Virtualna okruženja

Laboratorijske vježbe

Vježba 3

Umrežena virtualna okruženja

FER – ZTEL – Igor S. Pandžić, Mirko Sužnjević

Suradnja na pripremi vježbe:

Robert Holovka

Sadržaj

1. Uvod

Tema ove vježbe su umrežena virtualna okruženja (UVO). UVO su sustavi koji omogućavaju da više fizički udaljenih korisnika sudjeluje u zajedničkom virtualnom okruženju. U ovoj vježbi upoznat ćete se s pojmovima karakterističnima za takve sustave poput skalabilnosti, konzistentnosti i karakteristikama mrežnog prometa. Kroz pokazni primjer ćete vidjeti kako ostvariti navedene pojmove i kako se nositi s nedostacima mreže na koje su ovakvi sustavi osjetljivi. Pokazni primjer je višekorisnička umrežena igra koja predstavlja jednu vrstu UVO.

Vježba se sastoji od dva dijela. Prvi dio vježbe mogu raditi dva studenta ili možete raditi sami ako imate dodatni monitor ili računalo. Ako odlučite raditi sami, jedini uvjet je da na oba ekrana broj slika u sekundi (engl. *Frames Per Second* - FPS) bude barem 50. Drugi dio vježbe se obavlja u grupi od 3-5 studenata.

2. Alati potrebni za izvođenje vježbe

Za odrađivanje osnovnog dijela vježbe nisu potrebni nikakvi dodatni alati.

Za pregledavanje projekta ili odrađivanje naprednog dijela potrebno je skinuti besplatnu verziju Unity aplikacijskog sustava. Aplikacija je izrađena s verzijom 2019.2.6f1, no projekt je moguće otvoriti i s bilo kojom kasnijom verzijom.

3. Teorijska podloga

Umrežena virtualna okruženja su sustavi koji omogućavaju da više fizički udaljenih korisnika sudjeluje u zajedničkom virtualnom okruženju. Korisnicima je omogućena komunikacija i interakcija s drugim korisnicima, te interakcija sa samim virtualnim okruženjem i predmetima koji se nalaze u njemu. Svaki korisnik ima vlastiti prikaz unutar virtualnog okruženja koji se zove *avatar*. Prikaz avatara može biti raznolik, od jednostavnih geometrijskih oblika pa sve do složenih humanoidnih modela visokih razina detalja.

UVO su specifični jer svako računalo posjeduje svoju, lokalnu kopiju okruženja kojom može upravljati. Sve promjene napravljene unutar pojedine kopije se porukama moraju dojaviti svim ostalim računalima. Promjene lokalnih kopija se sinkroniziraju putem mreže i u idealnom slučaju svi korisnici imaju identičnu kopiju u svakom trenutku.

3.1. Arhitektura

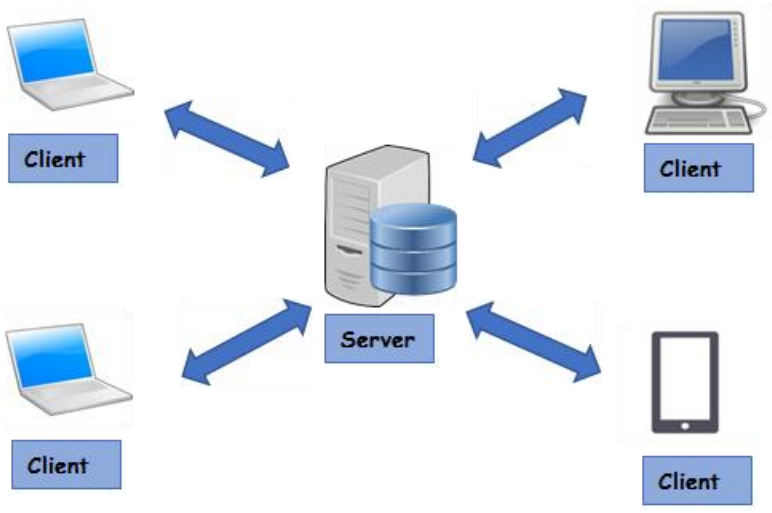
Umrežena virtualna okruženja mogu istovremeno podržavati veliki broj korisnika i zbog toga često imaju problema sa skalabilnošću. **Skalabilnost** je mjera koja govori koliko sustav može rasti, a da ne naruši korisnikovo viđenje istog. Skalabilnost sustava se može donekle postići odabirom ispravne arhitekture. Odabir arhitekture ovisi o tipu i namjeni UVO gdje treba voditi računa o njenim prednostima i nedostacima.

Programska arhitektura određuje logičku strukturu sustava, njegove programske komponente, njihovu organizaciju i način na koji te komponente komuniciraju. Osim skalabilnosti, prilikom odabira arhitekture treba voditi računa i o njezinoj pouzdanosti, sigurnosti i podršci za perzistenciju podataka. **Pouzdanost** predstavlja koliko je arhitektura otporna na ispade poslužitelja. **Sigurnost** govori koliko dobro arhitektura može prepoznati i kako se nosi s varanjem. **Perzistencija** predstavlja trajnu pohranu stanja.

Dva osnovna primjera arhitekture su klijent-poslužitelj i ravnopravni procesi. Ostali primjeri su većinom nadogradnje ili kombinacija ta 2 primjera te u ovoj vježbi neće biti razmatrane.

3.1.1. Klijent-Poslužitelj

Arhitektura klijent-poslužitelj (engl. *Client-Server*) je način oblikovanja sustava na kojem se zasniva većina današnjeg weba. Svi korisnici su spojeni na jednu središnju točku koja se zove poslužitelj (engl. *Server*). Svaka promjena od strane korisnika šalje se do te središnje točke, na njoj se računa novo stanje virtualnog svijeta i razašilje dalje ostalim korisnicima koji su spojeni na nju.



Slika 3.1: Arhitektura klijent-poslužitelj (*preuzeto sa* www.toolsqa.com)

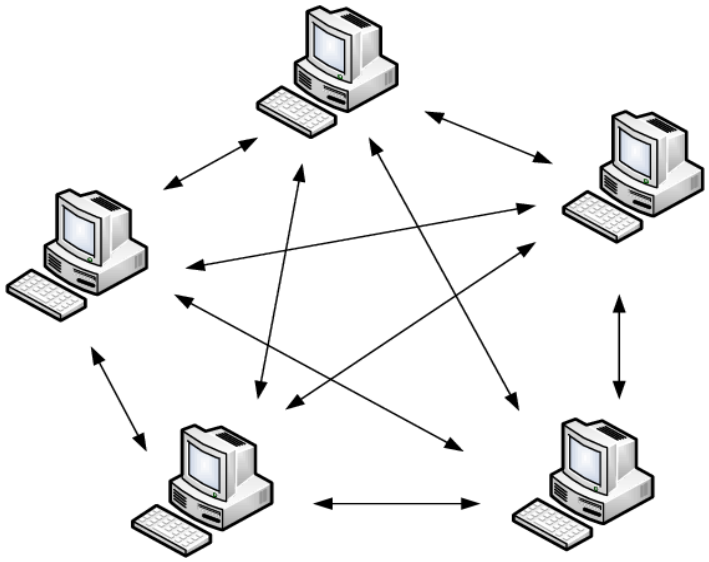
Glavne prednosti su sigurnost i olakšana perzistencija podataka. Poslužitelj je u stanju provjeriti da li su događaji koji su generirali sami korisnici validni i tako detektirati varanje. Dodatna prednost je i što poslužitelj može rasteretiti korisnike sa slabim resursima te umjesto njih vršiti nekakve kompleksne operacije i putem mreže im

dostaviti rezultat. Npr. filtriranje mrežnog prometa prema području interesa može biti izvedeno na samom poslužitelju.

Nedostatak je u tome što poslužitelj može postati usko grlo (engl. *Bottleneck*) ako se spoji mnogo korisnika. Razmjena svih informacija se provodi kroz centralnu točku i poslužitelj može postati preopterećen. Poslužitelj predstavlja i jedinstvenu točku ispada te ispadom poslužitelja sustav prestaje raditi. Iz navedenog se vidi da kod ovakve arhitekture skalabilnost i pouzdanost su loši.

3.1.2. Ravnopravni procesi

Ravnopravni procesi ili entiteti (engl. *Peer*) su računala koja ujedno mogu biti i klijent i poslužitelj. Ravnopravni procesi tvore sustav u kojem međusobno komuniciraju (engl. *Peer-to-Peer* - P2P) tako da se na aplikacijskom sloju povezuju u „prekrivajuću mrežu“ (engl. *Overlay Network*) čvorova ravnopravnih sudionika.



Slika 3.2: Osnovna inačica P2P mreže (*preuzeto sa* www.semanticscholar.org)

Ovakva vrsta arhitekture ima značajne prednosti u odnosu na klijent-poslužitelj što se tiče pouzdanosti i skalabilnosti. Pouzdanost je ostvarena time što ne postoji jedinstvena točka ispada. Ispad jednog čvora ima jako mali utjecaj na P2P mrežu zbog decentralizirane strukture. Skalabilnost je bolja jer ne postoji centralna točka kroz koju prolaze svi podaci i time se rješava problem nastanka uskog grla kada se spoji mnogo korisnika. Nedostatak osnovne inačice P2P mreže je u tome što sigurnost ovisi o tome koliko će sami korisnici biti pošteni, perzistencija podataka ne postoji i nema središnje točke koja bi ostvarila filtriranje mrežnog prometa.

3.2. Mrežni promet

Internet postaje sve brži i dostupniji, no i dalje ostaje problem kašnjenja i gubitka paketa. Kašnjenje unutar mreže i vjerojatnost gubitka paketa razmjerno raste s fizičkom udaljenosti između korisnika i poslužitelja zaduženim za održavanje stanja

UVO. Kod UVO mrežom se prenose raznorazne vrste podataka koji su osjetljivi na spomenute nedostatke.

Razmjena informacija se zasniva na standardnim IP komunikacijskim protokolima. Komunikacijski protokoli definiraju pravila komunikacije među procesima. Dva najčešće korištena protokola su TCP (engl. *Transmission Control Protocol*) i UDP (engl. *User Datagram Protocol*). Glavna razlika je u tome što je TCP pouzdan mehanizam koji jamči dostavu paketa na odredište dok UDP to ne garantira.

TCP je konekcijski protokol jer procesi prvo moraju razmijeniti kontrolne poruke, tj. uspostaviti vezu ili asocijaciju. TCP također takvu vezu održava aktivnom neko vrijeme, jamči redoslijed pristiglih poruka i radi retransmisiju ako dođe do gubitka paketa.

UDP je beskonekcijski protokol gdje se odmah prenosi sadržaj poruke. UDP nije pouzdan protokol, ne jamči ispravan redoslijed pristiglih poruka i ne radi retransmisiju ako dođe do gubitka paketa.

Upravo radi osiguravanja pouzdanosti TCP je znatno sporiji protokol. Uporabom tog protokola troši se vrijeme na uspostavljanje veze, potrebno je više mrežnih resursa i veličina zaglavlja paketa je veća. Prilikom dizajniranja sustava potrebno je odrediti koja vrsta prometa je kritična za dostavu i na temelju toga odlučiti koji će se protokol koristiti. Neke informacije se šalju dovoljno često pa njihov gubitak možemo zanemariti koristeći UDP protokol koji je brži.

Mrežni promet u UVO možemo podijeliti na:

- učitavanje,
- poruke sustava,
- događaji,
- osvježavanje stanja,
- tekst,
- zvuk i
- video.

Učitavanje je prijenos informacija o virtualnim predmetima poput tekstura, modela ponašanja i prikaza avatara. Poruke sustava su uglavnom vezane za kontrolu sjednice, tj. ulazak i izlazak korisnika iz sustava. Za zvuk i video je prikladan protokol RTP (engl. *Real-time Transport Protocol*) koji je napravljen za prijenos podataka medija u stvarnom vremenu.

Sinkronizacija lokalnih kopija se događa pomoću događaja i poruka osvježavanja stanja. Te dvije vrste poruka mijenjaju samu virtualnu scenu i informacije o samim predmetima unutar nje. Glavna razlika je u tome što se osvježavanje stanja odvija periodično, više puta u sekundi i radi se za sve promjenjive objekte unutar virtualnog prostora dok se događaji pojavljuju povremeno i imaju trajni utjecaj na scenu. Poruke osvježavanja konstantno pristižu, stoga njihov gubitak i nije toliko kritičan koliko i gubitak poruke o događajima koje mogu predstavljati dodavanje ili uništavanje nekog

objekta unutar scene. Tablica 1 prikazuje odabir transportnog protokola s obzirom na vrstu sadržaja.

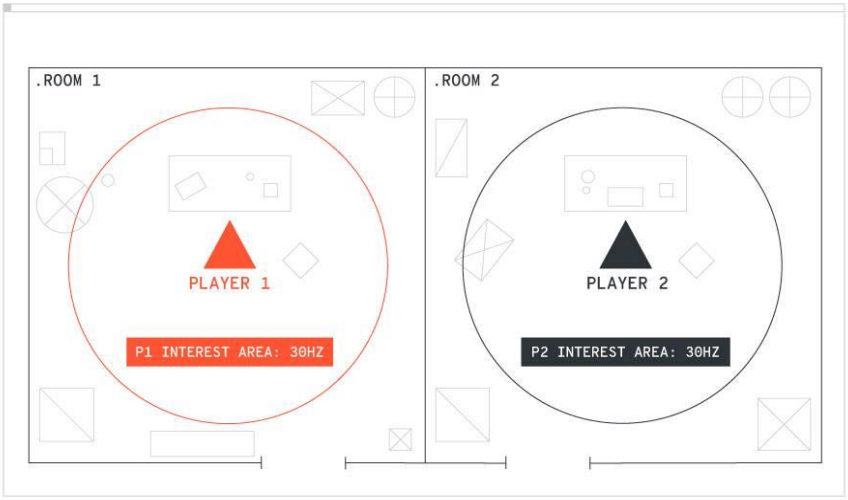
Tablica 3.1: Odabir transportnog protokola s obzirom na vrstu sadržaja

(*preuzeto iz* Pandžić, I. S; Pejša, T; Matković, K; Benko, H; Čereković, A; Matijašević, M. (2011). Virtualna Okruženja: Interaktivna 3D Grafike i Njene Primjene. Zagreb: Element)

Vrsta prometa	**Količina podataka**	**Potrebna pouzdanost**	**Prikladni protokol**
Učitavanje	Velika	Visoka	TCP
Poruke sustava	Mala	Visoka	TCP
Događaji	Mala	Srednja	TCP
Osvježavanje stanja	Srednje	Niska	UDP
Tekst	Mala	Srednja	TCP
Zvuk	Srednje	Niska	UDP (RTP)
Video	Velika	Niska	UDP (RTP)

3.3. Filtriranje mrežnog prometa i strukturiranje virtualnog svijeta

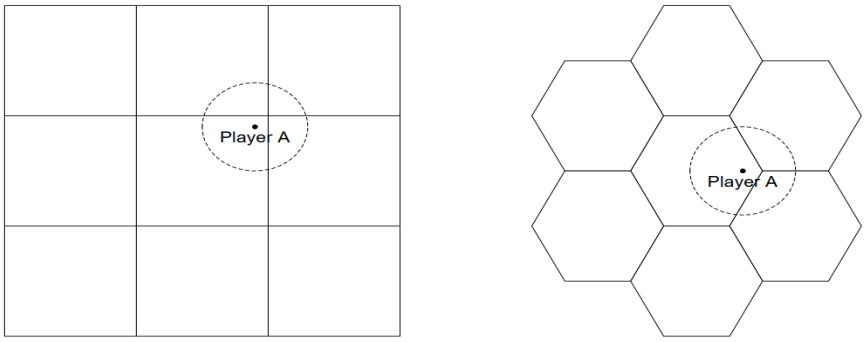
Jedan od načina da se smanji opterećenje mreže je da se ujedno smanji i količina podataka koja se šalje prilikom osvježavanja stanja. Filtriranje prema području interesa (engl. *Area Of Interest Management* – AOIM) je jedna od glavnih tehnika za održavanje skalabilnosti sustava, a temelji se na ideji odbacivanja informacija koje u nekom trenutku nisu bitne za pojedinog korisnika. Svaki korisnik ima dio kojeg ga zanima, recimo kružnicu koja predstavlja njegovo vidno polje i onda mu pristižu poruke osvježavanja samo vezano za objekte koji se nalazi u tom polju.



Slika 3.3: Interesno područje korisnika (*preuzeto sa* www.improbable.io)

Prednost je u tome što je ova metoda jednostavna za izvedbu i smanjuje mrežni promet. Nedostatak je u tome što poslužitelj treba periodično raditi kalkulaciju trenutnog područja interesa za svakog korisnika pojedinačno. S obzirom na to da UVO mogu imati veliki broj korisnika računanje AOIM za svakog pojedinog korisnika dodatno opterećuje poslužitelje.

Drugačiji pristup filtriranju prometa je podjela virtualnog svijeta na zone/komadiće/ćelije. Podjela virtualnog svijeta se može kategorizirati kao metoda upravljanja interesom. Korisnik prima samo poruke iz svoje i susjedne jedinice i na taj se način smanjuje mrežni promet. Virtualni prostor se može raskomadati na jednolične i slobodne geometrijske strukture. Kod jednolične geometrijske strukture virtualni svijet je podijeljen na jednake veličine i oblike. Primjeri takvih oblika su trokut, kvadrat i heksagon. Kod slobodne geometrijske strukture takve jedinice su varijabilne veličine i nema nekog jedinstvenog oblika.



Slika 3.4: Podijela virtualnog svijeta na kvadrate i heksagone

(*preuzeto iz* Prasetya, K. (February 2010). Efficient Methods for Improving Scalability and Playability of Massively Multiplayer Online Game, Bond University)

3.4. Zajedničko dinamičko stanje

Objekti koji se nalaze unutar UVO mogu se kategorizirati kao:

- nepromjenjivi objekti - njihovo stanje se ne mijenja,
- promjenjivi objekti - stanje im je promjenjivo,
- avatari – prikaz korisnika kojim on upravlja i
- NPC (engl. *Non-Player Characters*) – slični objekti avatarima ali su pokretani umjetnom inteligencijom.

Stanje sustava predstavlja sveukupna informacija o svim objektima unutar virtualnog svijeta u nekom trenutku. Nepromjenjivi objekti tipično nisu dio stanja i informacija o njima se preuzima prilikom instalacije ili ažuriranja aplikacije. Stanje se mijenja promjenom pozicije objekata, interakcijom korisnika s objektima i interakcijom među korisnicima.

Svaki objekt unutar UVO može promijeniti svoj položaj, orijentaciju, brzinu, izgled, ponašanje itd. Svaka promjena takvog objekta mora biti svim korisnicima vidljiva na jednak način. Promjene su vrlo česte, asinkrone i dinamične. U idealnom slučaju stanje bi trebalo biti vidljivo svima na jednak način, no to nije moguće zbog karakteristika

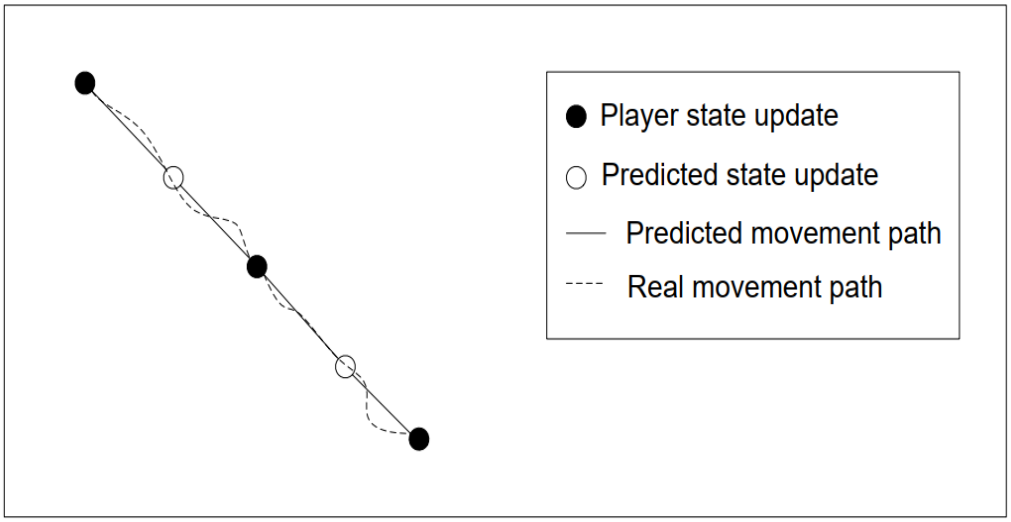
same mreže. Zbog kašnjenja i gubitka paketa dolazi do neujednačenosti samih instanci (lokalne kopije) i tako se narušava konzistentnost.

Konzistentnost se može definirati kao razina sličnosti originala i kopije. Kod potpune konzistentnosti (jaka konzistentnost) svi korisnici bi vidjeli isto stanje svijeta u svakom trenutku. Za neke vrste aplikacija ili u nekim trenutcima povremena nekonzistentnost (slaba konzistentnost) je prihvatljiva, a razlika među kopijama se nastoji minimizirati koristeći učestalu regeneraciju stanja i tehnike predviđanja.

Kod učestale regeneracije stanja svaki vlasnik entiteta u pravilnim vremenskim razmacima šalje puni opis stanja entiteta. Za dostavu se koristi nepouzdan protokol i nema očuvanog globalnog redoslijeda pristiglih poruka.

Pojedine promjene unutar UVO odvijaju se u skladu sa zakonima fizike pa se ponašanje nekog objekta može predvidjeti. Predviđanjem se može dobro prikriti kašnjenje i gubitak paketa.

Jedna od tehnika predviđanja je interpolacija. Interpolacija služi popunjavanju razmaka između osvježavanja stanja da bi animacija i kretanje u samoj aplikaciji izgledalo glađe. Pokušava se predvidjeti kojom putanjom se korisnik kretao između prethodnog i novog stanja. Bez interpolacije, kretanje avatara od drugih korisnika bi izgledalo dosta loše, uz puno trzanja. Na slikama 3.5 i 3.6 obratite pozornost na raspored crnih i bijelih točki.



Slika 3.5.: Interpolacija

(*preuzeto iz* Prasetya, K. (February 2010). Efficient Methods for Improving Scalability and Playability of Massively Multiplayer Online Game, Bond University)

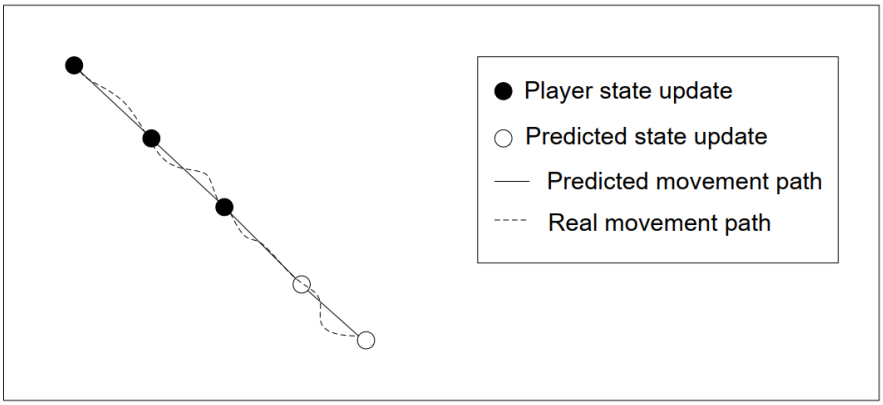
Još jedna od tehnika predviđanja, mrtva procjena (engl. *Dead Reckoning*), temelji se na pogađanju trenutne lokacije objekta. Mrtva procjena se sastoji od dvije operacije: predikcija i konvergencija.

Predikcija se koristi kada paket zakasni ili se izgubi. U tim trenutcima nedostaju pojedine informacije i potrebno je doći s rješenjem da bi iscrtavanje i dalje bilo glatko.

Najčešći pristup je računanje nove pozicije objekta koji se kreće konstantom brzinom prema sljedećoj formuli:

$$\vec{x}(t_i) = \overrightarrow{x_0} + \vec{v}(t_i - t_0)$$

gdje se nova pozicija računa na osnovi trenutno poznate lokacije, brzine i smjera kretanja. Konvergencija se vrši kada stigne stvarna pozicija objekta prema kojoj je potrebno onda pomaknuti taj objekt.



Slika 3.6: Mrtva procjena

(*preuzeto iz* Prasetya, K. (February 2010). Efficient Methods for Improving Scalability and Playability of Massively Multiplayer Online Game, Bond University)

4. Upute za pokazni primjer

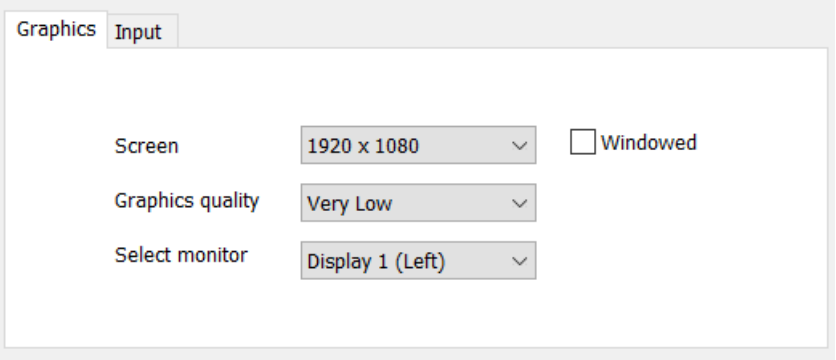
Materijali potrebni za izradu vježbe nalaze se u arhivi *VO-V3-UVO.rar* koja je dostupna na Web stranicama predmeta. U direktoriju *VO-V3-UVO\Aplikacija* nalazi se izvršna datoteka, a u direktoriju *VO-V3-UVO\Projekt* se nalazi projekt.

Aplikacija predstavlja višekorisničku umreženu igru kao jednu od vrsta umreženih virtualnih okruženja. Igra je napravljena pomoću alata **Unity** i **PUN (Photon Unity Networking)** te koristi **klijent-poslužitelj** arhitekturu.

Za izradu vježbe potrebna vam je dobra mrežna povezanost. Prilikom odabira mreže na koju ćete se spojiti potrebno je spomenuti *Ping*. *Ping* predstavlja vrijeme koje je potrebno paketu da stigne do poslužitelja i nazad do pošiljatelja. Pravu vrijednost *pinga* ćete tek moći vidjeti kada uđete u virtualni svijet. Prije ulaska u virtualni svijet zanemarite tu vrijednost jer se u tim fazama nalazite na različitim poslužiteljima u odnosu kada ste u virtualnom svijetu. Preporučeno je biti spojen na mrežu gdje je *Ping* manji od 100ms i da je ta vrijednost stabilna (± 5).

4.1. Pokretanje igre

Igru pokrećete odabirom izvršne datoteke vo3.exe koja se nalazi u mapi *VO-V3-UVO\Aplikacija*. Nakon što pokrenete igru prikazat će vam se prozor s postavkama za samu igru. Preporučene postavke možete vidjeti na sljedećoj slici.



Slika 4.1 Preporučene postavke za pokretanje igre

Sučelje igre sadrži mnogo grafičkih elemenata stoga je igru preporučljivo pokrenuti u maksimalnoj rezoluciji preko cijelog zaslona kako bi se svi elementi pravilno skalirali. Za prikaz nekih tehnika bitan je što veći *framerate*, stoga je preporučeno staviti kvalitetu slike na najmanju moguću. Ako vam *framerate* ne bude barem 50 fps onda možete postepeno i smanjivati rezoluciju. Ako imate više ekrana spojeno na jedno računalo u zadnjoj opciji možete odabrati na kojem od njih će se igra pokrenuti. To je praktična opcija ukoliko prvi dio vježbe izvodite samostalno. Nakon što ste sve podesili pritisnite tipku *Play*.

4.2. Prijava u igru

Nakon što pokrenete igru pojavljuje se ekran koji vas pita da unesete korisničko ime. Korisničko ime odaberite tako da se sastoji od prvog slova imena, punog prezimena te sufiksa *VO* (primjerice, *ihorvatVO*). Nakon prijave ponuđene su vam dvije opcije: kreiranje nove sobe i pregledavanje liste postojećih soba. Sobe služe za grupiranje igrača koji žele igrati zajedno.

Prilikom kreiranja sobe potrebno je unijeti njezino ime i maksimalni broj igrača (3-5). Nakon što kreirate sobu pojavljuje se ekran gdje vidite listu igrača koji se u njoj nalaze. Na tom ekranu također možete pokrenuti učitavanje virtualnog svijeta ako ste kreator sobe, odnosno *Master*. Nakon što je *Master* stisnuo *Start Game,* učitava se virtualni svijet te se soba zaključava i miče s liste postojećih soba. Zato, prije nego odlučite ući u virtualni svijet prvo pričekajte da svi studenti iz vaše grupe budu u njoj. Igrači mogu ući u sobu ako su prilikom pregledavanja liste postojećih soba pritisnuli gumb *Join*.

4.3. Sučelje i kontrole unutar igre

Nakon što ste ušli u virtualni svijet vidjet ćete da sučelje igre sadrži mnogo grafičkih elemenata. Sučelje igre je prikazano sljedećom slikom.

Slika 4.2: Sučelje igre

Na gornjoj traci lijevo piše broj ubijenih protivnika. Na istoj traci desno piše koliko iznosi *framerate* i *ping*. Ako vrijednosti nisu zadovoljavajuće promijenite postavke prije ponovnog ulaska u igru ili se spojite na drugu mrežu.

Dolje lijevo piše koliko Vam je života preostalo. Nakon 3 pogotka Vaš lik umire, te ponovno oživljava nakon 8 sekundi na mjestu gdje je izgubio život. Dolje po sredini piše koliko Vam je metaka ostalo. Desno od toga se nalazi mapa koja prikazuje u kojoj se zoni trenutno nalazite. Na mapi su označene strane svijeta i iznad nje se nalazi kompas radi lakšeg snalaženja unutar virtualnog prostora.

Po sredini s lijeve strane se nalaze opcije koje ćete koristiti prilikom odrađivanja 1. dijela vježbe, a s desne strane se nalaze opcije potrebne za 2. dio vježbe. Više o njima kasnije.

Unutar virtualnog svijeta se krećete pritiskom na tipke W, A, S i D. Rotacija pogleda se obavlja pomicanjem miša u odgovarajućem smjeru. Puca se lijevim klikom miša, a trčati možete pritiskom na tipku *Space*. Za prvi dio vježbe Vam ne treba i nemojte koristiti trčanje. Za drugi dio vježbe će Vam trebati trčanje jer je virtualni svijet velik, a morat ćete se kretati među različitim zonama kako bi vidjeli učinak filtriranja mrežnog prometa.

Pritiskom na tipku *Escape* otvara se izbornik sa sljedećim opcijama:

- *Resume* - nastavlja igru,
- *Options* - ulazi u opcije,
- *Leave World* - vodi vas na prije spomenuti ekran gdje imate dvije opcije, napraviti novu sobu ili pregledati listu postojećih i
- *Exit Game* - potpuno gasi aplikaciju.

Ulaskom u opcije otvara vam se ekran gdje možete mijenjati brzinu trčanja i brzinu rotacije pogleda (osjetljivost, brzinu miša). Za brzinu trčanja je preporučeno odabrati

maksimalnu vrijednost. Virtualni svijet je velik stoga brzina trčanja ima nenormalno visoku vrijednost kako bi se lakše kretali među zonama. Ako Vam ta vrijednost ne odgovara možete ju korigirati na spomenutom ekranu.

5. Opis zadatka (1. dio)

U ovom dijelu ćete morati odabrati vrstu prometa, protokol i tip podatka za dolje opisane informacije. Također, vidjet ćete učinak tehnika predviđanja i odgovoriti na pitanja vezane za njih. Na kraju će vam biti jedan problem slabe konzistentnosti koju ova igra koristi. Tu ćete morati ispuniti tablicu i odgovoriti na postavljena pitanja. U izvještaju navedite redni broj zadatka koje rješavate.

Ovaj dio možete odraditi u grupi od dva studenta ili sami. Ako radite sami potrebno je imati dva računala ili na jedno računalo spojiti dodatni ekran. Ako koristite opciju s jednim računalom i dva ekrana posebno pripazite na vrijednost *frameratea*. Ekran koji je trenutno u fokusu bi trebao imati veći *framerate* dok na drugome ta vrijednost onda drastično padne. Potrebno je da na oba ekrana *framerate* bude barem 50 fps.

Opis informacija/poruka koje se razmjenjuju u ovoj igri je sljedeći:

- **Pozicija avatara** – dva broja u rasponu [0.0, 4000.0]. Informaciju o tome na kojoj se visini avatar nalazi nije potrebno slati jer je virtualni svijet ravan,
- **Rotacija avatara** – da bi svi igrači ispravno vidjeli u kojem smjeru avatari drugih igrača gledaju potrebno je da svaki igrač šalje rotaciju svog pogleda. Ta vrijednost ima raspon [0.0, 360.0] i predstavlja rotaciju oko osi koja je okomita na virtualni svijet,
- **Ispaljeni metci** – svaki igrač šalje svima ostalima koliko je metaka ispalio kako bi oni mogli prikazati animaciju njegovog pucanja. **Ova informacija se šalje više puta u sekundi**. **Također, ne šalje se broj ispaljenih metaka od početka igre nego koliko je ispalio metaka od zadnjeg slanja poruke koja sadrži ovu informaciju**. **Očekuje se da korisnik ne može ispaliti više od 15 metaka po sekundi,**
- **Metci pokupljeni** – unutar scene postoje modeli metaka, nakon što jedan igrač pokupi metke on je zadužen da javi svima ostalima da unište taj objekt.
- **Pogodak igrača**– Ako je igrač pogodio avatara drugog korisnika potrebno je tom korisniku to i signalizirati,
- **Igrač ubijen** – Svaki igrač lokalno si umanjuje *Health* nakon što mu netko signalizira da ga je pogodio. Ako *Health* bude 0 ili manji igrač umire i zadužen je da to signalizira svima ostalima i
- **Igrač napustio igru** – nakon što se netko od igrača odspoji, poslužitelj treba to javiti svima ostalima kako bi uklonili njegov avatar.

Poruke osvježavanja stanja u ovoj igri svaki igrač šalje u prosjeku 10 puta po sekundi, dakle svakih 100 ms. Ostale vrste poruka se generiraju ovisno o akcijama igrača.

Zadatak 1. Ispunite sljedeću tablicu, tj. odredite koji bi transportni protokol koristili i vrstu prometa za pojedinu informaciju.

Tablica 5.1 Određivanje vrste prometa i protokola za pojedine informacije

Informacija	Vrsta prometa	Protokol
Pozicija (x)		
Pozicija (z)		
Rotacija		
Ispaljeni metci		
Metci pokupljeni		
Pogodak igrača		
Igrač ubijen		
Igrač napustio igru		

Zadatak 2. Dostupni tipovi podataka koje Photon podržava nalaze se na sljedećoj poveznici: https://doc.photonengine.com/en-us/realtime/current/reference/serialization-in-photon.

Na toj stranici pronađite tablicu i proučite podržane tipove podataka i njihovu veličinu (strukture podataka možete zanemariti). Ispunite tablicu 5.2, retke nadodajte po potrebi. Svakoj informaciji za koju ste odlučili da spada pod **osvježavanje stanja** pridijelite tip podatka i ispunite tablicu. **Nije potrebna dvostruka preciznost te prioritet ima tip podatka koji zauzima manje memorije a dovoljan je za prikaz pojedine informacije**.

Tablica 5.2: Određivanje tipa podatka za informaciju

Informacija	Tip podatka	Veličina

Odgovorite na sljedeća pitanja:

- Koliko iznosi veličina tijela (zaglavlje zanemarite) paketa jedne poruke osvježavanja stanja?
- Koju količinu informacija svaki igrač šalje za osvježavanje svog stanja po sekundi?

Nakon što ste ušli u igru mogli ste primijetiti da ne vidite kretanje drugih avatara. Na lijevoj strani pod *Synchronization* možete uključiti opciju *Sync Position*. Sljedeće što možete primijetiti jest da animacija hodanja i nije baš ispravna. Za odabir ispravne animacije koja se računa lokalno potrebna je i informacija o tome u kojem smjeru avatar gleda. Stoga, uključite i sinkronizaciju rotacije.

Kretanje igrača izgleda dosta loše uz puno trzanja. To je zato što informacije o pozicijama drugih avatara pristižu u prosjeku 10 puta po sekundi. To znači da će se avatar postaviti na novu poziciju 10 puta u sekundi, što nije dovoljan broj slika da se

ljudskom oku stvori privid glatkog kretanja. Isto vrijedi i za rotaciju. Visoko kvalitetne animacije iziskuju barem 60 slika po sekundi što bi onda značilo da bi svaki igrač toliko puta morao slati informaciju o svojoj poziciji i rotaciji. Ta opcija ne dolazi u obzir jer bi onda zatrpali mrežni promet, no možemo koristiti interpolaciju da bi riješili navedeni problem.

Uključite interpolaciju i sada pogledajte promjene pozicije i rotacije drugog avatara. Nakon što pristigne nova informacija o poziciji avatara drugog korisnika, avatar se ne pomiče odmah na novu poziciju kao što je bio slučaj kad je interpolacija isključena. Između prethodne i nove pozicije se generira niz novih točaka i avatar se polako pomiče prema novoj poziciji. Isto vrijedi i za rotaciju. Što je framerate veći, to implementacija interpolacije u ovoj igri bolje radi jer će se generirati više novih pozicija.

Zadatak 3. Između dvije poruke osvježavanja stanja**,** prije iscrtavanja svake slike, avatar se pomiče na jednu od interpoliranih točki te konačno na poziciju koja je bila u samoj poruci. Prilikom odgovaranja na sljedeća pitanja zapitajte se koliko se slika može iscrtati između dvije poruke osvježavanja stanja. Poruke osvježavanja stanja se šalju 10 puta po sekundi. Odgovorite na sljedeća pitanja

a) Ako korisnikovo računalo iscrtava 30 slika po sekundi, koliko **novih**, **interpoliranih** pozicija generira između 2 poruke osvježavanja stanja?
b) Ako korisnikovo računalo iscrtava 100 slika po sekundi, koliko **novih**, **interpoliranih** pozicija generira između 2 poruke osvježavanja stanja?

Ako ste ispravno odgovorili na prethodna pitanja možete vidjeti zašto je *framerate* bitan pri izradi ove vježbe. Interpolacija nam omogućuje glađu animaciju, no i dalje ostaje problem kada se paket u mreži izgubi ili zakasni. S lijeve strane imate sučelje za simuliranje mrežnih uvjeta. Uključite *Network Simulation* i povećajte gubitak paketa na 30%. Možete primijetiti da hodanje i rotacija opet izgledaju dosta loše. Uključite ekstrapolaciju i pogledajte sad.

Zadatak 4. Na prvom računalu hodajte u jednom smjeru. Na drugom računalu promatrajte kretanje avatara od prvog računala. Na drugom računalu povećajte gubitak paketa na 100%. Na prvom računalu se sada prestanite kretati.

Što računalo 2 sada vidi? Zašto?

Pronađite metke u sceni - nalaze se u zoni 1, odmah do mjesta gdje ste započeli igru. Kada igrač pokupi metke, broj metaka koje posjeduje uveća se za 20. Ako igrači pokupe isti objekt u istom trenutku on mora pripasti samo jednome od njih.

Zadatak 5. Recimo da igrač 1 i igrač 2 imaju isto kašnjenje od 100 ms. Zamislite da je implementacija u ovoj igri sljedeća (iako nije): **kada jedan igrač pokupi metke on ih prvo pridodaje sebi i onda šalje ostalima poruku da unište taj objekt.** Igrač 1 pokupi objekt, nakon 20 ms pokupi ga i igrač 2.

Koji problem se događa u ovakvom scenariju i zašto

Koje svojstvo UVO je narušeno?

U ovoj igri, kada igrač pokupi metke, on prvo zabilježi vrijeme kada je pokupio taj objekt. Zatim šalje svima ostalima zahtjev koji sadrži vremensku oznaku i naziv objekta kojeg je pokupio. Drugi igrači koji nisu pokupili objekt s tim imenom mu odmah šalju potvrdu da može pokupiti taj objekt. Ako je neki drugi igrač isto pokupio taj isti objekt onda on uspoređuje svoju vremensku oznaku s oznakom iz zahtjeva te na temelju toga mu šalje potvrdu da li smije pokupiti objekt ili ne. Igrač s manjom vremenskom oznakom ima prednost jer je on i prije pokupio metke. Za vrlo rijedak slučaj gdje obojica imaju istu vremensku oznaku metke dobiva igrač s manjim indeksom koji ova igra koristi da bi razlikovala igrače.

Zadatak 6. Na prvom računalu imajte ugašenu simulaciju mreže. Na drugom računalu upalite tu opciju te gubitak paketa stavite na 0%, a kašnjenje povećajte na maksimalnu vrijednost. Pričekajte da se *ping* poveća na postavljenu vrijednost. Kašnjenje će vam olakšati da simulirate situaciju kada dva igrača pokupe metke u približno isto vrijeme. (slučaj 2. i 3. dolje). Vidjet ćete tko je pokupio metke tako da pratite vrijednost *Ammo* na oba računala, a vremenske oznake će vam se ispisati s lijeve strane.

Generirajte 3 slučaja i ispunite tablicu 5.3:

- 1. slučaj - Igrač 1 kupi metke, drugi ne radi ništa. (ovdje već možete vidjeti da zbog kašnjenja igrača 2 kasni i njegova potvrda, pa igrač 1 ne pokupi metke odmah, nego tek kad pristigne potvrda),
- 2. slučaj - Igrač 1 pokupi metke, igrač 2 ih pokupi odmah nakon igrača 1
- 3. slučaj - Igrač 2 pokupi metke, igrač 1 ih pokupi odmah nakon igrača 2

Tablica 5.3: Tri slučaja kod kupljenja metaka

Igrač 1 (pickup time)	Igrač 2 (pickup time)	Ime objekta	Kome je pripao objekt?
	Ništa		

6. Opis zadatka (2. dio)

U ovom dijelu vježbe ćete vidjeti kako je ostvareno filtriranje mrežnog prometa strukturiranjem virtualnog svijeta. Potrebno je ispuniti navedene tablice i odgovoriti na postavljena pitanja. U izvještaju navedite redni broj zadatka koje rješavate.

Ovaj dio rješavate u grupi od 3-5 studenata. Pazite – prilikom kreiranja sobe, prvo pričekajte da svi studenti iz grupe uđu u sobu pa tek onda pokrenite igru. Također, prilikom prijave u igru napišite ime kako je opisano u poglavlju 4.2. i priložite *Screenshot* na kojem se vidi koji su sve studenti sudjelovali s Vama u izradi ovog dijela vježbe. Virtualni svijet je velik i treba vam mnogo vremena da prijeđete čak i do vama

susjedne zone. Zato koristite trčanje pritiskom na tipku *Space*. Brzinu možete povećati u opcijama kako je opisano u poglavlju 4.3.

Virtualni svijet je podijeljen na jednolike geometrijske oblike, kvadrate. Svaki kvadrat predstavlja jednu zonu te postoji 16 takvih zona. Trenutna pozicija korisnika predstavljena je kružnicom na mapi. Vaš avatar se nalazi u središtu kružnice a njezin radijus predstavlja otprilike vidno polje igrača.

Svaki igrač uvijek šalje poruke samo onoj zoni u kojoj se trenutno nalazi. Igrač je ujedno i pretplaćen na tu zonu te prima poruke od drugih korisnika ako se nalaze u istoj. Igrač se može pretplatiti i na susjedne zone. Korisnik je zainteresiran za susjednu zonu ako njegovo vidno polje (kružnica) dodiruje ili prelazi granicu susjedne zone. U slučaju kada se igrač nalazi na rubu svoje zone bit će povezan i s više susjednih zona. U tom slučaju mrežni promet je povećan ako se ostali igrači stvarno nalaze u tim, njemu susjednim zonama. Ova pravila vrijede kada uključite opciju *Network Culling* koja se nalazi s desne strane. Ispod opcije *Network Culling* vam također piše u kojoj ste zoni i koje su vam susjedne zone u svakom trenutku. Nadalje, piše broj dolaznih i odlaznih poruka po sekundi.

Zadatak 7. Pogledajte mapu unutar igre. U najgorem slučaju, na koliko najviše zona jedan igrač može biti pretplaćen?

Zadatak 8: Krećite se virtualnim svijetom i pratite broj odlaznih poruka (*Outgoing Messages/s*). Prosječan broj odlaznih poruka je otprilike uvijek isti. Zašto?

Zadatak 9: Sljedeću tablicu ispunjava samo voditelj grupe (*Master*, kreator sobe), ali sudjeluju svi studenti. Uključite *Network Culling* i krećite se virtualnim svijetom te pratite broj dolaznih poruka (*Incoming messages/s*). Zabilježite kada je promet na vašem računalu bio najmanji i kada je bio najveći (ako su svi igrači u istoj zoni to će uvijek biti slučaj - pokušajte postići ovaj scenarij tako da budete susjed s više zona). Voditelj grupe je igrač 1 te treba ispuniti tablicu za te slučajeve, tj. napisati u kojoj zoni se on nalazio i koje su mu bile susjedne. Za druge igrače treba napisati u kojoj su zoni bili.

Tablica 6.1 Pozicije igrača kada je mrežni promet bio najveći/najmanji

	Igrač 1	Igrač 2	Igrač 3	Igrač 4	Igrač 5
Najmanji promet					
Najveći promet					

Generirajte opet slučaj najvećeg mrežnog prometa. Uključite opciju *Dynamic FOV* i primijetit ćete da se kružnica na mapi smanjila. Iznad te opcije će vam ispisati koje su zone bile kritične. Kritične zone su one na koje ste bili pretplaćeni u trenutku kada je mrežni promet bio najveći (kada se smanjila kružnica). Odlaskom u neku drugu zonu koja nije među kritičnim zonama kružnica će se opet povećati a kritične zone resetirati. Smanjivanjem kružnice se želi smanjiti i šansa da postanemo susjedi s drugim zonama.

Zadatak 10: Zašto bi željeli smanjiti šansu da postanemo susjed s nekom drugom zonom?

7. Predavanje rezultata vježbe

Rezultati vježbe predaju se u dokumentu **VO-V3-Rezultati-<ImePrezime>.pdf**, koji treba sadržavati:

- Izvještaj o izvođenju vježbe (**u PDF formatu**): odgovori na postavljena pitanja, popunjene tablice te *Screenshot* tamo gdje se tražio. Navedite redni broj zadatka kojeg rješavate.

Navedena arhiva treba biti predana korištenjem sustava *Moodle* koji je dostupan preko stranica predmeta.

Napomena: rezultati se šalju isključivo preko gore navedene aplikacije. U slučaju problema, javite se email-om na adresu vo@fer.hr . Sačuvajte kopiju poslanih rezultata.

8. Napredni zadaci (ovaj dio nije obavezan)

U ovom dijelu vježbe možete nadograditi igru kroz jedan od ponuđenih zadataka. Poželjno je poznavanje barem osnova programskog jezika C# i volja da se upoznate s osnovama Unity alata. Za izradu naprednih zadataka potrebno je preuzeti i instalirati Unity verziju 2019.2.6f1 ili noviju kako bi se pokrenuo projekt. Također, potrebno je i instalirati neki od podržanih IDE alata (preporučuje se instalirati Visual Studio za kojeg možete označiti da se preuzme prilikom same instalacije Unitya). Potreban projekt nalazi se u direktoriju VO-V3-UVO\Projekt.

Za dodatne bodove potrebno je izvršiti minimalno jedan od ponuđenih zadataka, pri čemu je odabir zadataka proizvoljan. Za svaki izvršeni napredni zadatak potrebno je u izvještaj dodati opis izrade naprednog zadatka i slike (gdje se traži).

8.1. Dodavanje novog objekta u scenu

Dodajte u scenu novi objekt koji će predstavljati kutiju prve pomoći. Kada igrač pokupi taj objekt potrebno mu je uvećati *Health* na maksimalnu vrijednost. Također, potrebno je osigurati da samo jedan igrač smije pokupiti taj objekt i da ga svi ostali maknu iz scene. Unity se temelji na radu s komponentama. Na svaki objekt možemo zakačiti više komponenti, pa tako i skripte. Tako ćete i vi morati na svoj objekt dodati skriptu koja mora ostvariti gore opisane funkcionalnosti.

Upute za rad:

- Otvorite i proučite skriptu *Assets/Scripts/Pickups/AmmoPickup.cs* da vidite kako je slična funkcionalnost ostvarena za metke,
- Proučite kako radi metoda *RaiseEvent* koju pruža Photon. (https://doc.photonengine.com/en-us/pun/v2/gameplay/rpcsandraiseevent),

- Da slučajno ne bi koristili jedan od zauzetih *eventCode* identifikatora pogledajte *Assets/Scripts/PhotonRaiseEventCodes.cs,*
- Dodajte na objekt komponentu *Collider* kako bi registrirali da li je igrač pokušao pokupiti kutiju prve pomoći i
- Vašu skriptu je potrebno zakačiti na kutiju prve pomoću. U skripti, kad odlučite da igrač smije pokupiti kutiju, pozovite *PlayerManager.LocalPlayerPrefab.GetComponent<Combat>().ResetHealth()* kako bi mu povećali Health na maksimalnu vrijednost.

Model za kutiju prve pomoći možete pronaći na Internetu ili sami izraditi u nekom od alata popu: Blender, Maya, Gimp itd. Priložite *screenshot* svog objekta unutar scene i *screenshot* objekta gdje se vidi koje su sve komponente zakačene na njega.

8.2. Unaprijediti mehanizam borbe

Trenutno se u igri koristi samo 1 *collider* za model avatara. Cijelo tijelo obuhvaćeno je tim *colliderom* i ako metak pogodi igrača *Health* mu se uvijek umanjuje za 40. (https://docs.unity3d.com/Manual/CollidersOverview.html)

Vaš zadatak je da smanjite taj *collider* i dodate novi za glavu. Pogodak u glavu nanosi štetu iznosa 100, što znači da avatar od pogođenog korisnika odmah umire. Potrebno je pratiti koji je zapravo od *collidera* pogođen i na osnovi toga poslati drugom korisniku štetu koju smo mu nanijeli.

Upute za rad:

- Model avatara se nalazi pod *Assets/Resources/PlayerPrefab,*
- Smanjite postojeći *collider* te dodajte novi za glavu,
- Ako idete odmah namjestiti poziciju i veličinu *collidera* vidjet ćete da u igri ipak nije ipak ispravno postavljen. To je zato što se koriste animacije za avatare koje mijenjaju poziciju i rotaciju određenih dijelova tijela. Poziciju *collidera* i njegovu veličinu je nabolje namjestiti kada pokrenete samu igru unutar Unity Editora. Sve promjene u tom načinu rada neće biti spremljene, no možete na *collider* komponenti stisnuti desni klik -> *Copy Component*, te nakon što ugasite igru opet pritisnite desni klik -> *Paste Component Values* kako bi kopirali ispravne vrijednosti.
- Proučite metodu *FireBullet()* koja se nalazi u *Assets/Scripts/Player/Combat.cs* Unutar te metode potrebno je otkriti koji je od 2 *collidera* pogođen te na osnovi te informacije u liniji *_photonView.RPC("TakeDamage", RpcTarget.Others, damage);* nanijeti odgovarajuću štetu protivniku.

Priložite screenshot modela avatara na kojem se vide novi *collideri*.