TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduste instituut

ITV40LT
Sander Saarsen 134683 IAPB

3D VÕRGUMÄNGU ARENDUS UNITY MÄNGUMOOTORIL

Bakalaureusetöö

Juhendaja: Jaagup Irve
Magister
Tarkvarainsener

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Sander Saarsen

23.05.2016

Tallinn 2016

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua üle võrgu mängitav mitmikmäng ning uurida ja analüüsida võrgumängudes esinevaid probleeme ning pakkuda neile lahendusi.

Töös käsitletakse mitmeid võrgumängude puhul esinevaid probleeme, näiteks nagu latentsus, paketikadu ja petmine ning tutvustatakse ka lahendusi nendele probleemidele. Lisaks antakse töös põhjalik ülevaade erinevatest kasutusel olevatest võrguarhitektuuridest.

Bakalaureusetöö praktilise osana valmis 3D võitlusmäng, kus mängijad saavad üle võrgu üksteise vastu võidelda. Mäng on tehtud Unity mängumootoriga, millest käesolevas töös samuti detailsem ülevaade antakse.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 6 peatükki, 14 joonist.

Abstract

3D multiplayer game development with Unity game engine

The goal of this thesis is to create an online multiplayer game and to analyse the different problems that occur in online games and also to provide solutions to these problems.

Problems like latency, packet loss and cheating in online games will be discussed in this document and solutions to these problems will be introduced. Also in addition different online game network architectures will be discussed.

As a result of this work, a 3D fighting game was made, where players can play with each other online. The game was created using Unity game engine, which will be also discussed about in this thesis.

The thesis is in Estonian and contains 35 pages of text, 6 chapters, 14 figures.

Lühendite ja mõistete sõnastik

IDE	Integrated development environment, integreeritud programmeerimiskeskkond

Sisukord

1 Sissejuhatus 9
2 Võrgumängude ülesehitus 10
2.1 Võrgumängude üldine tööpõhimõte 10
2.1.1 Klient 10
2.1.2 Server 10
2.2 Kasutatavad võrguarhitektuurid 11
2.2.1 Klient-server 11
2.2.2 Partnervõrk 11
2.2.3 Jagatud võrk 12
2.3 Võrguserverite tüübid 13
2.3.1 Autoratiivsed 13
2.3.2 Mitteautoratiivsed 14
2.4 Probleemid võrgumängudes 14
2.4.1 Petmine 14
2.4.2 Latentsus 14
2.4.3 Paketikadu 15
2.5 Võrguprobleemidega toimetulek 16
2.5.1 Interpoleerimine 16
2.5.2 Ennustamine 16
2.5.3 Kompenseerimine 17
2.5.4 Andmekontroll 18
2.5.5 Andmemahtude optimeerimine 18
3 Kasutatud tehnoloogiad 20
3.1 Unity 3D 20
3.1.1 Võimalused 20
3.1.2 Multiplatvormilisus 21
3.1.3 Litsenseerimine 21
3.2 MonoDevelop 21
3.3 C# 22
4 Rakenduse ülesehitus 23
4.1 Peamised kasutatavad Unity komponendid 23

4.1.1 Komponendid 23
4.1.2 Scene 23
4.1.3 Prefab 23
4.1.4 GameObject 24
4.1.5 Collider 24
4.1.6 Rigidbody 25
4.1.7 NetworkManager 26
4.1.8 NetworkTransform 26
4.1.9 NetworkIdentity 26
4.2 Peamised autori poolt loodud komponendid 26
4.2.1 Skriptid 26
4.2.2 PlayerController 26
4.2.3 PlayerMovementSync 27
4.2.4 PlayerRotationSync 27
4.2.5 HealthManager 27
4.2.6 BulletController 27
4.2.7 MenuController 27
4.2.8 CharacterSelectionController 27
4.3 Stseenid 28
4.3.1 Menu 28
4.3.2 Options 28
4.3.3 Main 29
4.4 Rakenduse kirjeldus 30
4.4.1 Mängu kirjeldus 30
4.4.2 Mängu töökäik 31
5 Töö analüüs 32
5.1 Vastavus ülesandele 32
6 Kokkuvõte 33
Viited 34

Jooniste loetelu

Joonis 1 Klient-server arhitektuur *11*
Sander Saarsen, joonis klient-server arhitektuurist

Joonis 2 Partnervõrgu arhitektuur *12*
Sander Saarsen, joonis partnervõrgu arhitektuurist

Joonis 3 Jagatud võrgu arhitektuur *13*
Sander Saarsen, joonis jagatud võrgu arhitektuurist

Joonis 4 Pilt mängust Counter-Strike, kus on sinise ja punasega näha tegelase mänguseisude muutus laskmishetkest kliendi arvutis info jõudmiseni serverisse. *18*
Valve, https://developer.valvesoftware.com/w/images/c/ca/Lag_compensation.jpg

Joonis 5 Unity kasutatavus võrreldes muude turul saadavate mängude arendusplatvormidega *20*
Unity, https://unity3d.com/public-relations

Joonis 6 Tühi stseen *23*
Unity, http://docs.unity3d.com/uploads/Main/NewEmptyScene.png

Joonis 7 Näide mänguobjektist Unitys..... *24*
Unity, http://docs.unity3d.com/uploads/Main/GameObjectCubeExample.png

Joonis 8 Mängus kasutatava tegelase Collider kujutatuna roheliste äärtega kuubina . *25*
Sander Saarsen, pilt Unity arenduskeskkonnas

Joonis 9 Näide Rigidbody komponendist Unitys *25*
Sander Saarsen, pilt Unity arenduskeskkonnas

Joonis 10 Mängu avamenüü..... *28*
Sander Saarsen, pilt arendatud mängust

Joonis 11 Seadete stseen *29*
Sander Saarsen, pilt arendatud mängus

Joonis 12 Tegelase valik mängu sisenemisel *29*
Sander Saarsen, pilt arendatud mängus

Joonis 13 Mängu maailm, kasutaja poolt valitud tegelasega..... *30*
Sander Saarsen, pilt arendatud mängus

Joonis 14 Escape klahviga mängu ajal avatav menüü *30*
Sander Saarsen, pilt arendatud mängus

1 Sissejuhatus

Bakalaureusetöö eesmärgiks on luua üle võrgu mängitav mitmikmäng ning uurida ja anlüüsida põhilisi võrgumängude tegemisel esinevaid probleeme ning võimalusi nende probleemidega toimetulekuks.

Töö esimeses osas räägitakse võrgumängude tööpõhimõttest ja erinevatest arhitektuuridest ning tutvustatakse peamisi võrgumängudes esinevaid probleeme nagu latentsus, paketikadu ja petmine. Tutvustatakse ka erinevaid võimalusi, mida kasutatakse toimetulemiseks eelnimetatud probleemidega, sh näiteks interpoleerimine, ennustamine ja kompenseerimine.

Bakalaureusetöö teises pooles tutvustatakse lähemalt autori poolt loodud mängu ja Unity mängumootori poolt pakutavaid vahendeid ja võimalusi võrgumängude tegemiseks.

Töö on koostatud aastal 2016 Tallinna Tehnikaülikooli informaatika bakalaureuseõppe eriala lõputööna.

2 Võrgumängude ülesehitus

2.1 Võrgumängude üldine tööpõhimõte

Võrgumängud on mängud, mis kasutavad interneti või kohtvõrku, et lisada mängule täiendavat funktsionaalsust. Näiteks nagu ülevõrguline edetabel või võimalus mitmel mängijal sama mänguruumiga liituda. Võrgumängud sarnanevad paljuski tavalistele mängudele selles mõttes, et enamik mänguobjektidest ja animatsioonidest asuvad mängija enda arvutis ning kuvatakse kasutajale samal viisil nagu tavalises mängus, lisandub ainult see, et üle võrgu vahetatakse teiste kasutajatega infot, lisades seeläbi mängule lisaväärtust. Näiteks võivad mängijad vahetada üle võrgu omavahel andmeid enda liikumisest ja tegevustest ning see võimaldab igal kasutajal lokaalselt teiste mängijate tegevused enda mänguruumis kuvada.

Viise, kuidas andmeid saata ja mis andmeid saata, on erinevaid. Seejuures on palju detaile, mida peab täiendavalt arvesse võtma. Üldiselt saadetakse andmeid minimaalses mahus nagu näiteks tegelaste ja objektide koordinaadid, elude hulk või mängija punktiskoor. Animatsioonid ja reaalne liikumine tehakse juba kasutajate enda arvutis.

2.1.1 Klient

Võrgumängudes võib kliente käsitleda kui mängijaid. Kõik kliendid käivitavad lokaalselt samasuguse mänguruumi ning üksteise vahel vahetatakse infot kasutajate tegevustest või liikumisest. Infovahetus võib toimuda kas otse kliendilt kliendile või läbi serveri.

2.1.2 Server

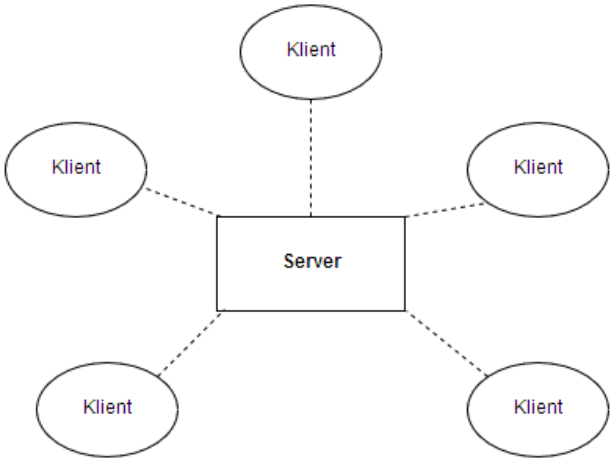
Võrgumängus kasutatakse servereid erinevatel eesmärkidel, kuid peamiselt infovahetuseks klientide vahel. On võimalik üles ehitada võrgumängud ka ilma servereid kasutamata, aga üldiselt on serverid laialdaselt kasutusel. Servereid kasutatakse ka klientide poolt saadetavate andmete valiidsuse kontrollimisel, et raskendada petmist. Üldiselt on serveril ülevaade kõikidest klientidest ja nende mänguseisudest. Serveril on tavaliselt ka rohkem võimu ja õigusi kui klientidel - nt kliendid saadavad enda tegevuste

kohta andmeid, aga server otsustab, mida nende andmetega teha või kas saadetud andmed on sobivad.

2.2 Kasutatavad võrguarhitektuurid

2.2.1 Klient-server

Klient-server arhitektuur on üks levinumaid võrgulahendusi, mida mitmikmängudes kasutatakse. Selle võrguarhitektuuri tüübi puhul on keskne server ning kõik mängijad on kliendid, kogu info liikumine toimub läbi sellesama keskse serveri (vt joonis 1). Serveris hoitakse kõikide klientide mänguseise ning info iga tegevuse kohta, mida mingi klient teeb, saadetakse serverile, et seal see uuendada ja saata edasi teistele klientidele [1].



Joonis 1 Klient-server arhitektuur

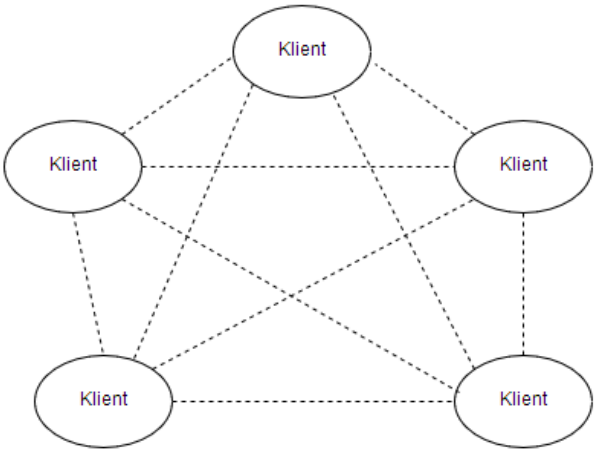
Sellise arhitektuuri probleemiks on halb skaleeruvus, kui mängijate arv kiiresti suureneb. Võib tekkida olukord, kus server ei suuda sisse- ja väljaminevaid andmeid piisavalt kiiresti edastada ning sellega tekivad probleemid klientide mängukeskkondades, kus objektide asukohti ning tegevusi uuendatakse liiga hilja või aeglaselt ning seeläbi saab kannatada kasutajakogemus.

2.2.2 Partnervõrk

Partnervõrk ehk P2P-võrgustik on võrgu arhitektuur, kus ei ole kasutusel serverit, vaid kliendid moodustavad omavahel võrgu ning jagavad infot üksteise vahel otse (vt joonis 2). Iga klient võrgus omab lokaalselt enda mängu seisu ning vastutab ise selle edastamise

eest teistele klientidele. Iga muutus mänguseisus edastatakse kõikidele teistele klientidele võrgus.

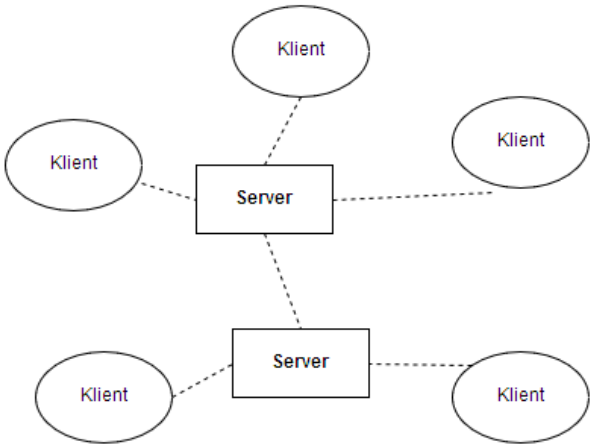
Partnervõrgu eelis klient-serveri ees on arhitektuuri robustsus ning kuna mänguseisude uuendamise eest ei vastuta üks keskne server, on see ka oluliselt töökindlam. Sellise arhitektuuri puhul on aga probleemiks andmete valiidsuse kontrollimine - kuna info käib läbi mitme kliendi, siis on raskem kontrollida andmete valiidsust ning see teeb süsteemi petmise lihtsamaks. [2] Lisaks, kuna iga klient ei saada infot ühte kesksesse serverisse, vaid kõikidele klientidele võrgus, suureneb oluliselt kliendi poolt saadetava info hulk, mis võib olla probleemiks nendele kasutajatele, kellel on limiteeritud või aeglane võrguühendus. See tähendab ka seda, et kasutajate mängukogemus sõltub nende enda võrguühendusest ja arvutist.



Joonis 2 Partnervõrgu arhitektuur

2.2.3 Jagatud võrk

Jagatud võrk tähendab mitme serveri kasutamist võrgus, mängu info liigub nende serverite vahel vastavalt vajadusele (vt joonis 3). Info liikumist serverite vahel võib lahendada mitut moodi. Näiteks võib igale serverile määrata teatud kliendid - sellega vähendatakse serverite koormust. Teiseks võimaluseks on jagada serverite vahel ära mitte kliendid, vaid klientide tegevused ja mänguseisud. Sellisel meetodil on võimalik samuti serverite koormust vastavalt vajadusele reguleerida. Praktikas on kasutusel veel kolmas meetod, milleks on peegeldamine. Peegeldamise puhul on kasutuses mitu serverit, mis hoiavad kõikide klientide mänguseise. Iga klient vahetab andmeid endale kõige lähemal asuva serveriga. [3]



Joonis 3 Jagatud võrgu arhitektuur

Jagatud võrkude arhitektuuri eelisteks on parem töökindlus ja vajaduse korral skaleeruvus. Jagatud võrk võimaldab ka näiteks serverite paigutamist erinevatesse regioonidesse, mis võimaldab kasutajatel kasutada endale lähimat serverit ning seeläbi suurendada serveriga andmete vahetamise kiirust. Sellist arhitektuuri kasutab näiteks Valve, kellel on servereid erinevates riikides ja kasutaja vahendab infot lähima serveriga ning server saadab selle info mööda kiirteid edasi vajaminevasse serveriparki.

2.3 Võrguserverite tüübid

2.3.1 Autoratiivsed

Üheks levinumaks serveri tüübiks võrgumängudes on autoratiivne server, mille korral on kogu kontroll mängu üle serveril. Kliendid saadavad serverile info nupu vajutamisest või soovist teha mingit tegevust ning server seejärel otsustab, kas see on lubatud või mitte. Klientidel ei ole õigust ise mängumaailmas muutusi läbi viia ja kõik tegevused käivad läbi serveri. [4] Näiteks selle asemel, et klient teavitab serverit vastase tabamisest, annab ta serverile info, et tegi lasu. Server kontrollib seejärel ise, kas lask tabab vastast või mitte. Peamine eelis autoratiivsetel serveritel on see, et klientidel on mängu oluliselt raskem petta. Kuna kõikide tegevuste üle on serveril kontroll olemas, on lihtne kontrollida, kas mõne kliendi tegevus on korrapärane või mitte. Autoratiivsete serverite miinuseks on see, et klientidel on vaja oodata kinnitust oma tegudele, mis tähendab seda, et kasutaja ei saa visuaalset tagasisidet mängult enne kui server on loa andnud. Seega võib mäng muutuda kasutaja jaoks mittemängitavaks, kui vastused serverilt piisavalt kiiresti kasutajani tagasi ei jõua. Mängudes nimetatakse sellist nähtust ka läägiks (ingl k. *lag*).

2.3.2 Mitteautoratiivsed

Kasutatakse ka mitteautoratiivseid servereid, mille puhul kõik tegevused kontrollitakse ja täidetakse lokaalselt kasutaja mängumaailmas ning server lihtsalt vahendab infot mänguseisude muutustest. [4] Selline meetod teeb petmise lihtsamaks, kuna üks klient võib saata välja mistahes infot ning server ja teised kliendid ei saa seda kontrollida. Mitteautoratiivsete serverite eeliseks on aga see, et ei ole vaja oodata serverilt kinnituste saamist ning tegevusi on võimalik kohe täide viia. See vähendab mängusisest latentsust ja annab kasutajale sujuvama mängukogemuse.

2.4 Probleemid võrgumängudes

Võrgumängudele on iseloomulikud spetsiifilised probleemid, mida üksikmängude korral ei esine. Neid probleeme on võrgumängude arendamisel vaja teadvustada ning nende lahendamisele tähelepanu pöörata.

2.4.1 Petmine

Võrgumängude puhul tuleb arvestada sellega, et kasutajate poolt saadetavad andmed ei pruugi olla alati õiged. Kasutajad võivad proovida mängu petta, näiteks saates enda tegelase kohta ebaõigeid koordinaate ja tehes seeläbi end mängumaailmas kiiremaks või saates infot vastase tabamusest, mida tegelikult ei toimunud. Üksikmängudes pole see üldiselt probleem, aga kuna võrgumängudes on mängijaid rohkem kui üks, siis mõjutab petmine otseselt teiste mängijate mängukogemust ning seetõttu tuleks võrgumängu arendades rakendada meetmeid petmisega toimetulekuks, näiteks kasutades autoratiivseid servereid või andmete valideerimist.

2.4.2 Latentsus

Latentsus ehk hilistumine näitab ooteaega, mis kulub mängus kasutaja poolt antud sisendist kuni tagasisideni. Näiteks aeg, mis möödub kasutaja poolt noole klahvile vajutamisest kuni kontrollitava tegelase liikuma hakkamiseni või hiireklikist püssilasuni. Latentsust leiab igast mängust ja see on vältimatu.

Latentsuse võib jaotada kahte kategooriasse. Esiteks võrguväline latentsus, mis pole probleem mitte ainult võrgumängudes, vaid ka üksikmängudes. Tegu on latentsusega, mis on üldiselt väga väike ning mis on põhjustatud arvuti riistvara ja tarkvara poolt. Siia alla

kuuluvad näiteks riistvara komponentide reageerimisajad, aga ka mängu enda kaadrisagedus või ekraani pikslite uuendamise kiirus. [5]

Teiseks latentsuse allikaks on võrgust tingitud ooteajad, mis on üldiselt suurem probleem. Kuna mängijate vahel saadetavad andmed ei liigu otse ühest arvutist teise, võib latentsusaeg muutuda pikaks. Ooteaja pikkus sõltub mängijate asukohast, pakettide teekonnast ja võrgus kasutusel olevast riistvarast. Esiteks võtab aega juba andmete käsitlemine ehk teise kasutaja asukoha ja teekonna välja selgitamine. Seejärel võtab aega andmete saatmine, see sõltub juba andmete hulgast ja klientide vahemaast. Kuigi paketid liiguvad ainult natukene aeglasemalt kui valguskiirus, tekitab see pikkade vahemaade korral siiski märgatavalt latentsust. Lisaks võivad paketid sattuda ka ruuterites järjekordadesse, kui ruuterid ei suuda andmeid piisavalt kiiresti töödelda. [6]

Kuna latentsus mõjutab otseselt kasutajate mängukogemust, on oluline hoida see võimalikult väiksena ning mängijate jaoks võimalikult märkamatuna. Võrgumängu arendades on võimalik latentsusaega vähendada, aga paljudel juhtudel on see siiski teatud ulatuses paratamatu.

2.4.3 Paketikadu

Üks suuremaid probleeme, millega võrgumänge arendades tuleb arvestada, on paketikadu, mille korral saadetud paketid ei jõua oma sihtpunkti. Selline olukord tekib vahel ruuterites. Kui näiteks ruuter ei jõua pakette piisavalt kiiresti edasi toimetada, võidakse ruuteri saatmisjärjekorrast hakata pakette eemaldama. Sellist olukorda nimetatakse ka pudelikaelaks ehk ruuterile tuleb rohkem pakette sisse, kui ta jõuab välja saata. Lisaks võivad paketid kaduma minna ka vigase riistvara korral, näiteks katkise kaabli või nõrga raadiolaine tõttu. [7]

Paketikadu on vältimatu ning sellega tuleb mängu arhitektuuri kavandades arvestada. Mida lühem on pakettide teekond, seda väiksem on ka paketikadu. Seega sõltuvalt mängus kasutatavast võrguarhitektuurist tuleks kasutajatele mänguserver võimalikult lähedale paigutada.

2.5 Võrguprobleemidega toimetulek

Ebasoodsates tingimustes võib võrgumängudes tekkida palju probleeme ning seeläbi võib mäng muutuda kasutajale mittemängitavaks. Võrguprobleemide parendamiseks ja peitmiseks on olemas erinevaid võtteid, mida arendajad mängudes kasutavad.

2.5.1 Interpoleerimine

Interpoleerimine ehk interpolatsioon on matemaatiline meetod, mis võimaldab leida funktsiooni vahepealsed puuduvad väärtused tema etteantud teadaolevate väärtuste alusel.
Üldiselt ei uuendata klientide infot üle võrgu iga kaadri järel, vaid mingi pikema perioodi tagant. Selle tulemusel hakkavad kliendi mängumaailmas objektid liikumisel hüppama, kuna nende positsiooni uuendatakse teatud aja tagant, alles teatud arvu kaadrite järel. Eriti probleemseks muutub objektide liikumine suure paketikao korral, mis tõttu kahe mänguseisu vaheline aeg võib minna väga pikaks ja ilma interpoleerimiseta hakkavad mänguruumis liikuvad objektid häirivalt hüppama. [8]
Interpoleerimisega on võimalik need objektid sujuvamalt liikuma panna. Võetakse objekti hetkepositsioon ja serverilt saadud uus positsioon ning arvutatakse välja nende vahele jäävad punktid ja liigutatakse objekti mööda neid punkte. Tulemuseks on visuaalselt palju sujuvam liikumine.

2.5.2 Ennustamine

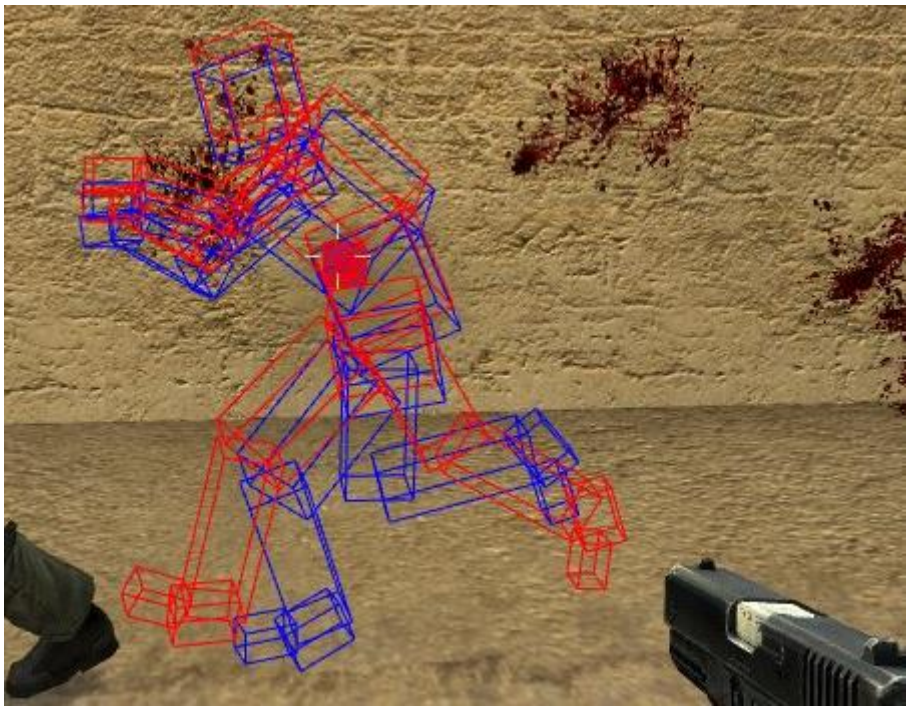
Ennustamist kasutatakse selleks, et tulla toime infovahetuse viivitustega serveri ja klientide vahel. Näiteks võib oletada, et serveri ja kliendi vaheline latentsusaeg on 150 millisekundit. Klient hakkab oma tegelast klaviatuuri klahvi abil liigutama. Serverit teavitatakse sellest. Server saab info kätte ning uuendab kõikides klientides mänguseisu vastavalt saadud infole. See tähendab aga seda, et kasutaja vajutas klaviatuuril liikumise nuppu, aga tegelane hakkab liikuma alles 150 millisekundi pärast, kuna serveri poolt pole kinnitust saadud. Selline viivitus iga tegevuse korral mõjutab oluliselt kasutajakogemust. Sellise probleemi lahendamiseks kasutatakse ennustamist. See tähendab näiteks seda, et kui kasutaja liigutab oma tegelast, siis selle asemel, et oodata serverilt vastust, ennustatakse lokaalselt tegelase uus asukoht ning liigutatakse tegelast kohe pärast klahvi vajutust. Sel viisil saab kasutaja koheselt visuaalselt tagasisidet oma tegevustele ning mäng tundub sujuvam ja naturaalsem. Pärast serverilt vastuse saamist võrreldakse saadud

andmeid ennustatud andmetega. Kui andmed ei kattu, siis kliendi poolel mänguseis parandatakse vastavalt serverilt saadud andmetele. Ennustamine mõjutab mänguseisu ainult kliendi enda arvutis, teiste klientide arvutites liigutatakse tegelasi vastavalt serverilt saadud infole. [9]

Kui ennustatav info erineb serveri omast, proovitakse mänguseisud uuesti sünkroniseerida, parandades lokaalselt kliendi tegevused. Parandamine võib aga tuua kaasa tõrkeid kasutaja mängupildis. Kuna lokaalselt on teada tegelaste kiirused, positsioon ja suund, siis peaksid üldjuhul ennustatavad andmed siiski kattuma serveri andmetega ja seetõttu mängukogemus läbi ennustamise parenema. Üldiselt tekivad ennustamisega suuremad vead alles siis, kui paketikadu võrgus on suurenenud.

2.5.3 Kompenseerimine

Info liikumine kliendi arvutist serverini võtab alati mingi hulga aega, mille tulemusel ei ole serveri ja kliendi mänguseisud kunagi täpselt samad. Kiire iseloomuga mängudes võib see saada probleemiks, kui kasutatakse autoratiivset serverit. Näiteks, tulistamismängus üks mängija laseb vastast. Lask toimub ajahetkel x, aga võrguühenduse viivituse tõttu jõuab info serverisse 200 millisekundit hiljem. Selle aja jooksul võib vastane, keda kasutaja tulistas, olla juba liikunud. Tekib olukord, kus oleks pidanud toimuma tabamus, aga server seda ei registreeri.
Selle probleemi lahendamiseks kasutatakse mõnedes mängudes kompenseerimist. Kompenseerimine tähendab seda, et server ei oma teavet ainult hetkelise mänguseisu kohta, vaid alles hoitakse ka varasemad mänguseisud. Selline mänguseisude ülevaade võimaldab serveril näiteks lasu korral välja arvutada lasu toimumise aja kliendi arvutis ning kontrollida mänguseisude ajaloost, kas lask tabas või mitte (vt. joonis 4). [10]



Joonis 4 Pilt mängust Counter-Strike, kus on sinise ja punasega näha tegelase mänguseisude muutus laskmishetkest kliendi arvutis info jõudmiseni serverisse.

2.5.4 Andmekontroll

Petmise vältimiseks võrgumängudes on vaja veenduda, et kõikide klientide poolt saadetavad andmed ja tegevused oleksid valiidsed. Üks põhilisemaid meetodeid selleks on eelpool mainitud autoratiivsete serverite kasutamine ehk server kontrollib kõike mängus toimuvat. Võimalik on aga kasutada ka klientide poolset andmete valideerimist. Sellise meetodiga eeldab klient alati, et saadetavad andmed on valed ning kontrollib, kas tehtud liigutused või tegevused teise mängija poolt on lubatud. Vigaste andmete korral otsustab klient, kas näiteks keelata tehtud liigutus ja muuta mänguseisu vastavalt sellele või lihtsalt teine mängija mängust eemaldada. Tuleb muidugi arvestada, et vigaste andmete põhjuseks võib olla ka suur latentsus või muud võrguprobleemid. [11]

2.5.5 Andmemahtude optimeerimine

Üks viis latentsuse vähendamiseks on piirata andmemahtusid, mis klientide vahel liiguvad. Selleks on võimalik muuta ajavahemikku, kui tihedalt mänguseise uuendatakse ning ka reguleerida andmete hulka, mis välja saadetakse. See aga sõltub juba arendatavast mängust ja vajadustest. Näiteks käigupõhistes mängudes on infot vaja uuendada ainult käikude vahetusel ning pidevalt andmeid vahendada ei ole mõtet. Kiireloomulistes mängudes, nagu näiteks tulistamismängudes, on mänguseisude uuendamine tähtsam ning

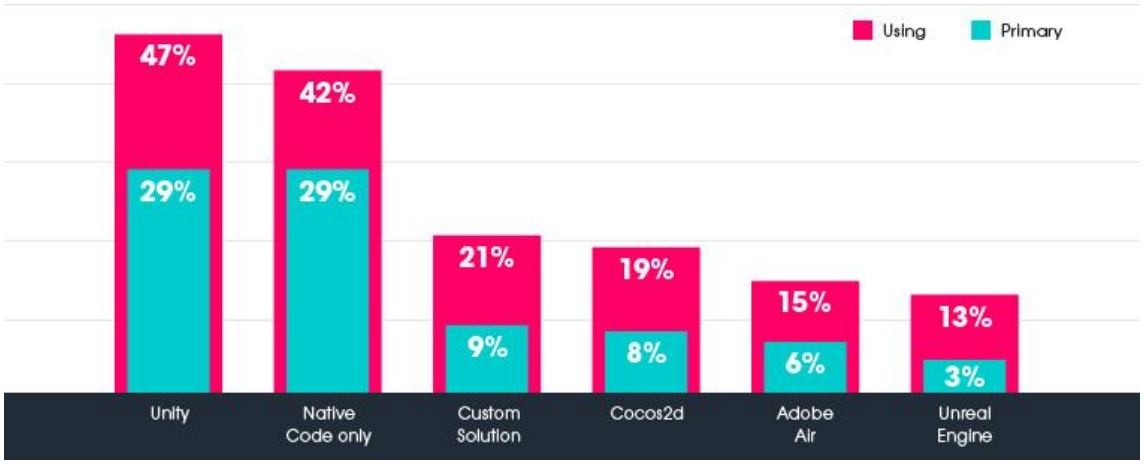
vaja on regulaarsemat andmevahetust. Samas pole näiteks vaja uuendada klientide positsioone, kui kasutaja kontrollitav tegelane seisab. Sel viisil on võimalik serveri poolt vahendatavate andmete hulka piirata. Lisaks on oluline ka info, mida saata. Näiteks kui sünkroniseerida kasutajate kontrollitavate tegelaste andmeid, on vaja saata ainult tegelaste positsioon ja suund, kuhu tegelane vaatab. Sünkroniseerida pole ka mõtet andmeid, mis ei muutu. Näiteks kui mäng on kolmemõõtmeline ja kasutajal on lubatud liikuda või pöörata end ainult teatud telgedel, siis pole teiste telgede andmeid mõtet üle võrgu saata, kuna see tähendab samade andmete korduvat edastamist, mis liigselt suurendab väljasaadetavate ja vastuvõetavate andmete hulka. [12]

3 Kasutatud tehnoloogiad

Töö käigus luuakse Unity mängumootori abil üle võrgu mängitav 3D mitmikmäng. Mäng kasutab erinevaid Unity komponente, sealhulgas ka Unity Unet võrgukomponente, mille abiga toimub üle võrgu andmete edastus ja sünkroniseerimine. Mängu poolt kasutatavad skriptid on kirjutatud C# keeles ning mäng on arendatud Windowsi 32 ja 64 bitistele operatsioonisüsteemidele.

3.1 Unity 3D

Unity 3D on mängude arendusplatvorm ja mootor, mis on loodud aastal 2005 ning on saanud praeguseks hetkeks üheks populaarseimaks 2D ja 3D mängude arenduskeskkonnaks (vt. joonis 5). Unityt kasutavad nii algajad mänguarendajad kui ka suured mängustuudiod, kuna see pakub rohkelt võimalusi ning on tasuta kättesaadav kõikidele arendajatele. [13]



Joonis 5 Unity kasutatavus võrreldes muude turul saadavate mängude arendusplatvormidega

3.1.1 Võimalused

Unity mängumootoris on olemas palju erinevaid võimalusi - nagu näiteks füüsika mootor, maastiku redigeerija, kokkupõrgete tuvastus, helide ja valgustuse haldamine, varjude töötlus ning palju muud. Kõiki komponente on lihtne projektis rakendada ning kohandada. Lisaks võimaldab Unity ka kasutajal lisada mängule enda kirjutatud skripte ja komponente. Seda on võimalik teha kolmes erinevas programmeerimiskeeles: C#, Javascript ja Boo.

Häid lisavõimalusi pakub veel Unity Asset Store, kust on võimalik oma mängule erinevaid tekstuure, kujundust, komponente ja mänguobjekte alla laadida, mis võivad olla nii tasulised kui ka tasuta. See võimalus on suureks abiks paljudele väiksematele arendajatele, kellel ei ole näiteks disaini või 3D mudelite tegemise osas kogemust. [14] *Asset store*'i võivad kõik mängude arendajad oma loomingut ülesse panna. Selle tulemusena on sealne variantide valik lai ja pakub palju erinevaid võimalusi. Lisaks kõigele muule on võimalik osta ka komponente, mis ei ole mõeldud mängudes kasutamiseks, vaid laiendavad otseselt Unity funktsionaalsust.

3.1.2 Multiplatvormilisus

Üheks suureks Unity populaarsuse põhjuseks on ka multiplatvormilisus, millele on arendajad palju tähelepanu pööranud. Unity võimaldab arendada mänge rohkem kui kahekümnele erinevale platvormile, mille hulka kuuluvad erinevate arvutite ja telefonide operatsioonisüsteemid, konsoolid, virtuaalreaalsuse seadmed ning ka näiteks nutitelerid. Valminud projekti on seega võimalik välja lasta mitmele platvormile ning sellega haarata potentsiaalselt suurem kasutajaskond. [15]

3.1.3 Litsenseerimine

Unity pakub mitmesuguseid tasulisi teenuseid ja võimalusi, aga mängumootor kõikide võimalustega on tasuta kättesaadav kõikidele kasutajatele Unity individuaalkasutuseks mõeldud versiooniga. Lisaks on kogu Unityga tehtud looming autori enda oma ning Unity ei küsi selle pealt litsentsitasu, isegi kui projekt on loodud tasuta versiooniga.

Lisaks on võimalik kasutada tasulise teenusena mitmeid täiendavaid võimalusi, näiteks nagu pilveteenused, analüütikavahendid, ligipääs lähtekoodile, jõudluse aruanded ning lisavõimalused projektide arendamiseks meeskonnas. [16]

3.2 MonoDevelop

Käesolevas bakalaureusetöös kasutatavad skriptid on kirjutatud MonoDevelop IDEs. Tegu on Unity mängumootoriga kaasatuleva tarkvaraga, mis on kohandatud Unity arendajate poolt spetsiaalselt skriptide kirjutamiseks Unity keskkonnas. MonoDevelop kombineerib tekstiredaktori omadused lisavõimalustega, sh vigade tuvastamine ja projekti haldamise vahendid. [17].

3.3 C#

Unity võimaldab programmeerimiseks kasutada kolme keelt: JavaScript, C#, ja Boo. Käesolevas töös kasutatakse skriptimise keelena C# kuna antud keel on hetkeseisuga kõige paremini dokumenteeritud ning pakub ka programmeerimiskeelena rohkem võimalusi kui JavaScript või Boo.

4 Rakenduse ülesehitus

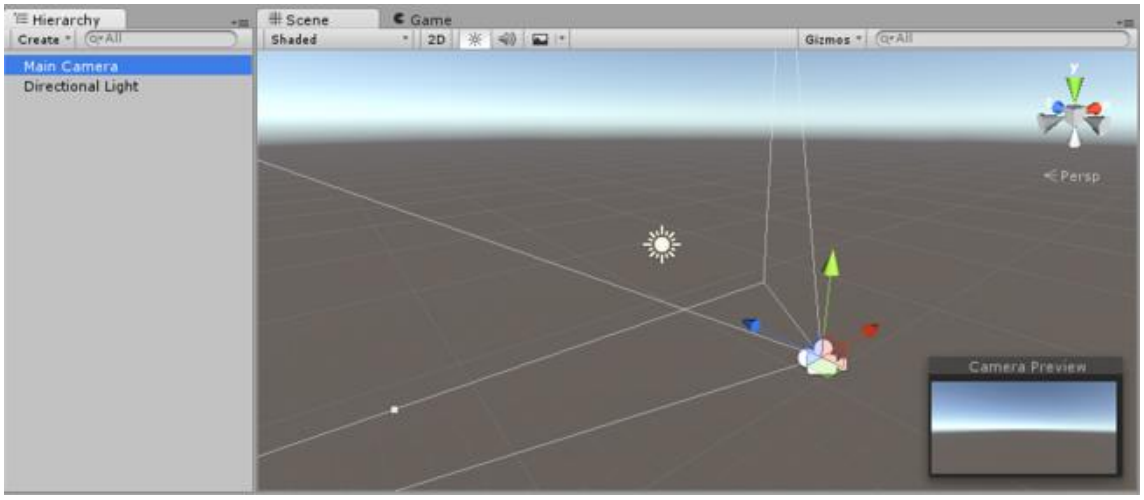
4.1 Peamised kasutatavad Unity komponendid

4.1.1 Komponendid

Komponendid on funktsionaalsed osad, mida on võimalik mängus olevatele objektidele külge panna. [18] Iga komponent täidab mingit kindlat rolli objekti töös. Näiteks üks komponent võib anda objektile värvi või tekstuuri, teine komponent vastutab liikumise eest ning kolmas kontrollib põrkumisi teiste objektidega. Objektidele võib vastavalt vajadusel komponente juurde lisada või neid välja vahetada.

4.1.2 Scene

Stseenid on 3D ruumid, mis sisaldavad endas mänguobjekte. Tühjas stseenis on alati kaamera ja valgusallikas (vt. joonis 6). Kaamera peab olema igas stseenis ning selle paigutusest ja seadetest sõltub, kuidas kasutaja mänguruumi näeb. Stseenidesse võib vastavalt vajadusele lisada juurde mänguobjekte, et luua näiteks menüüsid või individuaalseid tasemeid. [18] Stseenid ei ole üksteisest sõltuvad ning nende vahel on võimalik vastavalt vajadusele liikuda.



Joonis 6 Tühi stseen

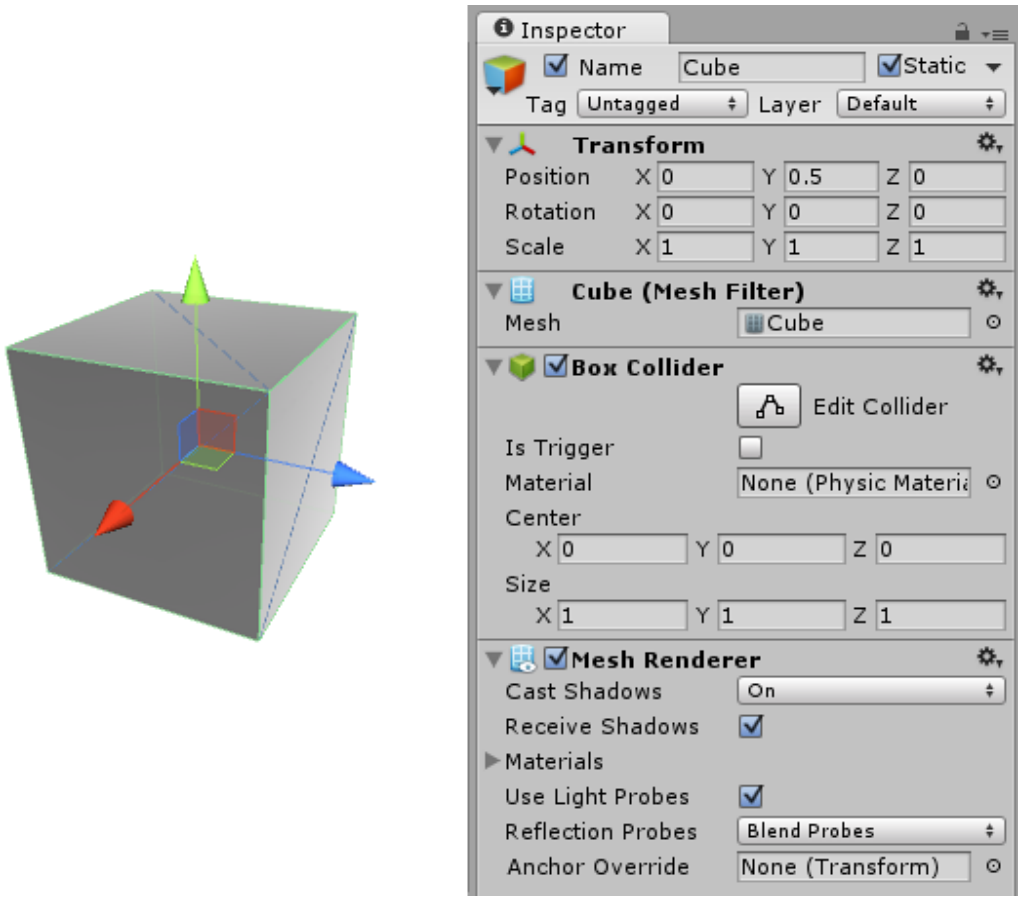
4.1.3 Prefab

Prefabid on mõeldud mänguobjektide hoiustamiseks. Need hoiavad endas objekti komponente ja väärtusi, et neid oleks võimalik stseenides korduvkasutada. Muutes

prefab'i väärtusi või komponente, kanduvad kõik muudatused üle objektidele mängumaailmas. [19] Loodud mängus on *prefab*'ina kasutusel näiteks mängija tegelase objekt.

4.1.4 GameObject

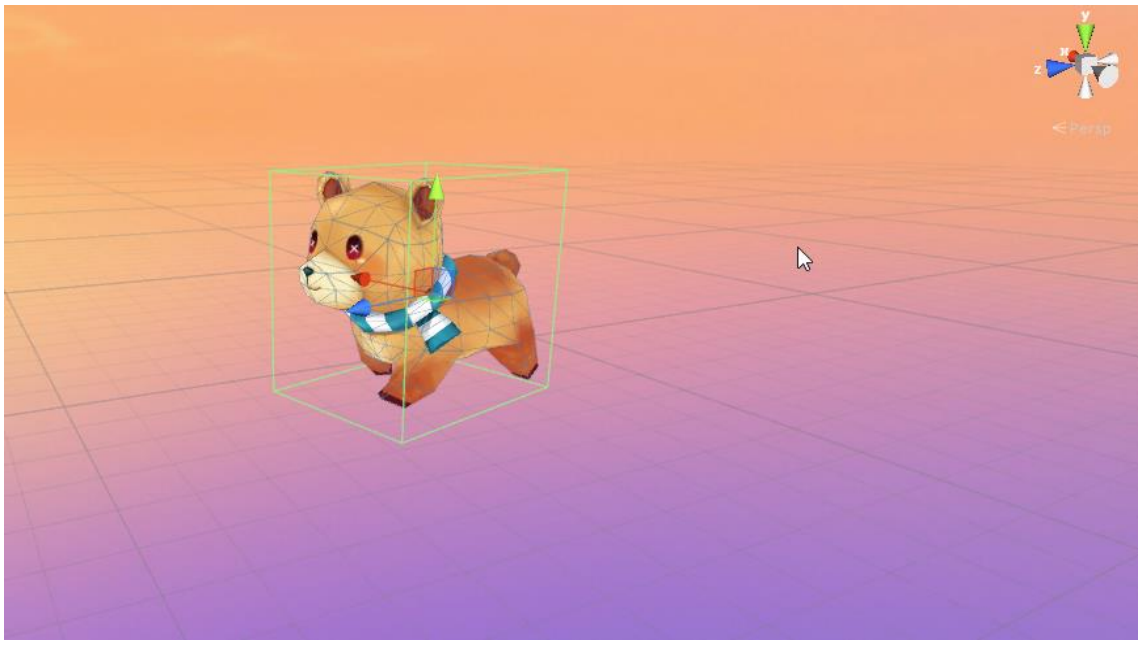
Mänguobjektid on fundamentaalsed objektid Unitys, mis esitavad tegelasi, rekvisiite ja dekoratsioone. Nad ei oma iseseisvalt palju funktsiooni, aga käituvad konteinerina komponentidele, mis rakendavad päris funktsionaalsust [20]. Mänguobjektile võib sõltuvalt vajadusest lisada juurde teisi Unitys olevaid komponente või skripte. Joonisel näide lihtsast kuubiku mänguobjektist. (vt joonis 7)



Joonis 7 Näide mänguobjektist Unitys

4.1.5 Collider

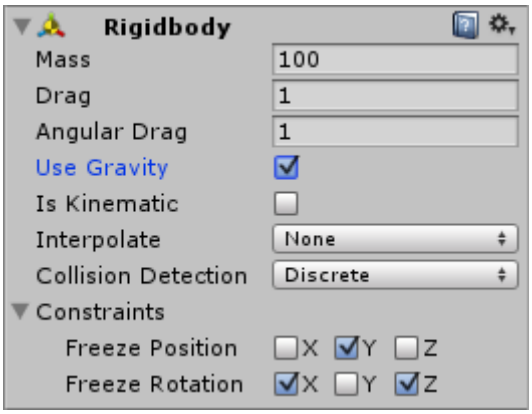
Collider on nähtamatu komponent, mida kasutatakse kokkupõrgete tuvastamiseks teiste objektidega (vt joonis 8). Üldiselt on Collider sama suurusega kui objekt, mille külge see kinnitatud on. [21]



Joonis 8 Mängus kasutatava tegelase Collider kujutatuna roheliste äärtega kuubina

4.1.6 Rigidbody

Rigidbody on üks põhilisi Unity komponente, mida kasutatakse, et anda objektile füüsikalised omadused mänguruumis (vt joonis 9). Objekte, millel on küljes RigidBody komponent, on võimalik mõjutada näiteks gravitatsiooni või plahvatustest tekkivate füüsikaliste jõududega. [22] Projektis on Rigidbody komponent olemas kõikidel kasutaja poolt kontrollitavatel tegelastel ning ka erinevatel objektidel, mida tegelased tekitada ja hävitada saavad.



Joonis 9 Näide Rigidbody komponendist Unitys

4.1.7 NetworkManager

Network manager võimaldab käivitada serveri ja klientidel serveritega üle võrgu ühineda. Iga kliendi mängus on olemas NetworkManager komponent, mis tegeleb selle sama kliendi andmete vahendamisega. [23]

4.1.8 NetworkTransform

Komponent, mis sünkroniseerib üle võrgu objekti positsiooni, mille külge see kinnitatud on. [24]

4.1.9 NetworkIdentity

NetworkIdentity on komponent, mida kasutatakse, et üle võrgu saata objekte, mille külge see lisatud on. NetworkIdentity sünkroniseerib objekti informatsiooni üle võrgu. [25] See kinnitatakse üldiselt objektide külge, mida on vaja serveri poolt tekitada (nagu näiteks kuulid).

4.2 Peamised autori poolt loodud komponendid

Lisaks Unitys olevatele komponentidele, võib lisada mänguobjektidele ka enda poolt kirjutatud komponente ehk skripte. Valminud mängus on kasutusel 20 autori poolt loodud skripti, millest alljärgnevalt on välja toodud 7 põhilisemat.

4.2.1 Skriptid

Skriptid lubavad kasutajal kirjutada oma mängule vajaminevaid komponente, kasutades isetehtud funktsioone ja klasse. Nende skriptide abil on näiteks võimalik kontrollida ja tekitada graafilisi efekte või füüsikalisi liikumisi, kõik vastavalt kasutaja soovidele. [26] Lisaks pakub ka Unity enda poolt palju erinevaid funktsioone ja klasse, mida võib kasutaja poolt loodud skriptides kasutada. Näiteks *Update()* funktsioon, mis kutsutakse esile iga kaadri vahetusel mängus või *OnCollisionEnter()*, mis kutsutakse välja, kui objekt, mille küljes skript on, põrkub kokku mõne teise objektiga.

4.2.2 PlayerController

PlayerController on kasutaja kontrollitava tegelase külge pandud komponent, mis haldab kasutaja liikumist ja võimeid. Lisaks sünkroniseerib PlayerController üle võrgu teiste kasutajate nimed ja tegelaskujud.

4.2.3 PlayerMovementSync

PlayerMovementSync sünkroniseerib üle võrgu saadetavaid tegelaste asukohtade koordinaate. Kasutab interpoleerimist, et muuta tegelaste liikumine sujuvamaks.

4.2.4 PlayerRotationSync

PlayerRotationSync skript sünkroniseerib kõik tegelaste poolt tehtud pöörded. Nii nagu liikumise sünkroniseerimisel, kasutatakse ka siin interpoleerimist, mis teeb liikumised sujuvamaks.

4.2.5 HealthManager

HealthManager on elude haldamise skript. Kui tegelane saab pihta mõne vastase võimega, siis kaotab ta elusid. Elude haldust teeb ainult server ning see saadab klientidele nende elud hetkel.

4.2.6 BulletController

Tegelaste poolt tekitatud võimeid haldab BulletController, mis määrab ära liikumise kiiruse, suuna ja eluea. Lisaks kontrollib see ka kokkupõrkeid teiste mänguobjektidega ning saadab serverile infot, kui kokkupõrge toimus näiteks mõne tegelasega ning mingilt objektilt on vaja elusid maha võtta.

4.2.7 MenuController

MenuController on kontroller, mis haldab menüüsid. Mängus on kasutusel erinevad menüüd. Ka selles skriptis asuvad erinevad funktsioonid, mis on seotud stseenis olevate nuppudega. Skript mängib helisid ja muudab stseenis menüü olekuid vastavalt tegevustele.

4.2.8 CharacterSelectionController

CharacterSelectionController on skript, mis käivitatakse mängija sisenemisel *main* stseeni ning võimaldab kasutajal valida endale mängimiseks tegelane. Pärast tegelase valikut viiakse kasutaja edasi mängumaailma ning valitud tegelase andmed edastatakse *PlayerController* skriptile, mis seejärel sünkroniseerib need andmed teiste klientidega.

4.3 Stseenid

Mängus on kolm stseeni, mille vahel on kasutajal võimalik mängu sees liikuda.

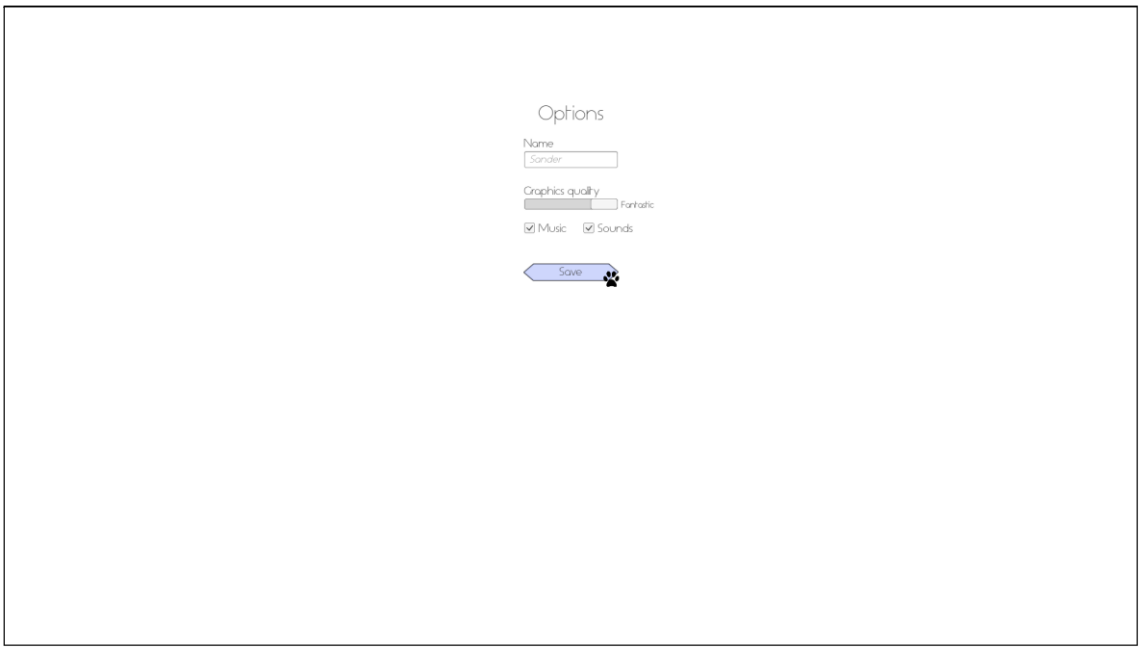
4.3.1 Menu

Menu stseen on mängu alguse stseen kuhu kasutaja jõuab mängu käivitades. Seal on kasutajal võimalik mängust väljuda, alustada uus mäng, liituda olemasoleva mänguga või minna seadetesse (vt. joonis 10).

Joonis 10 Mängu avamenüü

4.3.2 Options

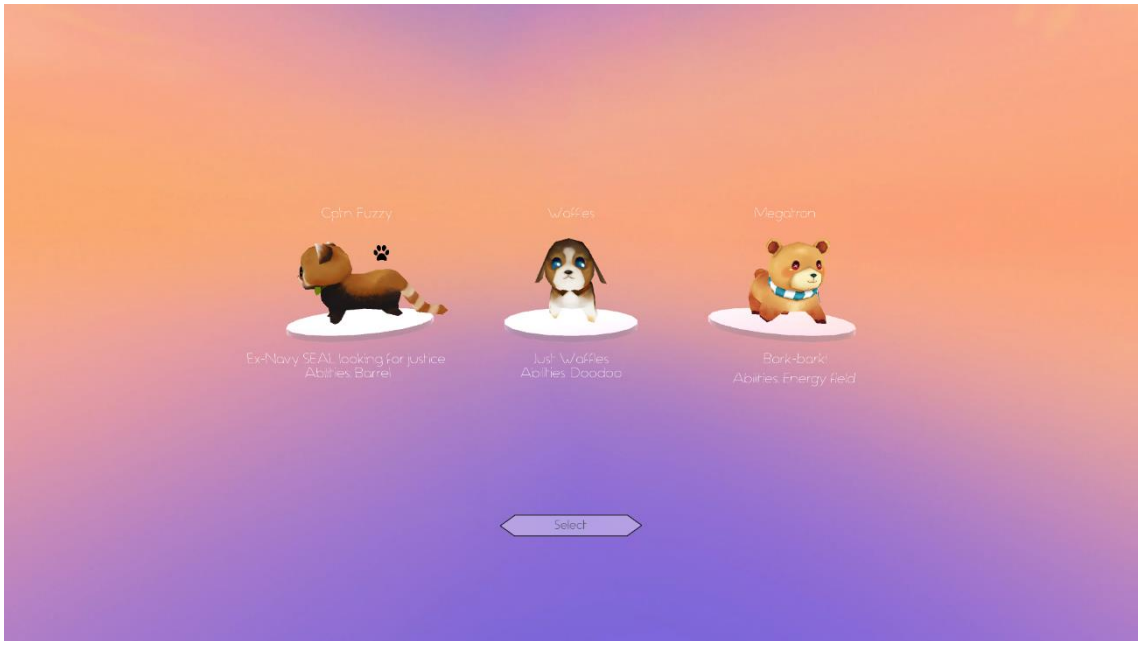
Seadete stseen hõlmab endas erinevaid seadeid, mida kasutajal on võimalik oma soovidele vastavalt muuta. Muuta on võimalik graafika taset, heli ja muusika olemasolu. Muuta saab ka kasutaja nime, mida mängus kõikidele teistele osavõtjatele kuvatakse (vt joonis 11).



Joonis 11 Seadete stseen

4.3.3 Main

Main on põhistseen, kus toimub mängu tegevus. Kui mängija on jõudnud main stseeni, on tal võimalik valida tegelane (vt. joonis 12) ning minna edasi mängumaailma (vt joonis 13 ja 14), kuhu teised mängijad saavad liituda üle võrgu. Mängumaailmas saab tegelastega ringi liikuda ning üksteisega võidelda. Lisaks on mänguruumis liikumatud purustatavad objektid.



Joonis 12 Tegelase valik mängu sisenemisel

Joonis 13 Mängu maailm, kasutaja poolt valitud tegelasega

Joonis 14 Escape klahviga mängu ajal avatav menüü

4.4 Rakenduse kirjeldus

4.4.1 Mängu kirjeldus

Loodud mäng on 3D võitlusmäng, kus iga mängija omab kontrolli ühe looma üle. Erinevaid loomi on valikus kolm ning iga loom omab erinevaid võimeid. Kõik tegelased

alustavad mängu kindla hulga eludega ning üksteiselt on võimalik tegelastele antud võimetega elusid vähemaks võtta. Mängu tegevus toimub piiratud pargialal, mis on ümbritsetud aiaga, millest mängijad välja ei pääse. Mänguplatsile on ka lisatud lõhutavaid kive ja muid objekte, mille taha on tegelastel võimalik varjuda. Mängu käivitades on kasutajal võimalik luua uus mäng või liituda olemasolevaga. Uue mängu loomisel hakkab kasutaja klient-serveriks ning teiste mänguga liitujate info hakkab käima läbi selle sama mängija. Teised liitujad on kliendid, kes hakkavad enda kontrollitava tegelase liikumise ja tegevuste infot üle võrgu serveriga vahetama.

4.4.2 Mängu töökäik

Mängu sisenedes valib kasutaja tegelase. Uute kasutajate liitumisel sünkroniseeritakse kõigepealt omavahel tegelaste nimed ja tegelaste mudelid, millega iga mängija mängib. Nimi on lokaalselt salvestatud iga kliendi arvutis ning on muudetav seadetes. Seejärel hakatakse perioodiliselt saatma infot tegelaste liikumisest, rotatsioonidest, võimete kasutamisest ning eludest. Süsteem saadab väikese intervalliga andmeid kasutaja positsioonist ning nendele andmetele vastavalt liigutatakse iga kasutaja arvutis tegelasi. Kasutatakse ka interpoleerimist. Andmete saatmine on limiteeritud ning näiteks tegelase peatumise korral peatatakse andmete saatmine. Kontrollitakse ka võrgu seisu ning kehvade tingimuste korral hakatakse positsioone lisama massiivi ning vastavalt sellele proovitakse liikumist ennustada ning ka sujuvamaks teha.

Võime kasutamise korral antakse serverile teada, et tegelane kasutas võimet ning server saadab käsu kõikidele klientidele see võime tekitada ka kohalikes arvutites. Seejärel kontrollitakse võime kollisiooni teiste mänguobjektidega ning teise mängija tabamise korral võetakse antud tegelaselt elupunkte maha. Kui tegelase elud langevad alla nulli, annab server klientidele märku, et tegelane uuesti taastada algpositsioonidel. Elude üle on kontroll ainult serveril ning klientidel pole elusid võimalik muuta.

Lisaks on mängu ajal võimalik avada *Escape* klahviga menüü, kust on võimalik mängust väljuda, avada seaded või mängujuhised. Kasutaja poolt määratud seadeid hoiustatakse lokaalselt kasutades Unity *PlayerPrefs* klassi, mis võimaldab sõne ja numbrilisi väärtusi salvestada arvuti kõvakettale mängus kasutamiseks.

5 Töö analüüs

5.1 Vastavus ülesandele

Töö põhiliseks ülesandeks oli valmistada reaalselt töötav võrgumäng, mis lõpptulemina ka õnnestus. Unity mängumootori baasil valminud mäng võimaldas praktiliselt näidata teoreetilises pooles kirjeldatud probleeme ning ka lahendusi nendele probleemidele, sh andmete valideerimine ja interpolatsioon.

Töös said ka kirjeldatud põhilised võrguarhitektuurid ning analüüsitud mängude loomisel tekkivad probleemid ja peamised võimalused, mida kasutatakse võrgumängude mängitavuse parandamiseks.

Mängu osas võiks edaspidi rakendada erinevaid võrguarhitektuure ning analüüsida tulemust ning teostatavust Unitys. Hetkel töötab mäng klient-server arhitektuuri põhimõttel, aga tulevikus võiks katsetada ka näiteks partervõrgu lahendust. Lisaks võiks mängu teha terviklikumaks, lisades näiteks valikusse rohkem tasemeid, mängule juurde rohkem heliefekte ning täiendada ka disaini.

Kokkuvõtteks võib siiski väita, et lõpptulemus vastab püstitatud ülesandele ning said täidetud ja realiseeritud kõik eesmärgid, mis lõputööga sooviti saavutada.

6 Kokkuvõte

Töö eesmärgiks oli luua üle võrgu mängitav 3D mäng ning tutvustada takistusi võrgmängude arendusel ning pakkuda ka lahendusi ja parendusi nende takistustega toimetulekuks.

Mäng valmis Unity mängumootoril, mis võimaldas oma Unet komponentidega valmistada üle võrgu töötava tervikliku mängu. Mängijatel on võimalik üle võrgu omavahel mängida ning mängu näitel oli võimalik ka rakendada teoreetilises pooles analüüsitud erinevaid võrguprobleemidega toimetuleku metoodikaid.

Töös said käsitletud kõik põhilised probleemid, millega võrgumängude arendamisel kokku puututakse ning tutvustati ka lahendusi nende probleemidega toimetulemiseks ja nende peitmiseks või leevendamiseks.

Kokkuvõtteks võib öelda, et töö täitis oma eesmärki. Mängu arendamine võimaldas tuua välja esinevaid probleeme ning lõpptulemusest on näha, et töös analüüsitud probleemidega toimetuleku võtted on toimivad ning praktikas teostatavad Unity mängumootori abiga.

Viited

[1] M. C. Angelides ja H.Agius, „Client-Server Architectures,“ *Handbook of Digital Games*, 2014, pp. 177-178.

[2] M. C. Angelides ja H.Agius, „Peer-to-Peer Architectures,“ *Handbook of Digital Games*, 2014, pp. 178-179.

[3] M. C. Angelides ja H.Agius, „Distributed Architectures,“ *Handbook of Digital Games*, 2014, pp. 179-182.

[4] Unity, „http://docs.unity3d.com/Manual/net-HighLevelOverview.html,“ [Võrgumaterjal].

[5] J. Glazer ja S. Madhav, „Non-Network Latency,“ *Multiplayer game programming: Architecting networked games*, 2015, pp. 200-202.

[6] J. Glazer ja S. Madhav, „Network Latency,“ *Multiplayer game programming: Architecting networked games*, 2015, pp. 202-204.

[7] J. Glazer ja S. Madhav, „Packet Loss,“ *Multiplayer game programming: Architecting networked games*, 2015, pp. 206-209.

[8] Valve, „https://developer.valvesoftware.com/wiki/Interpolation,“ [Võrgumaterjal].

[9] Glenn Fiedler , „http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/,“ [Võrgumaterjal].

[10] Valve, „https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking,“ [Võrgumaterjal].

[11] J. Glazer ja S. Madhav, „Input Validation,“ %1 *Multiplayer game programming: Architecting networked games*, 2015, pp. 270-271.

[12] MuchDifferent, „http://developer.muchdifferent.com/unitypark/uLink/MinimizingBandwidth,“ [Võrgumaterjal].

[13] L. Ivanov, „3D game development with unity in the computer science curriculum“.

[14] P. Polsinelli, „http://designagame.eu/2013/12/unity-popular-videogame-development/,“ [Võrgumaterjal].

[15] Unity, „https://unity3d.com/unity/multiplatform,“ [Võrgumaterjal].

[16] Unity, „https://unity3d.com/get-unity,“ [Võrgumaterjal].

[17] Unity, „http://docs.unity3d.com/Manual/MonoDevelop.html,“ [Võrgumaterjal].

[18] Unity, „http://docs.unity3d.com/Manual/CreatingScenes.html,“ [Võrgumaterjal].

[19] Unity, „http://docs.unity3d.com/Manual/Prefabs.html,“ [Võrgumaterjal].

[20] Unity, „http://docs.unity3d.com/Manual/class-GameObject.html,“ [Võrgumaterjal].

[21] Unity, „http://docs.unity3d.com/Manual/CollidersOverview.html,“ [Võrgumaterjal].

[22] Unity, „http://docs.unity3d.com/Manual/RigidbodiesOverview.html,“ [Võrgumaterjal].

[23] Unity,
„http://docs.unity3d.com/ScriptReference/Networking.NetworkManager.html,“
[Võrgumaterjal].
[24] Unity,
„http://docs.unity3d.com/ScriptReference/Networking.NetworkTransform.html,“
[Võrgumaterjal].
[25] Unity,
„http://docs.unity3d.com/ScriptReference/Networking.NetworkIdentity.html,“
[Võrgumaterjal].
[26] Unity, „http://docs.unity3d.com/Manual/ScriptingSection.html,“ [Võrgumaterjal].