A white rectangular object with black text

AI-generated content may be incorrect.

**ČAS ZA OBRAČUN**

Strokovno poročilo za 4. predmet poklicne mature

Mentorica: Nataša Makarovič, prof. Avtor: Klemen Skok, R 4. A

Moravče, april 2025

Povzetek

V strokovnem poročilu je predstavljen razvoj večigralske 2D videoigre, ki deluje po principu odjemalec-strežnik. Najprej so predstavljena orodja in tehnologije, ki so bile uporabljene pri razvoju. Sledi opis načrtovanja projekta, nato pa še podrobnejši opis načina komunikacije med napravami, skupnimi funkcionalnostmi, strani odjemalca in strani strežnika. Na koncu so predstavljene še težave, s katerimi sem se soočil med samim razvojem ter nova znanja in izkušnje, ki sem jih prodobil tekom razvoja.

**Ključne besede:** C++, SDL2, večigralska videoigra, strežnik, UDP

Abstract

...

**Keywords:** C++, SDL2, multiplayer videogame, server, UDP

Kazalo vsebine

[1. Uvod 5](#_Toc195532716)

[2. Orodja in knjižnice 5](#_Toc195532717)

[2.1 Programski jezik C++ 5](#_Toc195532718)

[2.2 SDL2 5](#_Toc195532719)

[2.3 spdlog 6](#_Toc195532720)

[2.4 TinyXML-2 6](#_Toc195532721)

[2.5 XML 6](#_Toc195532722)

[2.6 Program Tiled 6](#_Toc195532723)

[2.7 CMake 7](#_Toc195532724)

[3. Načrtovanje sistema 7](#_Toc195532725)

[3.1 Cilji projekta 7](#_Toc195532726)

[3.2 Potek razvoja 7](#_Toc195532727)

[4. Struktura in implementacija igre 8](#_Toc195532728)

[4.1 Osnovni koncepti 8](#_Toc195532729)

[4.1.1 Tipi objektov v igri 8](#_Toc195532730)

[4.1.2 Potek igre 10](#_Toc195532731)

[4.1.3 Mrežni sistem 10](#_Toc195532732)

[4.1.4 XML struktura zemljevida 11](#_Toc195532733)

[4.1.5 Komunikacijski protokol 12](#_Toc195532734)

[4.2 Implementacija strežnika 14](#_Toc195532735)

[4.2.1 Glavna zanka 15](#_Toc195532736)

[4.2.2 Čakalna vrsta paketov 15](#_Toc195532737)

[4.2.3 Obdelava paketa 16](#_Toc195532738)

[4.2.4 Vodenje seje 17](#_Toc195532739)

[4.3 implementacija odjemalca 18](#_Toc195532740)

[4.3.1 Glavna zanka 18](#_Toc195532741)

[4.3.2 Čakalna vrsta paketov 18](#_Toc195532742)

[4.3.3 Prikazovanje stanja 18](#_Toc195532743)

[4.3.4 Uporabniški vmesnik 18](#_Toc195532744)

[5. Ugotovitve 18](#_Toc195532745)

[6. Viri in gradiva 19](#_Toc195532746)

Kazalo slik

# Uvod

Moj maturitetni izdelek...

# Orodja in knjižnice

## Programski jezik C++

Programski jezik C++ je visokonivojski programski jezik, ki je bil razvit v 80. letih kot razširitev jezika C. Omogoča tako objektno kot tudi proceduralno programiranje. Znan je predvsem po hitrosti izvajanja, nadzoru nad programsko opremo in učinkovitosti.

Jezik podpira koncepte, kot so razredi, dedovanje, polimorfizem, šablone (ang. “templates”) in delo s pomnilnikom z uporabo kazalcev. Poleg tega ima standardno knjižnico (STL), ki ponuja širok nabor podatkovnih struktur in algoritmov. Za vsem tem pa stoji podrobna dokumentacija, ki vsebuje vse potrebne podatke za razvijalce.

## SDL2

SDL2 je odprtokodna knjižnica, ki je namenjena razvoju iger in odprtokodnih aplikacij v jeziku C oz. C++. Razvijalcem omogoča, da na enostaven način dostopajo do različnih vrst strojne opreme, kot so miška, tipkovnica, grafična kartica in podobno. Omogoča naprednejše funkcionalnosti, kot so rokovanje z dogodki, upravljanje z okni, predvajanje zvoka in uporaba širšega nabora vhodnih naprav (igralni ploščki ipd.).

Polega osnovne knjižnice obstajajo še uradne podknjižnice, ki razširjajo njeno funkcionalnost. Nekatere od teh so SDL2\_image, ki se uporablja za prikazovanje slik v različnih formatih, SDL2\_net za omrežno komunikacijo in SDL2\_ttf za uporabo poljubnih TrueType pisav v oknu.

## spdlog

Spdlog je odprtokodna knjižnica za C++, ki se uporablja za beleženje izvajanja programa (ang. logging). Omogoča izpisovanje ličnih sporočil o delovanju programa v konzolo, datoteko ali druge izhode. Knjižnica je zelo zmogljiva in enostavna za uporabo, saj lahko beleženje nastavimo že z nekaj vrsticami. Podpira več nivojev beleženja, na primer *info*, *warn*, *error*, *debug* in *trace*. To razvijalcem olajša sámo razvijanje programa in iskanje napak v izvajanju.

## TinyXML-2

TinyXML-2 je preprosta in hitra knjižnica za delo z XML datotekami. Omogoča učinkovito razčlenjevanje XML datotek, upravljanje z njimi in manipulacijo podatkov, ki jih vsebujejo.

Knjižnica uporablja objektni model dokumenta (ang. Document Object Model – DOM), kar pomeni, da XML datoteko razčleni v C++ objekte, katere lahko razvijalec spreminja, dodaja ali briše, in jih potem po potrebi shrani nazaj v datoteko.

## XML

XML (“eXtensible Markup Language”) je označevalni jezik in datotečni format za shranjevanje in prenos podatkov v obliki, ki je razumljiva tako ljudem, kot tudi računalnikom.

Podatki so organizirani v drevesni strukturi z elementi (oz. značkami) oblike “*<ime></ime>”*. Značke lahko vsebujejo atribute za dodatne informacije o posamezni znački, znotraj njih pa lahko tudi gnezdimo ostale značke. Imena značk niso vnaprej določena, pač pa jih avtor dokumenta smiselno poimenuje sam.

## Program Tiled

Tiled je brezplačen in odprtokoden program za oblikovanje nivojev za 2D igre, ki temeljijo na zgradbi iz plošč (angl. “tiles”). Omogoča vizualno oblikovanje nivojev z uporabo različnih orodij in razdelitev na plasti. Projekt lahko izvozimo v različnih datotečnih formatih, med drugimi tudi v XML, kar omogoča enostavno prebiranje podatkov o našem nivoju. Program je priljubljen med samostojnimi razvijalci videoiger, saj omogoča enostavno integracijo v igralne pogone in ostale oblike projektov.

## CMake

CMake je zmogljiv sistem za generiranje gradbenih (angl. “build”) sistemov, namenjenih prevajanju programske opreme, napisane v jezikih C in C++. Omogoča enostavno vključitev zunanjih knjižnic, upravljanje nastavitev gradnje ter podpira kompleksne projekte, hkrati pa je neodvisen od operacijskega sistema in prevajalnika. Namesto neposrednega prevajanja kode CMake ustvari ustrezne datoteke za orodja, kot sta Make ali Ninja, ki nato poskrbita za dejansko prevajanje. Pri ponovnem prevajanju CMake prevede le tiste datoteke, ki so bile spremenjene, kar lahko bistveno skrajša čas prevajanja.

# Načrtovanje sistema

## Cilji projekta

Glavni cilj tega izdelka je bil ustvariti zabavno 2D igro za več igralcev, ki temelji na arhitekturi odjemalec-strežnik (angl. “client-server”). Za komukacijo s strežnikom poskrbi protokol UDP, ki omogoča hiter, a nezanesljiv prenos podatkov po omrežju.

Sama igra deluje po principu igre “ujemi zastavico” (angl. “Capture the Flag”). 4 igralci so postavljeni v areno in razdeljeni v dve ekipi. Cilj igre je, da ena izmed ekip zastavico prinese na svojo stran igralnega polja. Igro popestrijo ovire, ki so bolj ali manj vidne. Polega tega pa se lahko igralci med seboj obmetavajo s predmeti in tako preprečijo nasprotni ekipi hiter napredek. Igra se odvija hitro, potrebno pa je veliko ekipnega dela in strategije.

## Potek razvoja

Igre sem se najprej lotil brez računalnika. Kot prve so nastale skice strežniške arhitekture, poteka povezovanja na strežnik in komunikacijskega protokola. Sledilo je testiranje orodij ter knjižnic, priprava razvojnega okolja, konfiguracije CMake in ostalih orodij, ki sem jih uporabil za razhroščevanje in prevajanje programa.

Pisanje kode sem začel na strani strežnika. Definiral sem glavno zanko in implementiral komunikacijo prek omrežja. Povezani igralci so bili razvrščeni v seje.

Ko je bila narejena podlaga na strani strežnika, sem lahko začel z razvijanjem aplikacije za odjemalce. Pri zgradbi programa sem se zgledoval po strežniku. Ko so se igralci lahko povezali na strežnik, se premikali ter lahko videli ostale igralce, sem lahko na tej podlagi zgradil celotno igro.

# Struktura in implementacija igre

## Osnovni koncepti

V tem podpoglavju so predstavljeni koncepti, ki se jih držijo tako odjemalci kot strežnik, in so ključni za delovanje celotnega sistema.

### Tipi objektov v igri

Igra vsebuje več različnih objektov, od katerih vsak po svoje vpliva na igralno izkušnjo. Razdeljeni so v dve skupini, in sicer objekte, ki se lahko premikajo in objekte, ki so zgolj del mape in ves čas stojijo na mestu.

Premični objekti

**Igralec** je verjetno najpomembnejši tovrstni objekt. Je osnovna enota v igri, ki jo upravlja igralec. Ima naslednje lastnosti:

* Položaj (x, y) in smer pogleda.
* Hitrost premikanja, ki se lahko spreminja glede na površino
* Možnost pobiranja zastave. Nošenje zastave ga upočasni in tako naredi bolj ranljivega. Zastavo je mogoče tudi izpustiti.
* Življenjske točke oziroma “stopnja telesne drže” (angl. “posture”). Zmanjša se, če je igralec zadet z izstrelkom. Ko pade na 0, se igralec težje premika odkler se vrednost čez nekaj sekund ne poveča. Če nosi zastavo, jo izpusti.

Vse igralce strežnik označi z identifikacijsko številko (ID), številko seje in ekipo. Igralec lahko pripada rdeči ali modri ekipi, kar je tudi vidno v igri.

[slika igralcev obeh ekip]

**Izstrelki** so objekti, ki jih igralci lahko mečejo v igralce nasprotne ekipe z namenom, da jih upočasnijo oziroma jim onemogočijo nošnjo zastave. Izstrelki imajo naslednje lastnosti:

* Začetni položaj in smer gibanja
* Hitrost in največjo razdaljo, ki jo lahko prepotujejo. Ko je ta presežena, izstrelek izgine.
* Učinek – če zadanejo igralce nasprotne ekipe, mu zmanjšajo življenjske točke.
* Ob trku z oviro se uničijo

[slika izstrelkov obeh ekip]

**Zastava** se ob začetku igre nahaja na sredini igralnega polje, med obema ekipama. Igralci jo lahko poberejo ali izpustijo. Zastava sama po sebi ne vsebuje nobene logike. Njen položaj se ne spreminja oziroma sledi položaju igralca, ki jo nosi.

[slika zastave]

Statični objekti

**Ovire** (angl. Barriers) so objekti, ki služijo kot fizična ovira za igralce in izstrelke. Uporabljeni so lahko v več oblikah:

* Kot stene, ki določajo igralno površino,
* samostojni objekti, ki oblikujejo notranjost igralne površine,
* ali pa kot dekoracija.

[slike sten, škatle, kavča]

**Pasti** so posebna vrsta površin, ki igralcu ne preprečujejo premikanja, pač pa na premikanje vplivajo na različne načine:

* *Lepljive pasti*, ki igralca upočasnijo.
* *Spolzke pasti*, ki igralcu odvzamejo nadzor nad premikanjem tako, da zmanjšajo trenje in tako povzročijo drsenje.

Igralcu so manj vidne, zato lahko delujejo kot element presenečenja.

[slika obeh pasti]

### Potek igre

Ko je v sejo dodan prvi igralec, se začne stanje čakanja na igralce. V tej fazi se igralci lahko prosto premikajo in raziskujejo igralno površino. Ko se poveže četrti igralec se čakanje zaključi in začne se štirisekundno odštevanje do začetka runde. Igralci se teleportirajo na začetne položaje. Premikanje je onemogočeno. Ko se runda začne, se igralci lahko premikajo, pobirajo razstavo in streljajo izstrelke. Runda se konča, ko je zastava s svojo celotno površino na eni izmed baz. Rundo dobi ekipa, ki je zastav prinesla do svoje baze. Sledi trisekundni odmor, v katerem se na zaslonu izpiše zmagovalec runde. Zatem se začne ponovno odštevanje do naslednje runde.

Zmaga ekipa, ki v petih rundah doseže največ zmag.

[slika – primerjava, ko je zastava cela v bazi vs. ko ni]

### Mrežni sistem

Namesto, da bi bili deli mape shranjeni shranjeni zgolj v navadnem seznamu, sem se odločil za pristop z mrežnim sistemom. Gre za mrežo plošč, ki si jo lahko predstavljamo kot poenostavljen koordinatni sistem, v katerem vsaka plošča zavzema eno enoto. Vsaka plošča vsebuje seznam objektov, ki se je dotikajo.

Pri dodajanju objekta v mrežo je kazalec nanj shranjen v vseh ploščah, ki se jih objekt dotika oziroma si z njim deli koordinate. Večji objekti se lahko razpostirajo čez več plošč.

[slika kode, ki preračuna položaj ovire in jo doda v sistem]

Ta pristop sem izbral predvsem zaradi potrebe po preverjanju trkov med premikajočimi se objekti in deli mape (stene, ovire ipd.). Namesto, da bi za vsak objekt preverjal trke z vsemi elementi na mapi, program izračuna približno lokacijo objekta na igralnem polju in preveri le trke s tistimi elementi, ki so v njegovi bližini. Ta način lahko znatno zmanjša število nepotrebnih preverjanj in prispeva k optimizaciji programa. Slednja je posebej pomembna na strežniku, saj ta upravlja več iger hkrati. Za omrežni sistem skrbi razred *MapData.*

### XML struktura zemljevida

Zemljevid igralne površine sem izdelal v programu Tiled. Najprej sem določil obliko tlorisa s stenami, potem pa sem prazen prostor popestril z raznimi ovirami, kot so stebri, škatle in kavči. Kasneje sem dodal še pasti.

[slika rezultata v programu Tiled]

Do podatkov o položajih objektov sem lahko enostavno dostopal prek projektne datoteke programa Tiled. Podatke sem nato pretvoril v svoj XML format in jih nato uporabil v programu.

[primer zapisa ovire v mojem XML formatu]

Podatki o celotni mapi so shranjeni v XML datoteki “map\_data.xml”. Podatki so razdeljeni v več skupin glede na funkcijo objekta:

* *<barriers>* - podatki o vseh ovirah. Vsaka ovira vsebuje podatke o položaju in dimenzijah. Na strani odjemalca je dodana še številka strukture, ki določa, kako je dana ovira prikazana na zaslonu.
* *<floor>* - program na strani odjemalca mora igralnemu polju prikazati tudi tla. Ta so prikazana s pomočjo ene majhne slike, kise ponavlja, dokler ni zapolnjena celotna površina tal. Delo sem si olajšal tako, da sem celotno igralno površino razdelil na nekaj večjih prostorov, ki jih potem program zapolni z vzorcem. Tukaj so shranjeni podatki o teh prostorih.
* *<sites>* - podatki o bazah obeh skupin. Vsaka baza vsebuje položaj, dimenzije in številko ekipe (1 ali 2)
* *<traps>* - podatki o pasteh. Vsaka past vsebuje položaj, dimenzije, številko teksture (samo za odjemalce) in vrsto pasti.

Prednost shranjevanja strukture zemljevida v zunanji datoteki je, da ob spremembah ni potrebno ponovno prevesti programa. To omogoča hitre popravke in dodajanje novih elementov zemljevida po potrebi.

[okvirna struktura datoteke map\_data.xml]

Ob zagonu program prebere to datoteko in s pomočjo knjižnice TinyXML-2 iz nje pridobi vse potrebne podatke. Najprej so prebrane vse ovire, ki jih funkcija *parseBarrierNode* napolni s podatki. Nato so dodane v mrežni sistem. Na enak način so dodane tudi pasti. Sledita bazi obeh ekip, ki sta zaradi hitrega dostopa shranjeni posebej. Na koncu so pri odjemalci pridobijo še podatke, potrebne za risanje tal, ki so prav tako shranjeni ločeno od mrežnega sistema.

[slika dela funkcije, ki iz datoteke prebere ovire]

### Komunikacijski protokol

Za komunikacijo med odjemalci in strežnikom uporabljen omrežni protokol UDP, ker omogoča hiter prenos podatkov, kar je pomembno pri akcijskih igrah, kot je ta. S tem prinaša tudi slabosti, saj paketi lahko pridejo podvojeni, v napačnem vrstnem redu, ali pa sploh ne pridejo. Med načrtovanjem projekta je bilo potrebno paziti prav na to.

Zgradba omrežnega paketa

Vsak paket, ki je poslan v omrežje, ima natanko določeno zgradbo. Pri tem sem se zgledoval po delovanju zanesljivejšega protokola TCP, ki pakete označuje z zastavicami (angl. “flags”).

|  |  |  |  |
| --- | --- | --- | --- |
| HEX vrednost | Desetiška vrednost | Ime zastavice | Pomen |
| 0x01 | 1 | **X** | Ni v uporabi |
| 0x02 | 2 | **X** | Ni v uporabi |
| 0x04 | 4 | **X** | Ni v uporabi |
| 0x08 | 8 | **DATA** | Paket vsebuje podatke o igri |
| 0x10 | 16 | **KEEPALIVE** | Vzdrževanje povezave |
| 0x20 | 32 | **SYN** | Zahteva za vzpostavitev povezave |
| 0x40 | 64 | **FIN** | Zahteva za prekinitev povezave |
| 0x80 | 128 | **ACK** | Potrditev (angl. “acknowledge”) |

Paket je sestavljen iz bajtov (zlogov, angl. “Bytes”). Splošna zgradba paketa je sledeča:

1. Zastavice (1 bajt).
2. Številka seje, ki ji pripada igralec (1 bajt).
3. ID številka igralca (2 bajta).
4. Zaporedna številka paketa (4 bajti). Uporablja se za zaznavanje podvojenih paketov.
5. Tip paketa (1 bajt). Vsebuje informacijo o vsebini paketa.
6. Podatkovni del, ki ima različno velikost.

[slika zgradbe paketa]

Vrste paketov

Vrsta paketa prejemniku paketa pove, kako mora prebrati paket, da lahko iz njega izloči prave informacije. Vrste paketov so sledeče:

* PLAYER\_UPDATES – edina vrsta paketa, ki ga med izvajanjem seje pošilja odjemalec. Podaktovni del vsebuje stanja tipk, ki jih strežnik potrebuje za vodenje igre.
* PLAYERS\_IN\_RANGE – paket, ki ga strežnik pošlje odjemalcu. Podatkovni del vsebuje niz podatkov vseh igralcev, ki so trenutno v igri. Prvi v nizu so vedno podatki igralca, ki mu je paket namenjen.
* PROJECTILES\_IN\_RANGE – paket, ki ga strežnik pošlje odjemalcu. Vsebuje niz podatkov o vseh aktivnih izstrelkih.
* GAME\_STATE – paket, ki ga strežnik pošlje odjemalcu. Vsebuje podatek o trenutni fazi igre, trenutnem rezulatu in času, ki je minil od začetka trenutne faze.
* FLAG\_STATE – paket, ki ga strežnik pošlje odjemalcu. Vsebuje položaj zastave in ID številko igralca, ki jo nosi (0, če trenutno ni nošena).

[slika enum-a PacketType]

Povezovanje in seja

Povezavo med odjemalcem in strežnikom mora vedno začeti odjemalec z zahtevo za vzpostavitev povezave. To stori tako, da pošlje 1 bajt (zastavice) s prižgano zastavico SYN na naslov strežnika. Naslov strežnika je poznan, oziroma mu ga ročno določi uporabnik. Strežnik si igralca zapomni in se, če je mogoče dodati novega igralca, odzove z zastavicama SYN in ACK ter pripadajočo številko seje in številko igralca. V nasprotnem primeru odgovori s praznim sporočilom (1 bajt, vse zastavice imajo vrednost 0), ki pomeni zavrnitev povezave. Vsi paketi, poslani med sejo, imajo zgoraj opisano strukturo.

Ko hoče odjemalec prekiniti povezavo, strežniku pošlje prižgano zastavico FIN skupaj s svojo ID številko in števiko seje. Strežnik odgovori z zastavicama FIN in ACK ter igralca odstrani iz igre.

Izmenjava podatkov

Strežnik nadzoruje potek igre in določa veljavne položaje vseh objektov in igralcev na polju. Odjemalci strežniku pošiljajo svoja dejanja (stanja tipk za premikanje, interakcijo za zastavo in usmerjenost), strežnik pa te podatke preveri, izvede logiko igre (premikanje glede na stanja tipk, trke, interakcije z zastavo, sproža izstrelke ipd.) in nato vsem igralcem pošlje posodobljeno stanje.

Nezanesljivost protokola UDP je med izvajanjem seje rešena s tako, da strežnik podatke o stanju igre pošilja periodično, in sicer približno vsakih 32 milisekund oziroma s frekvenco 30 Hz. To je dovolj za gladko izkušnjo pri igranju, hkrati pa z relativno nizko frekvenco pošiljanja razbremeni strežnik.

## Implementacija strežnika

Strežniška aplikacija deluje na treh nitih (angl. “threads”). Na glavni niti se izvaja osrednja funkcionalnost strežnika. Na stranskih nitih pa poteka komunikacija – pošiljanje paketov na eni niti in sprejemanje na drugi niti.

Osrednji del je sestavljen iz več razredov s statičnimi metodami in lastnostmi, kar omogoča logično razdelitev funkcionalnosti programa. Glavni razred je razred *Server*, ki med drugimi vsebuje tri glavne metode za zagon strežnika. Te metode so *Server::Setup*, ki poskrbi za zagon komunikacijskih niti ter nalaganje zemljevida iz XML datoteke, *Server::Run*, kjer se izvaja glavna zanka programa in *Server::Cleanup*, ki se izvede ob zaključku programa in poskrbi za pravilno zaustavitev. Razred vsebuje še metode, ki se izvajajo v glavni zanki in metode, ki so namenjene upravljanju z odjemalci in sejami.

Lastnosti, ki jih vsebuje *Server*, so seznam aktivnih sej ter seznama prostih ID številk sej in odjemalcev, ki niso več v uporabi in bodo imeli prioriteto, ko bo potrebno dodeliti ID številko novemu odjemalcu ali seji. Razlog za to je omejenost s prostorom v omrežnem paketu. Številka seje zavzema 1 bajt (vrednosti od 0 do 255), številka igralca pa 2 bajta (vrednosti od 0 do 65535), kar pomeni, da bi številk prej kot slej zmanjkalo in prišlo bi do napake. Zato se številke sej in igralcev ponovno uporabljajo.

[slika definicije razreda Server]

### Glavna zanka

Glavna zanka strežika izgleda dokaj preprosto. Najprej funkcija *Server::processNewPackets* poskrbi za procesiranje novih paketov, ki so bili prejeti od zadnje iteracije. Nato se kliče metoda *Server::manageGameSessions*. Ta metoda posodobi stanja vseh sej, ki so trenutno aktivne. Hkrati tudi zaključi prazne ali končane seje. Metoda *Server::sendPendingPackets* pridobi vse pakete, ki so jih v prejšnjem koraku ustvarile seje in jih doda v kolono za pošiljanje. Metoda *Server::checkClientInactivity* odstrani tiste odjemalce, ki že dolgo niso poslale nobenega paketa (zaradi izgube povezave ali drugih dejavnikov). Tako se sprosti prostor za nove igralce.

Na koncu še preverimo, ali je uporabnik zahteval zaustavitev strežnika. Sledi še relativno kratek zamik, s katerim omejimo hitrost delovanja strežnika, hkrati pa poskrbimo, da vseeno ostane dovolj odziven.

[slika metode Server::Run]

### Čakalna vrsta paketov

Delovanje strežnika torej poteka na treh nitih (angl. “threads”): glavna nit je najhitrejša. Na njej se izvaja glavna zanka. Na prvi stranski niti se izvaja zanka, ki sprejema pakete, ki jih pošljejo odjemalci. Na drugi stranski niti pa se izvaja zanka, ki pošilja pakete odjemalcem.

Vse tri niti med seboj povezujeta dva vsebnika (angl. “containers”) iz C++ standardne knjižnice *std::queue*, ki vsebujeta pakete, ki morajo biti poslani oziroma predelani. Vsebujeta objekte razreda *UDPmessage*, ki vsebuje vse podatke o paketu (kanal, IP naslov, dolžino in vsebino paketa). Pri uporabi teh vsebnikov je potrebno paziti na pravilno dostopanje do njih. Lahko se namreč zgodi, da do istega vsebnika hkrati dostopata dve niti, kar lahko privede do nedefiniranega stanja in napak. V ta namen se uporablja zaklepanje z uporabo objekta *std::mutex* – “ključavnica”. Ko poskušamo zakleniti želeni vsebnik, bo std::mutex počakal dokler ne prost za uporabo. Nato lahko dostopamo do vsebnika.

[slika - primer zaklepanja z std::mutex]

Pošiljanje paketov

Pakete, ki morajo biti poslani odjemalcem glavna nit doda v čakalno vrsto – vsebnik *sendQueue*.

Za pošiljanje teh paketov skrbi razred *SocketSpeaker*. Ob zagonu se ustvari nova nit, v kateri se izvaja metoda *SocketSpeaker::Speak*, njen kazalec pa je shranjen v lastnosti *std::unique\_ptr<std::thread> worker*. Ko se program zaustavlja, je lastnost *\_shutdown* nastavljena na *true*, kar omogoči da se nit samodejno zaključi. Metoda *Speak* neprestano čaka na nove pakete, ki pridejo v čakalno vrsto, in jih nato pošlje preko omrežja.

[slika razreda SocketSpeaker]

Sprejemanje paketov

Paketi, ki so bili prejeti, so shranjeni v vsebniku *RecievedQueue* in čakajo na procesiranje.

Za sprejemanje paketov skrbi razred *SocketListener*. Deluje na podoben princip kot razred za pošiljanje – ob zagonu se ustvari nova nit, v kateri se izvaja metoda *SocketListener::Listen* in katere naslov je shranjen v lastnosti *std::unique\_ptr<std::thread> worker*. Nit se prav tako zaključi, ko lastnost *\_shutdown* nastavimo na *true*. Metoda *Listen* vse prejete pakete doda v čakalno vrsto, kjer čakajo na obdelavo v glavni niti.

[slika razreda SocketListener]

### Obdelava paketa

Obdelava paketa poteka v metodi *Server::processNewPackets*, ki se kliče v glavni zanki strežnika. Metoda preveri, ali je bil v čakalno vrsto *recievedQueue* dodan nov paket, in ga nato vzame iz vrste.

Najprej ugotovi, ali je pošiljatelj paketa že poznan. Pri tem strežnik izkorišča funkcionalnost knjižnice SDL2\_Net, ki omogoča shranjevanje IP naslova kot številko t.i. “kanala” (angl. “channel”). Ta številka je pri povezavi določena glede na ID številko igralca. Če IP naslov pošiljatelja še ni poznan, bo vrednost njegovega kanala enaka -1.

Če je torej kanal, prek katerega je prispel paket, enak -1 in če vsebina paketa ustreza zahtevani obliki za vzpostavitev povezave, strežnik ustvari novega igralca ter pošlje potrditev povezave. Če dodajanje novega igralca ni mogoče (npr. zaradi preseženega števila igralcev), strežnik zahtevo zavrne. Če vsebina paketa ni veljavna, se paket ignorira.

V primeru, da je pošiljatelj že znan, program s pomočjo zanke preveri stanje zastavic in ustrezno obdela paket:

* FIN – igralca odstrani iz igre in pošlje potrditev.
* KEEPALIVE – posodobi igralčev časovnik na trenutni čas, kar strežniku pove, da je povezava še vedno aktivna, tudi če se igralec ne premika.
* DATA – paket posreduje ustrezni seji (angl. “session”) za nadaljnjo obdelavo.

### Vodenje seje

Vse seje so objekti razreda *GameSession*, ki vsebuje vse potrebne lastnosti in metode za vodenje seje. To vključuje vsebnike za vse premične objekte na igralnem polju, ki so odvisni od seje – igralce(objekti razreda *Player*), izstrelke (objekti razreda *Projectile*) in zastavo (ne vključuje pa elementov zemljevida – ti so skupni vsem sejam). Ločeno so shranjeni še podatki o odjemalcih – številka in čas zadnjega prejetega paketa, številka zadnjega poslanega paketa ter IP naslov. Prisotne so še vrednosti za vodenje stanja igre – trenutna faza in števci trajanja.

Vsem sejam je skupna čakalna vrsta paketov (javna statična lastnost razreda *GameSession*), v katero seje dodajo pakete, ki morajo biti poslani odjemalcem. Ko so posodobljene vse seje, se v glavni zanki strežnika kliče zgoraj omenjena metoda *Server::sendPendingPackets*, ki pakete prestavi v vrsto *sendQueue*.

Seje omogočajo še dodajanje in odstranjevanje igralcev, predelovanje in pošiljanje različnih paketov vodenje seje.

Glavna metoda za vodenje seje je *manageSession*, ki predstavlja glavno zanko seje. Metoda preveri, koliko časa je minilo od zadnje posodobitve. Ko je čas za posodobitev, se kličejo naslednje metode:

1. *updateEverything*: metoda posodobi vse premične objekte. Kot argument sprejme čas, ki je minil od zadnje posodobitve (angl. “delta time”). Položaji so posodobljeni s formulo: . Posodobljen je tudi položaj zastavice, če jo kdo nosi. Za razliko od ostalih objektov so trki igralcev z elementi zemljevida preverjeni že pri posodobitvi igralca.
2. *checkCollisions*: metoda preveri še vse ostale trke. Najprej so preverjeni trki izstrelkov z igralci, sledijo še trki izstrelkov s stenami. Na koncu se preveri še, če se zastava nahajo na kateri izmed baz, saj bi to pomenilo zaključek runde.
3. *checkGameState*: po potrebi spremeni fazo igre.
4. *broadcastUpdates*: odjemalcem pošlje nove položaje igre, v krajših intervalih tudi pošlje trenutno stanje igre (paket tipa GAME\_STATE)

## implementacija odjemalca

Aplikacija odjemalca se zgleduje po strežniški arhitekturi. Na glavni niti poteka osrednji del programa, za razliko od strežnika pa pri odjemalcu komunikacija poteka le na eni stranski niti. To poenostavi komunikacijo s strežnikom, saj lahko ta pakete odjemalcu pošilja na ista UDP vrata (angl. “port”), s katerih jih prejema.

Program je logično razdeljen na več razredov s statičnimi lastnostmi in metodami, od katerih ima vsak svojo vlogo. Osrednji razred je razred *Game*, ki med drugimi vsebuje glavno zanko in metode za zagon in zaustavitev pograma.

* *Setup*: metoda je klicana ob zagonu programa. Zažene komunikacijsko nit, odpre okno, naloži teksture za uporabniški vmesnik ter pripravi zemljevid igre.
* *setServerIP*: uporabnik aplikacije ima možnost, da ročno vnese naslov strežnika tako, ta IP naslov strežnika (ali URL) ter vrata (angl. “port”), ki jih strežnik uporablja za sprejemanje paketov, zapiše kot argumenta pri zagonu programa. V kolikor ni ročnega vnosa bo nastavljen privzet naslov.
* *Run*: v tej metodi teče glavna zanka programa.
* *Cleanup*: ta metoda je klicana ob izteku izteku programma in poskrbi za pravilno zaustavitev.

Razred vsebuje še druge lastnosti in metode, ki služijo delu s paketi ter upravljanju s povezavo in podatki o igri.

Zaradi preglednosti obstajajo še drugi razredi s statično vsebino, ki skrbijo za različna opravila:

* *SocketHandler*: skrbi za pošiljanje in sprejemanje paketov
* *PacketHandler*: obdela prejete pakete in sestavlja pakete, ki bodo poslani
* *EventHandler*: skrbi za zaznavanje uporabnikovih vnosov
* *RenderWindow*: skrbi za prikaz igre in uporabniškega vmesnika na zaslon

### Glavna zanka

Glavna zanka odjemalca najprej pripravi vse potrebne spremenljivke (klic metode *Initialize*). Med izvajanjem glavne zanke si sledi več korakov.

Najprej program s klicem metode *processNewPackets* pridobi in predela prejete pakete. Nato preveri uporabnikove vnose in ob morebitnih spremembah ustvari paket posodobitev za strežnik in ga doda v čakalno vrsto, ki je statična lastnost razreda *PacketHandler*. Zatem klic metode *manageConnection* poskrbi za redno pošiljanje ustreznih paketov (zahteve za povezavo, pakete za ohranitev povezave ali zahteve za prekinitev povezave. To je odvisno od stanja povezave). Metoda *PacketHandler* pakete iz lokalne kolone prestavi v čakalno vrsto za pošiljanje. Iteracija se zaključi z lokalnim posodabljanjem igre (*Update*) in prikazom stanja (*Render*).

Glavna zanka se izvaja s frekvenco 60 hercov (med dvema iteracijama je približno 16,6 milisekund premora). To je dovolj za gladko prikazovanje stanja.

[slika metode Game::Run]

### Čakalna vrsta paketov

Glavni in komunikacijsko nit povezujeta dva vsebnika *std::queue* – *recievedQueue* za sprejete pakete, ki morajo biti predelani in *sendQueue* za pakete, ki morajo biti poslani strežniku. Tudi tukaj je potrebno paziti na dostopanje do vsebnikov, zato ima vsak od njiju svojo ključavnico – *std::mutex*, ki poskrbi za posamično uporabo vsebnikov.

Za komunikacijo na stranski niti skrbi razred *SocketHandler*. Na niti se ob zagonu začne izvajati metoda *Work*, ki skrbi tako za pošiljanje kot za sprejemanje paketov. Kazalec na metodo je shranjen v lastnosti *std::unique\_ptr<std::thread> worker*. Ob zaustavitvi programa se lastnost *\_shutdown* nastavi na *true* in nit se zaključi samodejno.

[slika razreda SocketHandler]

### Prikazovanje stanja

Igralna površina je velika, zato je potrebno okolico prikazati glede na igralca. Lokalni igralec je vedno narisan na sredini zaslona, premika pa se vse ostalo okrog njega.

Vredno je omeniti, da stanje v prejetih paketih ni takoj prikazano. Ker aplikacija odjemalca deluje hitreje kot strežnik pošilja podatke, dejanske podatke s strežnike vsebuje samo vsak drugi frame. Te “luknje” v prikazu so zapolnjene s napovedovanjem položaja (angl. “prediction”) glede na trenuten položaj in stanje tipk. Prejetega paketa ne uporabimo takoj, pač pa ga dodamo v buffer. Ko posodobimo igralca, najprej pogledamo, če je na voljo tak paket. Med položajem v prejšnjem paketu in položajem v naslednjem paketu izvajamo linearno glajenje (angl. “linear interpolation”), kar pomeni, da med položajema najdemo sredinsko točko, ki jo prikažemo v vmesnem frame-u. Tako bo premikanje ostalih igralcev ostalo gladko, tudi če strežnik ne pošilja podatkov dovolj hitro.

Za lokalnega igralca se uporablja drugačen pristop. Če bi namreč za lokalnega igralca prikazovali le stanje, ki ga pošlje strežnik, bi se sistem zdel počasen in neodziven. Zato aplikacija odjemalca takoj prikaže premika glede na svoje podatke, s podatki s strežnika pa vrednosti le popravi. Za boljšo izkušnjo je tudi tu uporabljena interplacija, podatki s strežnika pa so uporabljeni takoj.

[slika logike za interpolacijo (lahko je psevdo koda)]

### Uporabniški vmesnik

# Ugotovitve

# Viri in gradiva