon Fusion 2. Izrađena je scena početnog menija za unos imena, ulazak u postojeće ili stvaranje novih soba i odabir borbene čarolije. Izrađena je i scena igre pucanja iz trećeg lica gdje igrači mogu igrati u timovima ili svatko za sebe. Za igru su izrađeni potrebni modeli i vizualni efekti. Napravljene su dvije verzije Unity projekta, cjelovita verzija i verzija kojoj fale dijelovi koda uz koju dolaze upute za učenje sa zadatcima.

Ključne riječi: Unity, Photon Fusion, umrežene igre, video igre, mrežni algoritmi

Summary

A networked multiplayer game was created for this master's thesis using the Unity game engine. The Photon Fusion 2 library was used to implement the network algorithms. Starting menu scene was created for in which users can enter names, they can join existing or creating a new game room, and select a battle spell they will use. A third-person shooter game scene was also created where players can play in teams or individually. The necessary models and visual effects have been created for the game. Two versions of the project were created, a complete version and a version with missing parts of the code, which also comes with a learning manual with assignments.

Keywords: Unity, Photon Fusion, mutliplayer games, video games, network algorithms

Privitak

Izrađeni materijali za učenje zasnovani na ovome diplomskom radu

Diplomski studij

Računarstvo

Znanost o mrežama

Umrežene igre

Upute za izradu 5. laboratorijske vježbe

SADRŽAJ

Contents

UVOD 1

Što treba predati 1

ZADATAK 1 5

ZADATAK 2 8

ZADATAK 3 11

UVOD

Peta laboratorijska vježba vezana je uz izradu umrežene igre gađanja iz prvog lica.

Studenti dobivaju kompletan projekt te ga trebaju nadopuniti. Projekt se temelji na Photon biblioteci Photon Fusion 2. Igra se sastoji od scene početnog izbornika LoginScreen i dvije scene mapa za igru: CastleStadium i DesertStadium. U svakoj od ovih scena potrebno je nadopuniti određene skripte kako bi višekorisnička igra radila.

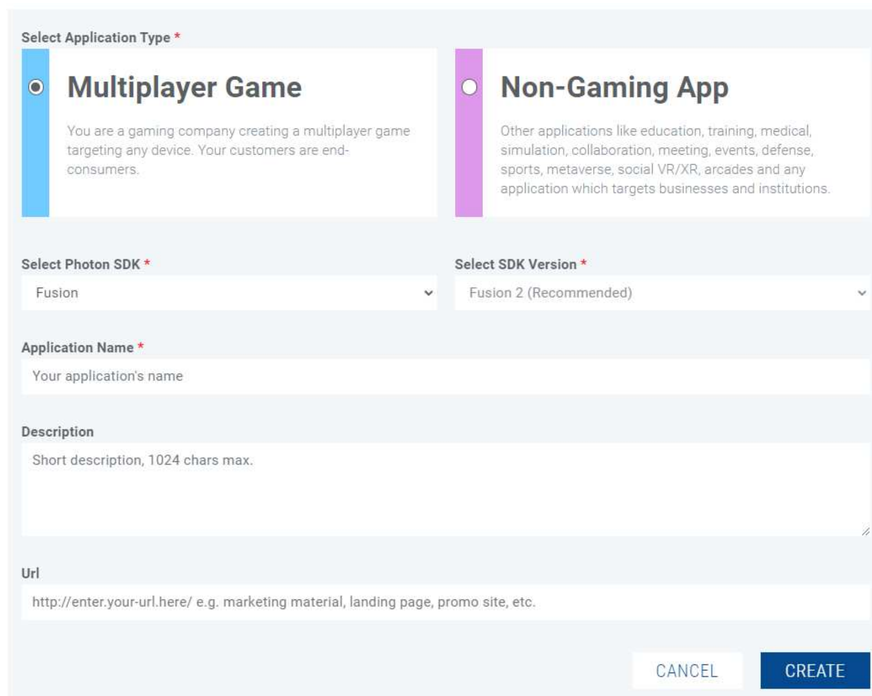
Što treba predati

Na Moodle je potrebno predati kratki izvještaj koji treba sadržavati:

- kratki opis **personaliziranog elementa** po vlastitom izboru,
- poveznicu na kompajlirani build za Windows (možete ga postaviti na primjerice svoj OneDrive ili bilo kakav drugi sustav za dijeljenje datoteka) i
- Unity projekt dovršene implementacije (možete ga postaviti na primjerice svoj OneDrive). Dodatno, slobodno izbrišite mape i datoteke navedene na linku https://github.com/github/gitignore/blob/main/Unity.gitignore (napravite si
  - prethodno kopiju cijelog projekta kako ne biste slučajno izbrisali mapu poput Assets itime izgubili odrađeni posao) kako bi znatno smanjili veličinu implementacije za upload. Mape koje trebaju ostati jesu: Assets, Packages, ProjectSettings.

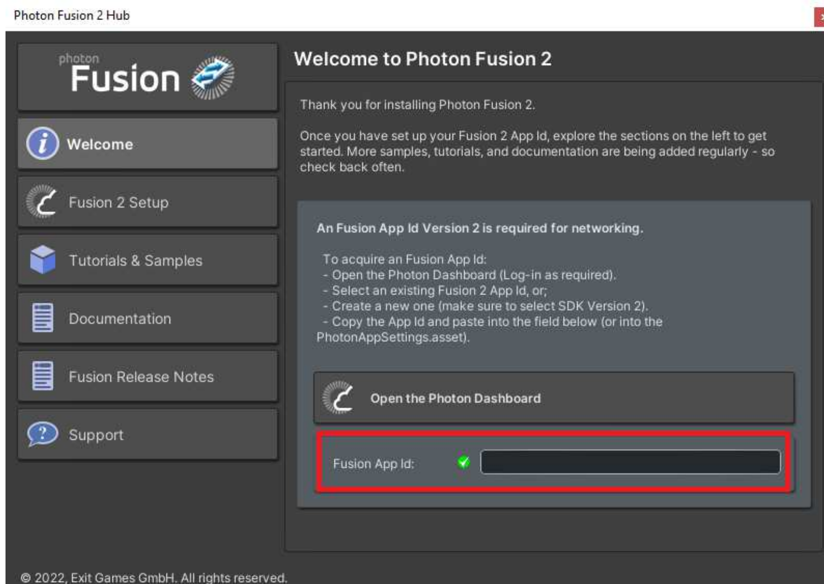
POSTAVLJANJE FUSION 2 i PHOTON POSLUŽITELJA

Kako bi bilo moguće korištenje *Photonovih* poslužitelja, prvo je potrebno stvoriti aplikaciju na *Photon* stranici (https://www.photonengine.com/). Potrebno je registrirati se i odabrati izbornik *Dashboard*. Tamo se nalazi pregled svih postojećih aplikacija – cilj je za svaku igru ili svaki projekt koji koriste Photon stvoriti novu aplikaciju na *Dashboardu*. Prilikom izrade nove aplikacije (*Create a new app*) potrebno je odabrati *Multiplayer Game* kao tip aplikacije, u SDK padajućem izborniku odabrati *Fusion*, te u padajućem zborniku verzije odabrati *Fusion 2* (Slika 1).

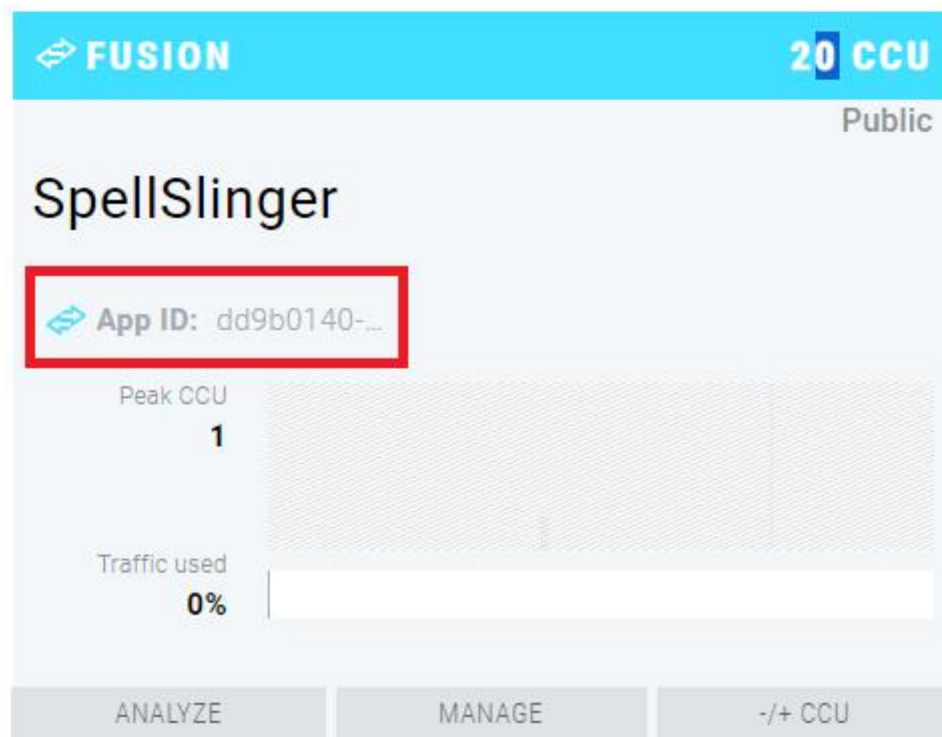


Slika 1 Stvaranje aplikacije

Nakon stvaranja aplikacije, na Dashboardu se sada prikazuje novonastala aplikacija i potrebno je kopirati cijeli App ID aplikacije (slika 2). Sada je Unity projektu potrebno odabrati *Tools* → *Fusion* → *Fusion Hub*, čime se otvara Photon Fusion 2 Hub izbornik. U izborniku u polje Fusion App Id potrebno je unijeti ranije stvoreni kod.



Slika 2 Unos App Id podatka u Photon Fusion 2 Hub izbornik



Slika 3 Nova aplikacija na Dashboardu

Pomoćni materijali uključuju :

Photon Fusion 2 dokumentaciju :

https://doc.photonengine.com/fusion/current/fusion-intro

ZADATAK 1

U sceni s izbornicima, korisnik ima mogućnost unijeti svoje ime, stvoriti novu sobu, pregledati postojeće sobe i pridružiti se nekoj od njih, odabrati borbeno oružje ili izaći iz igre. Objekti svih menija su djeca objekta Canvas. Canvas sadrži tri izbornika – LoginMenu, Sessions i RoomCreator. Za početak pogledajte od čega se sastoje ti izbornici i čemu služe.

Izbornikom LoginMenu se upravlja skriptom *LoginView*, izbornikom Session se upravlja skriptom *SessionView*, a izbornikom *RoomCreation* se upravlja skriptom *RoomCreationView*. Svaka skripta je priključena na pripadni izbornik, proučite sadržaje skripti i kako se koriste.

Konfiguracijske informacije za borbene čarolije i mape za igranje nalaze se u instancama *ScriptableObject* skripti. Te skripte su *WeaponTypeScriptable* i *WeaponDataScriptable*, a njihove instance se nalaze u direktoriju Assets/_ScriptableObjectInstances. Proučite koje podatke sadrže i koje metode za dohvat podataka podržavaju.

Za obavljanje zadataka komunikacije s Photon Cloud servisima, poput dohvaćanja popisa otvorenih soba, koristi se skripta FusionConnection priključena na istoimeni objekt u sceni. Skripta *FusionConnection* nasljeđuje sučelje *INetworkRunner* i implementira metode tog sučelja.

Klase *SessionView* i *FusionConnection* su izrađene u *Singleton* obliku. *Singleton* obrazac označuje da uvijek može postojati samo jedna instanca objekta te klase, te im se zbog toga može pristupati njihovom statičkom referencom. U ovom projektu *Singleton* skripte se mogu prepoznati po tome što nasljeđuju klasu *Singleton* i pristupa im se preko njihove reference u oblike *(ImeKlase).Instance*. Metoda *CreateSession* klase *FusionConnection* se poziva iz metode *CreateRoom* klase *RoomCreationView,* te se metoda *UpdateSessionList* poziva iz metode *OnSessionListUpdated* klase *FusionConnection* od strane Photon Fusion 2 biblioteke, te se preporuča njihovo popunjavanje u redoslijedu pozivanja.

U ovoj sceni potrebno je popuniti skripte i metode:

- *SessionView*,
  - *UpdateSessionList.*
- *RoomCreationView*,
  - *Awake,*
  - *CreateRoom.*
- *FusionConnection*,
  - *CreateSession,*
  - *LeaveSession,*
  - *OnSessionListUpdated.*

U svakoj metodi se nalaze upute za nadopunjavanje. Prijedlog je da nakon nadopunjavanja

svake metode provjerite radi li kao što bi trebala. U slučaju da se radi o pregledima soba i korisnika, otvorite projekt u još jednoj instanci Unity-a i simulirajte tako više korisnika i provjerite da li se informacije sinkroniziraju. Slijede tri primjera za bolje razumijevanje zadataka.

Primjer 1 – metoda *CreateSession* skripte *FusionConnection*

Ova metoda se poziva prilikom stvaranja nove sobe. Metoda postoji i samo ju je potrebno nadopuniti unutar vitičastih zagrada. Unutar njih nalazi se komentar koji navodi na ono što upravo nedostaje u ovoj metodi. U ovoj metodi potrebno je dodati kôd koji će stvoriti sobu i postaviti korisnika kao domaćina na Photon poslužitelju.

Igra se odvija u *Host* načnu, što znači da je format komunikacije klijent-poslužitelj. Ulogu poslužitelja obavlja igrač koji stvara sobu i on se smatra domaćinom (engl. *Host).* Pozivanjem metode *StartGame NetworkRunner* stvara se soba. Primjer pridruživanja sobi pozivom te metode može se vidjeti u metodi *JoinSession.*

```
public async void CreateSession(string sessionName, GameModeType gameMode, LevelType
level)
{
    /* U ovoj metodi potrebno je lokalno cache-irati odabrani način igre, te pozvati
       metodu StartGame NetworkRunner instance koja igrača spaja u sobu.
     * StartGame metoda prima argument tipa strukture. Potrebno je napraviti novu
       instancu strukture, te joj inicijalizirati vrijednosti.
     * Potrebno je postaviti način igre na Shared, proslijediti ime sesije, scenu
       koja se treba učitat nakon spajanja u sobu (parametar se
     * predaje u obliku SceneRef.FromIndex()), maksimalni broj igrača, scene manager iz
       lokalne reference i SessionProperties.
     * U SessionProperties ulaze custom svojstva, u ovom slučaju su to tip igre i level
       koji se treba učitati. Proučite kojeg tipa je
     * SessionPropeties, te mu proslijedite sva potrebna svojstva. (tip: za pretvaranje
       custom svojstva u pogodan oblik može se koristiti
     * SessionProperty.Convert() metoda.
     */
}
```

Za opis postojećih metoda tu je Photon-ova dokumentacija kojoj možete pristupiti pomoću poveznica u uvodu vježbe s pomoćnim materijalima.

Primjer 2 – metoda *UpdateSessionList* klase *SessionView*

```
public void UpdateSessionList()
{
    /* U ovoj metodi potrebno je očistiti lokalnu listu prikaza sesija i uništiti
       njihove game objekte.
     * Potom koristeći listu sesija koja se dohvaća iz Singleton instance klase
       FusionConnection je potrebno osvježiti listu.
     * Za svaku postojeću sesiju potrebno je stvoriti novu instancu SessionDataView
       prefab-a, pozvati njenu metodu za prikazivanje i dodati u lokalnu listu.
     * Za roditelja tih objekata potrebno je postaviti lokalnu referencu na kontejner
       sesija.
     * Korisnik u sceni može pritisnuti instance SessionDataView objekata, te ih time
       odabrati za pridruživanje pritiskom na tipku Join.
     * Stvaranje objekata se provodi na sličan način kao stvaranje
       WeaponSelectionToggle objekata iz metode Awake.
     */
}
```

U metodi *UpdateSessionList* (kôd 2)potrebno je pregledati hijerarhiju scene i uz objašnjenje iz komentara ostvariti rad objekata u sceni tj. na ekranu. Potrebno je koristiti se već postojećim objektima (engl. *prefab*), njihovim komponentama i skriptama. Kod ostalih objekata i komponenta nije potrebno dodavati nove metode ili funkcionalnosti, već samo iskoristiti postojeće da bi se postigao cilj.

Popunjavajući metode redoslijedom kako se nalaze u skripti je najlakši način. Neki komentari, poput komentara u metodi *UpdateSessionList*, referenciraju neke metode koje se nalaze drugdje u kodu u kojima je prethodno riješen sličan problem.

Primjer 3 – metoda *Awake* klase *RoomCreationView*

```
private void Awake()
{
    /* U ovoj metodi potrebno je inicijalizirati funkcionalnosti izbornika.
     * Prvo je za svaki podatak o levelu iz instance klase LevelDataScriptable
       stvoriti novi
     * gumb za odabir levela koristeći prefab LevelSelectionToggle objekta.
     * Stvorene objekte je potrebno inicijalizirati potrebnim parametrima.
     * Za parametar akcije selektira potrebno je poslati lambda izraz koji tip
       levela tog gumba
     * lokalno sprema kao odabrani tip levela.
     *
     * Nakon toga je potrebno inicijalizirati callback Return gumba da na pritisak gasi
     * izbornik stvaranja sobe i otvara izbornik odabira otvorenih soba.
     * Također potrebno je inicijalizirati callback Create Room gumba da na pritisak
     * poziva metodu za stvaranje sobe.
     */

}
```

ZADATAK 2

Nakon što je uspješno popunjen kod spajanja u sobe i započinjanja igre – igrači mogu započetiigrati igru. Igra se odvija u sceni za igru. Scena za igru koja se učitava ovisi o izboru mape u izborniku za izradu igara, ili prikaza na kojoj mapi se odvija soba u izborniku otvorenih soba. Prilikom učitavanja scene i ulaženja u sobu klasa *FusionConnection* za svakog korisnika stvara objekt *PlayerCharacterController* – objekt koji sadrži vizualni prikaz igrača, te omogućuje njegovo kontroliranje. Taj objekt također sadrži komponentu *PlayerStats* koja se brine o igračevim podatcima unutar logike igre. U sceni se također nalazi objekt s *PlayerManager* skriptom koja upravlja referencama o igračima u sceni, te postavlja timske boje na igrače. Igrač domaćin na početku igre stvara instancu skripte *GameManager* koja prati timske bodove i brine se za logiku završetka i započinjanja rundi.

Svaki igrač ima svoje zdravlje i svoj u borbenu čaroliju, čijim projektilima može gađati druge igrače. Ako igrač pogodi drugog igrača, drugi igrač gubi zdravlje (ovisno o jačini oružja), te ako je izgubio svo zdravlje onesposobljava se na neko vrijeme, te se ponovno stvara negdje u sceni. Pozicije stvaranja se dobivaju iz skripte *SpawnLocationManager,* koji ima listu referenci na prazne objekte u sceni koje je potrebno postaviti na željena mjesta, između kojih bira nasumično. Svaki igrač na svojem ekranu u donjem desnom kutu vidi svoje zdravlje. Igrači si mogu vraćati zdravlje skupljajući objekte ozdravljenja u sceni. Objektima ozdravljenja upravlja pripadna skripta *HealingPoint,* te se na početku stvaraju u sceni preko skripte *HealingPointSpawner.*

Igrači se po mapi mogu kretati koristeći WASD tipke na tipkovnici, mogu skakati koristeći tipku *Space* (igračima je omogućen dupli skok), te pucati prilikom pritiska na lijevu tipku miša. Okretanjem gumba *Scroll* igrač mijenjati *zoom* razinu kamere. Svaki put kada igrač nekoga pogodi i oduzme mu zadnje zdravlje (tj. ubije neprijateljskog igrača), u ljestvici bodova napadaču se povećava broj ubijenih neprijatelja (engl. *kills*), a igraču koji je izgubio zdravlje se zbraja uništavanje (engl. *deaths*). Također u timskom načinu igre (engl. *TeamDeathMatch)* killovi svakog igrača se zbrajaju u timske bodove. Na vrhu ekrana u timskom načinu igre prikazuju se bodovi svakog tima, a u načinu igre svatko protiv svakoga (engl. DeathMatch) prikazuju se vlastiti bodovi. Držanjem tipke *tab* prikazuje se platno s prikazom dobivenih i danih bodova svih igrača. Pritiskom tipke alt oslobađa se kontroliranje miša i onesposobljuje se okretanje lika, dok ponovni pritisak poništava tu radnju. Pritiskom na gumb *Pause* otvara se izbornik pauze za podešavanje osjetljivosti okretanja mišem, te napuštanje sobe i povratak u početni ekran pritiskom na tipku *Leave*. UI elementima scene upravlja skripta UiManager.

Pošto se igra izvodi u načinu komunikacije klijent-poslužitelj, samo domaćin ima autoritet (*HasStateAuthority* zastavica tog objekta mu je postavljeno na *true)* promjene pozicije objekata u sceni i stanja umreženih svojstava (varijable sa

atributom [*Networked*] čije promjene Photon Fusion 2 propagira svim igračima u sobi). Ostali klijenti samo imaju autoritet unosa naredbi (*HasInputAuthority* zastavicu postavljenu na *true)* nad objektima vlastitih likova. Klijenti svoje naredbi obrađuju lokalno kako bi se oslabio osjećaj kašnjenja mreže, te ih šalju domaćinu preko *InputManager* skripte. Ako igrač želi promijeniti vrijednost nekog umreženog svojstva, to mora napraviti pozivom metode udaljene procedure koja se izvodi na računalu domaćina.

Slika 4 Scena igre

Za ovaj dio vježbe potrebno je znati kako funkcioniraju pozivi udaljenih procedura (engl. *Remote Procedure Calls*, RPC) i umrežena svojstva (engl. *Network Properties).*

U ovoj sceni potrebno je nadopuniti sljedeće skripte i metode:

- *PlayerCharacterController,*
  - *FixedUpdateNetwork,*
  - *RespawnCoroutine,*
  - *RPC_PlayerKilled.*
- *PlayerManager,*
  - *GetTeamWithLessPlayers.*
- *PlayerStats,*

    - *HealthChanged,*
    - *KillsChanged,*
    - *DealDamage,*
    - *ApplySlow.*
- *FireballProjectile,*
    - *Explode,*
    - *ExplodeEffectRpc.*
- *GameManager,*
    - *GameEndCountdown.*
- *InputManager,*
    - *BeforeUpdate.*

ZADATAK 3

Nakon su popunjene sve metode u projektu koje nedostaju i igra je spremna za korištenje, vaš zadatak je izraditi novi tip borbene čarolije. Sami osmislite dizajn ponašanja tog projektila (npr. vatreni projektil eksplodira nakon kontakta, ledeni projektil usporava igrača na neko vrijeme prilikom pogotka). Za kod projektila napravite novu skriptu (*tipProjektila*)Projectile.cs, te ju popunite po uzoru skripti za preostale projektile. U projektu je pripremljen *prefab* GroundProjectileModel (Slika 4) i *sprite* EarthImage, koje možete koristiti za izradu zemljanog projektila. Te *assete* možete iskoristi za neku drugu vrstu projektila koja ne mora nužno biti zemljani ili izraditi vlastite *assete* po izboru. Ne zaboravite da se potrebne reference o projektilu moraju dodat u instancu WeaponDataScriptable skripte kako bi se projektil mogao koristiti u igri.

Slika 5 Model zemljanog projektila