Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jani Brezavšček

Razvoj mrežne večigralske igre

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Peter Peer
Asistent: as. Bojan Klemenc, univ. dipl. inž.

Ljubljana 2011

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00147/2011
Datum: 02.09.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JANI BREZAVŠČEK**

Naslov: **RAZVOJ MREŽNE VEČIGRALSKE IGRE**
DEVELOPMENT OF A NETWORK MULTIPLAYER GAME

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V zadnjem desetletju se smo priča razmahu mrežnih večigralskih iger, predvsem množičnih večigralskih iger, ki so postale precej vabljiv trg. Raziščite področje mrežnih večigralskih računalniških iger in razvijte večigralsko igro. Proučite možne arhitekture mrežnega dela igre. Pri pregledu področja bodite pozorni na problematiko zakasnitve paketov in varnosti.

Mentor:
doc. dr. Peter Peer

Dekan:
prof. dr. Nikolaj Zimic

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jani Brezavšček, z vpisno številko **63070218**, sem avtor diplomskega dela z naslovom:

Razvoj mrežne večigralske igre

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peera; asistent je bil Bojan Klemenc
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30. septembra 2011 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Petru Peeru in as. Bojanu Klemencu za pomoč in usmerjanje pri izdelavi diplomske naloge. Zahvaljujem se tudi najbližjim za podporo, lektorju, ki mi je slovnično pregledal diplomsko nalogo in vsem, ki so kaj prispevali pri realizaciji diplomske naloge.

Kazalo

Seznam uporabljenih kratic in simbolov

- 2D (angl. Two dimensional) dvorazsežni,
- 3D (angl. Three dimensional) trirazsežni,
- API (angl. application program interface, application programming interface) programski vmesnik,
- FPS (angl. First person shooter) prvoosebna streljačina,
- IP (angl. Internet protocol) medmrežni protokol,
- MMORPG (angl. Massive multiplayer online role playing game) igra z množičnim večigralskim spletnim igranjem vlog,
- MTU (angl. Maximum transmission unit) največja prenosna enota,
- RTS (angl. Real-time strategy) realnočasovna strategija,
- SDK (angl. Software Development Kit) orodje za razvoj programske opreme,
- TCP (angl. Transmission control protocol) povezavni protokol prenosne plasti v protokolnem skladu TCP/IP,
- TCP/IP (angl. Transmission Control Protocol/Internet Protocol) standardiziran sklad protokolov, na katerem temelji internet,

- UDP (angl. User data protocol) nepovezavni protokol transportnega sloja v protokolnem skladu TCP/IP.

Povzetek

V zadnjih letih opažamo precejšnjo porast mrežnih večigralskih računalniških iger. Predvsem pa je to področje zanimivo tudi iz tržnega vidika. Vendar pa se pri razvoju mrežne večigralske igre srečamo tudi s problemi, ki jih je potrebno rešiti. Namen diplomske naloge je tako predstaviti izdelavo računalniške igre, ki ponuja možnost mrežnega večigralskega načina igranja, predstaviti probleme, s katerimi se srečamo pri razvoju, in rešitve teh problemov. Po pregledu poteka razvoja računalniške igre in pregledu zgodovine večigralskih sistemov se osredotočimo na vrste arhitektur omrežnih večigralskih sistemov. Pogledamo si nekaj nevarnosti, s katerimi se sooča uporabnik omrežnega večigralskega sistema, ter podrobneje predstavimo problem zakasnitve paketov pri mrežnem igranju in rešitve tega problema. Pri realizaciji večigralskega sistema si lahko pomagamo z obstoječimi knjižnicami, ki nam olajšajo delo. Tako je s pomočjo brezplačne omrežne knjižnice ENet v okviru diplomske naloge nastala preprosta mrežna igra, ki predstavlja praktično realizacijo celotnega večigralskega sistema.

Ključne besede: večigralske igre, mrežno igranje, kompenzacija zakasnitve paketov, razvoj iger

Abstract

In the past few years there was a significant rise of network multiplayer games, which makes network computer games a very interesting market. But there are also some issues we have to solve when developing a multiplayer game. The purpose of this thesis is to present how to make a network multiplayer computer game, present the issues we face when developing them and review the solutions to the issues. After reviewing the development process and the history of multiplayer systems we focus on the different architectures of network multiplayer systems. We point out some dangers the user faces when using a multiplayer network system and make a detailed review of the problem of packet delay and solutions to this problem. We can use existing libraries when implementing the network subsystem of the game. To demonstrate the network subsystem we made simple game using free network library ENet.

Keywords: multiplayer games, network games, packet delay compensation, game development

Poglavje 1

Uvod

Računalniške igre postajajo najbolj priljubljeno razvedrilo, zato ni nič čudnega, da je tudi v Sloveniji vedno več podjetij, ki se ukvarjajo z izdelavo le-teh. Poleg tega pa je dandanes vedno več podjetij, ki razvijajo igre z možnostjo večigralskega načina igranja. Eno izmed njih je na primer neodvisno podjetje Nimbly Games, ki je z izidom računalniške igre Altitude[1], ki ponuja mrežno večigralstvo bitk vojaških letal z zelo preprostim grafičnim stilom pokazalo, da se z nekaj volje in znanjem o razvijanju računalniških iger lahko naredi zelo uspešno igro za skoraj vsa računalniška okolja [17]. Razvoj večigralskih iger, ki jih lahko igra več igralcev tudi, ko niso skupaj v istem prostoru, je lahko tudi zelo dobičkonosen posel. Tako so cilji diplomske naloge razložiti bralcu, zakaj je potreben varen večigralski sistem, kako se preko omrežja pošilja podatke ter ga seznaniti z zgodovino in sestavo večigralskih sistemov. Potek same izdelave igre je predstavljen z uporabo brezplačne knjižnice OGRE 3D, v katero smo integrirali mrežni večigralski sistem s pomočjo brezplačne knjižnice ENet.

V poglavju 2 je predstavljen postopek izdelave računalniške igre, kratek pregled akterjev, ki so potrebni za izdelavo igre, in na kratko opisane komponente računalniške igre. V poglavju 3 si bomo pogledali zgodovino večigralstva, trenutno stanje na področju večigralstva in kakšne so možnosti

[1] http://altitudegame.com/

zboljšave tehnologije v prihodnosti. Osnove, ki jih je potrebno vedeti, preden se lotimo programiranja večigralske mrežne igre, bomo predstavili v poglavju 4. Tako je potrebno izbrati ustrezno arhitekturo. Pomembno je tudi, da podatke pred pošiljanjem preko mreže ustrezno pripravimo. V poglavju je govora tudi o varnosti in nevarnosti večigralskih iger. Možne nevarnosti žal zelo hitro naraščajo, zato je včasih potrebno posredovati na sporne načine. Poleg tega si bomo pogledali praktične primere iz priljubljenih računalniških iger. Tisto, kar nikakor ne sme manjkati v večigralskem sistemu, bomo podrobneje pogledali v podpoglavju 4.7, hkrati pa predstavili težave, s katerimi se srečamo pri realizaciji gladko tekočega večigralskega sistema. Zaradi nizkega prenosa podatkov od enega do drugega računalnika nastajajo ozka grla, ki se jih da odpraviti z različnimi uspešnimi pristopi, opisanimi v tem podpoglavju. V poglavju 5 bomo predstavili realizacijo večigralske igre s praktično uporabo omrežne knjižnice ENet.

Poglavje 2

Postopek izdelave računalniške igre

Postopek izdelave računalniške igre je vse prej kot mačji kašelj. Gradbenih elementov in metod je preveč, da bi vse opisali. Tako je predstavljenih le nekaj glavnih pri izdelavi igre v 3D prostoru. Pred izdelavo računalniške igre se je potrebno najprej odločiti, kakšna je naša ciljna skupina oziroma kdo bo igral našo igro. Poleg tega moramo proučiti smotrnost razvoja za različna razvojna okolja (nekatera orodja so namenjena razvoju iger na več računalniških okoljih, nekatera samo za določene). S tem mislimo na razlike v načinih razvijanja iger za različne operacijske sisteme na osebnih računalnikih, konzolah ter mobilnih napravah. Novejše 3D računalniške igre so pogosto grajene na ločenem grafičnem pogonu, ki uporablja enega od dveh API-jev: OpenGL se lahko uporablja na večini sistemov, na Microsoftovih operacijski sistemih pa je tu dodatno še DirectX. Še pred izbiro računalniških okolij pa je potrebno imeti neko zamisel in načrt, kakšno računalniško igro bomo ustvarjali. Pri izdelavi računalniške igre lahko sodeluje več članov (oziroma vlog), zadolžitve posameznih članov pa se lahko izvajajo vzporedno. Pri tem velja še omeniti, da lahko ena oseba igra več vlog in eno vlogo pri razvoju igre lahko igra več oseb. V nadaljevanju je predstavljenih nekaj vlog in njihove zadolžitve ter orodja, ki jih uporabljajo.

Izdelava računalniške igre se začne pri dokumentu zasnove igre (angl. Game design document; GDD). Dokument vsebuje zgodbo, like, ki igrajo v njej, okolje stopenj ali igre, potek igranja igre (angl. gameplay), idejno zasnovo zvoka in glasbe, izgled uporabniškega vmesnika in ukaze, ki jih igra potrebuje [15].

Pri izdelavi resničnoživljenjske simulacijske igre je naloga grafičnega oblikovalca (angl. graphics designer) risanje komponent za uporabniški vmesnik (angl. user interface) in izbirnike (angl. menu). Pri ostalih zvrsteh iger pa je on običajno prvi, ki začne z delom. Narisati mora skice poteka igranja, kako naj bi izgledali 3D objekti v igri, v 2D igrah pa tudi končne slike igralnih objektov. Njegova orodja so običajno vektorski in bitni risarski programi, včasih pa tudi nariše z navadnim svinčnikom na list papirja.

V primeru, da se odločimo izdelati 3D igro, je potrebno izdelati 3D prvine igre. Ta korak običajno razvijalcem izvorne kode izdelajo modelerji. Ko je 3D prvina igre narejena, se jo izvozi z vgrajeno funkcijo ali pa s posebnim vtičnikom, ki je na voljo na spletni strani posamezne aplikacije. Orodja, ki jih modelerji uporabljajo, so običajno Maya[1], 3D Studio Max[2] ali brezplačni Blender[3].

Za vključitev 3D prvin v igro potrebujemo grafični pogon ali knjižnico za izrisovanje. Takšna knjižnica je lahko integrirana v različne pogone, ki ponujajo še veliko več kot samo izris objektov v 3D prostoru. Med najbolj priljubljenimi je Unreal Development Kit (UDK)[4], ki je plačljiv za podjetja, vendar le, ko so prihodki podjetja višji od neke vrednosti. Pogon podpira izdelavo iger na mobilnih napravah iOS in iger na operacijskem sistemu Windows. Unity[5] je tudi precej uveljavljen večnamenski pogon, ki je na voljo za izdelavo iger za praktično vse bolj razširjene operacijske sisteme. Med priljubljenimi pogoni za računalniška okolja PlayStation, Xbox in za operacijski

[1] http://usa.autodesk.com/maya/
[2] http://usa.autodesk.com/3ds-max/
[3] http://www.blender.org/
[4] http://www.udk.com/
[5] http://unity3d.com/

sistem Windows je ta trenutek tudi CryEngine[6]. Samostojna knjižnica za izrisovanje je tudi OGRE 3D[7]. Z uporabo knjižnice OGRE 3D za izris oziroma upodabljanje 3D prostora si zelo poenostavimo delo, saj nam praktično ni potrebno pisati izvorne kode za izrisovanje 3D prvin igre. OGRE 3D skrbi za pripravo scene v igri, kje in kako so obrnjeno 3D objekti, ali so skriti oziroma prikazani, kje je kamera in kam je obrnjena ter še veliko drugega. OGRE 3D smo uporabili tudi pri izdelavi večigralniške igre, kar je bolj podrobno opisano v poglavju 5. Dodatno imamo v knjižnici OGRE 3D tudi možnost pisanja izvorne kode C++ ali pisanja v skriptnem jeziku OGRE 3D za prikaz delcev in pisanja izvorne kode C++ ali pisanja v skriptnem jeziku OGRE variacije senčilnikov, ki imajo možnost barvne ali strukturne deformacije 3D prvin igre, izrisa njihovih senc in izrisa oziroma osvetlitve scene z lučmi [4].

Tipično pri nekaterih zvrsteh iger lahko uporabimo tudi fizikalni pogon. Med znanimi odprtokodnimi pogoni sta Open Dynamics Engine (ODE)[8] in fizikalni pogon Bullet[9]. Med priljubljenimi je tudi lastniška oziroma plačljiva NVIDIA Physx[10]. S fizikalnim pogonom lahko uporabljamo že obstoječe metode za premikanje objektov. Tako nam ni potrebno pisati svojih. Če na grobo povzamemo vsebino fizikalnega pogona, imamo v njem lahko več svetov, več prostorov (3D objekt je lahko na isti poziciji, ampak je v drugem svetu ali prostoru), uporaba težnosti, nastavitve nad skupinami fizikalnih objektov, trenje, odboj in še veliko več. Če želimo uporabljati fizikalni pogon pri izdelavi igre, je potrebno vsak 3D objekt tudi opisati z različnimi volumni, s pomočjo katerih fizikalni pogon zaznava trke z drugimi objekti v igri. Vsak takšen objekt postane tako dinamičen objekt. Dinamičnemu objektu lahko pripišemo še vrsto lastnosti, kot na primer upor, trenje, lepenje, odboj itd. Za izvoz iz oblikovne aplikacije je potreben vtičnik, ki podpira fizikalne objekte. Plačljivi fizikalni pogon NVIDIA Physx nam ponuja tudi orodje, v katerem

[6]http://www.crytek.com/cryengine
[7]http://www.ogre3d.org/
[8]http://www.ode.org/
[9]http://bulletphysics.org/
[10]http://www.nvidia.com

vidimo samo fizikalne objekte v igri. Težave neizkušenih programerjev so včasih neločevanje med 3D objekti igre, ki imajo sklepe okostja, in fizikalnimi objekti igre, kateri imajo dinamične sklepe. Skratka struktura mora biti taka, da so deli 3D objektov vgnezdeni v fizikalnih objektih tako, da se v igri potem premika in rotira fizikalne objekte [13].

Računalniška igra mora zaznavati tudi vnašanje znakov preko tipkovnice. Skupaj z OGRE 3D dobimo še eno knjižnico, ki nam omogoča prav to. Object Oriented Input System[11] je knjižnica, ki je bila napisana v programskem jeziku C++ in naj bi delovala na večini operacijskih sistemov.

Veliko programiranja pride na vrsto pri pisanju same logike igre, saj lahko vse druge dele igre ponovno uporabimo v drugih svojih igrah in je pisanje kode logike igre nekako edinstven del same igre. Tukaj pišemo vsa pravila, kaj se dogaja, kaj se bo zgodilo, kako in kam se kakšen objekt premika in podobno. Tukaj pride v poštev tudi pisanje umetne inteligence za računalniško vodene objekte.

V igrah brez večigralskega sistema bi lahko bilo zadnje poglavje pri izdelavi le-teh vključitev zvoka in glasbe v igro. Zvok je potrebno predhodno posneti, to opravi oblikovalec zvoka, ki tudi združi glasbo, ki jo je ustvaril skladatelj. Oblikovalci zvoka uporabljajo kakovosten mikrofon, s katerim posnamejo različne zvoke v naravi ali pa jih ustvarjajo s pomočjo kakšnega predmeta, nato pa s programi kot je Audacity[12] naredijo različne učinke, urejajo glasnost zvoka, višino zvoka ipd. Ko že imamo datoteke z zvočnimi zapisi lahko pri razvijanju igre uporabimo zvočno knjižnico OpenAL[13], razen, če nismo razvijali igre s prej omenjenimi pogoni(UDK, CryEngine, UNITY itd.), ki imajo zvočni sistem že napisan za uporabo. Tu se konča glavni del izdelave igre in začne faza odpravljanja napak in vzdrževanja.

V primeru želje po večjem zaslužku in storitvi, ki bi omogočala igranje igre s prijatelji, je tukaj še možnost izdelave večigralskega sistema igre. Pri igrah, ki so grajene okrog večigralskega sistema, je potrebno takšno kom-

[11] http://sourceforge.net/projects/wgois/
[12] http://audacity.sourceforge.net/
[13] http://connect.creativelabs.com/openal/

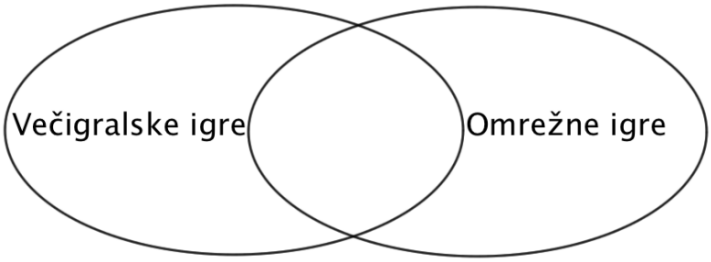
ponento načrtovati že od vsega začetka. Strukturo in realizacijo mrežnega večigralskega dela igre bomo obravnavali v sledečih poglavjih. Še prej si bomo pogledali zgodovino večigralnosti v igrah.

Poglavje 3

Zgodovina večigralnosti v igrah

Večigralstvo lahko razdelimo v dve skupini: omrežno večigraltvo na različnih računalnikih in večigralske igre na enem računalniku (torej brez omrežja). Danes so igre z večigralstvom pogosto omrežne igre, saj več igralcev ne igra več na isti napravi kot je bilo včasih pogosto. Veliko večigralskih iger, še posebej starejših, ni bilo omrežnih. Značilno je, da so igralci v takšnih večigralnih igrah izmenično igrali na istem računalniku. Na primer: en igralec je šel v boj proti tujim ladjam, medtem ko je drugi igralec gledal. Ko je prvi igralec uničil nasprotnika ali ko je končal stopnjo, je bil na vrsti drugi igralec. Rezultati vsakega igralca so bili ločeni. Za sočasno igranje sta igralca igrala skupaj ali eden proti drugemu tako, da sta igralca videla svojo podobo skupaj na istem zaslonu ali pa se je zaslon razdelil na dva ločena dela. Na primer: v večigralskih športnih igrah vsak igralec upravlja člana svoje ekipe. Posledično področje večigralskih iger vključuje igre, ki niso omrežne.

Obstajajo tudi nekatere omrežne igre, ki niso večigralske. Igra uporablja mrežo za povezovanje računalnika na oddaljen strežnik, ki nadzira različne vidike igranja. Igra sama po sebi pa je lahko v celoti enoigralska, kjer ni neposredne interakcije z drugimi igralci oziroma njihovimi liki. V zadnjih nekaj letih takšne igre uporabljajo mrežno povezavo zaradi varnostnih razlogov, kot je na primer potrditev serijske številke igre. Pri sodobnih računalniških sistemih lahko igralci vodijo igro lokalno na svojem računalniku in se povežejo s



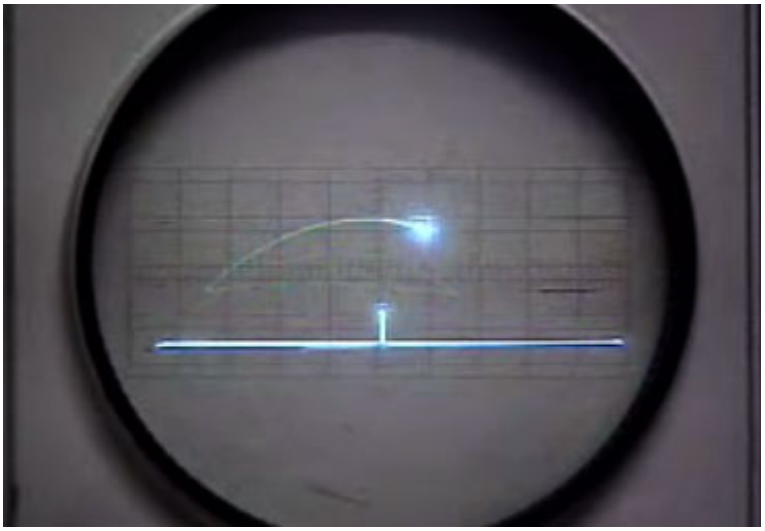
Slika 3.1: Prekrivanje večigralskih in omrežnih iger.

strežnikom zaradi interakcije z umetno inteligenco pod nadzorom strežnika. Večigralske in omrežne igre se prekrivajo kot je prikazano na sliki 3.1.

Prva ali ena izmed prvih dokumentiranih večigralskih iger je bila "tenis za dva"(slika 3.2). William A. Hinginbotham je leta 1958 v Brookhaven nacionalnem laboratoriju z uporabo osciloskopa simuliral navidezne partije tenisa. Igra omogoča dvema igralcema, da tekmujeta med seboj v poskušanju spravljanja žoge mimo nasprotnega loparja (ploščka).

Igra tenis za dva je uporabljala žično vezje in ne računalnika. Čast prvi pravi računalniški igri pa gre igri Spacewar (slika 3.3), ki je bila zasnovana leta 1961 in je delovala na takrat novem računalniku PDP-1 na MIT. V igri Spacewar sta se dva igralca spopadla med seboj z raketoplanom in poskušala uničiti s torpedi. Spacewar ni imel zvočnih učinkov ali učinkov delcev, vendar pa ponazarja, kako lahko takšna igra zasvoji tudi brez sodobne grafike. Igra kaže tudi, da umetna inteligenca ni bila potrebna, saj prava inteligenca v obliki človeškega nasprotnika, lahko izboljša izkušnjo igranja igre v tekmovalnem načinu in načinu sodelovanja [1].

Ob koncu leta 1993 je podjetje id Software izdelalo računalniško prvoosebno streljaško (angl. kratica FPS) igro Doom. Na sliki 3.4 je prikazana igra Doom v večigralskem načinu, kjer je v igri še en povezan igralec. Čeprav so bile izdelane druge prvoosebne streljaške igre pred njim, je Doom postavil svojo zvrst na višjo raven. Za igralce večigralskih iger omogoča Doom igranje štirih igralcev, ki med seboj sodelujejo s pomočjo protokola IPX (zgodnji



Slika 3.2: Prva dokumentirana večigralska igra Tennis for two [23].

Slika 3.3: Prva večigralska računalniška igra Spacewar [22].

Slika 3.4: Igralec povezan v večigralsko igro v prvoosebni streljaški igri Doom.

medomrežni protokol podjetja Novell) po lokalnem omrežju.

Poleg prvoosebnih streljaških iger je priljubljenost zvrsti množičnih igralskih spletnih iger igranja vlog (angl. kratica MMORPG) v letu 1995 začela rasti z igro Ultima Online, ki je večigralska omrežna igra, ki temelji na priljubljeni enoigralski igri serije Ultima podjetja Origin. Ultima Online je sprva omogočala sočasno igranje do 50 igralcev, kar je malo po današnjih standardih, vendar veliko za razmere tedanjega časa. Ultima Online je prva uspešna igra zvrsti MMORPG . Na sliki 3.5 je prikazana igra Ultima Online z nekaj sto igralci na enem strežniku v enem prostoru.

Omrežno igranje se razlikuje glede na zvrst igre. Zvrsti omrežnih večigralskih iger, ki jih pogosto srečamo so:

- potezne igre (angl. Turn based games),
- realnočasovne strategije (angl. Real-Time Strategy; RTS),
- prvoosebne streljačine (angl. First Person Shooter; FPS),
- igra z množičnim večigralskim spletnim igranjem vlog (angl. Massively multiplayer online role playing game; MMORGP),

Slika 3.5: Nekaj sto igralcev, povezanih v večigralsko igro Ultima online.

- dirkaške,
- igre s športno tematiko itd.

Več realizacij mrežnega igranja pri posamezni zvrsti je opisanih v naslednjem poglavju. V prihodnosti bomo lahko poslali vedno večje število podatkov po mreži. Nekatere priljubljene množične spletne igre, kot so World of Warcraft in Guild Wars 2, lahko celo dinamično organizirajo dogodke oziroma potek le-teh znotraj igre. V igri Diablo 3 podjetja Blizzard se bo objekte v igri, kot na primer meče, ščite in podobno, lahko kupovalo s pravim denarjem. Pri tem pa si Blizzard obeta velike prihodke, saj naj bi si za vsako transakcijo odtegnili delež denarja. Ta dodatna značilnost v večigralskih igrah bi lahko spremenila vse bodoče generacije igranja in služenja denarja s pomočjo računalniških iger [12].

Poglavje 4

Večigralstvo preko računalniškega omrežja

Po pregledu zgodovine večigralstva si bomo v tem poglavju pogledali podrobnejšo sestavo večigralniškega omrežnega sistema in njegovo varnost.

4.1 Kaj je omrežje?

Osnovni koncept mreže je komunikacija med več računalniki. Za to potrebujemo enega ali več odjemalcev in najmanj en strežnik ali pa vsaj dva enakovredna računalnika. Strežnik je lahko bodisi namenski gostiteljski stroj (angl. dedicated host machine) za vsakogar bodisi igralec, ki je hkrati strežnik, na kateremu teče igra. Ko sta strežnik in odjemalec povezana, lahko pričneta z izmenjavo podatkov, ki so potrebni za večigralsko omrežno igro [24].

Večigralska igra je igra, ki nudi možnost hkratnega igranja več igralcev, bodisi izmenično na isti napravi bodisi istočasno na različnih napravah, povezanih s katerim od tipov omrežij. Večina sodobnih računalniških iger vključuje večigralski način kot možnost poleg enoigralskega, nekatere pa so napisane izključno kot večigralske. V večigralskem načinu drug človek nadomesti soigralca, ki ga nadzoruje umetna inteligenca. Odvisno od vrste igre lahko igralci sodelujejo pri doseganju cilja igre ali tekmujejo med seboj

za najboljši rezultat. Tako igranje pridobi na tekmovalnosti in družabnosti, človeški nasprotniki pa so lahko zaradi iznajdljivosti tudi večji izziv.

Na sedanji generaciji konzol in računalnikov imamo skoraj vedno možnost, da ustvarimo večigralsko omrežje. Večigralsko omrežno igranje je v bistvu postalo zelo priljubljeno.

4.2 Arhitektura omrežnega dela igre

Najpomembnejša odločitev pri oblikovanju omrežnega protokola za igre, ki je potrebna pred pisanjem omrežnega dela igre, je odločitev o arhitekturi mrežnega dela igre. Arhitekturna izbira lahko močno vpliva na kasnejše pomembne spremembe, sploh če smo se odločili za napačno arhitekturo, saj jo spremlja dolgotrajen proces predelave. Pomembne so tri točke: večnitna zasnova, arhitektura omrežja in varnost kode. Nobene od treh točk ni mogoče pripisati na že obstoječi igralni pogon brez kasnejših hudih težav in napak, katerih posledica je slaba kakovost za vse ostale komponente pogona. Tako je najbolje, da se igro izdeluje okoli teh treh točk, namesto da so to kasnejši pripisi že obstoječi igri [5].

Glavni izbiri pri arhitekturi sta arhitekturi vsak z vsakim (angl. peer-to-peer) in odjemalec-strežnik.

V sliki 4.1 so prikazane možne arhitekture večigralnih in omrežnih iger:

a) dva ali več igralcev na enem računalniku, običajno z razdeljeno sliko,

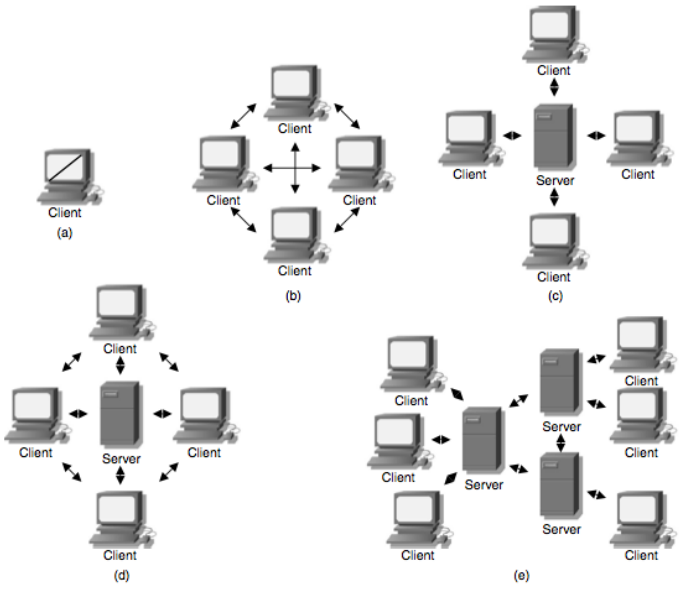
b) vsak z vsakim,

c) odjemalec-strežnik,

d) kombinacija med vsak z vsakim in odjemalec-strežnik,

e) omrežje strežnikov.

Na sliki 4.1 je več možnih arhitektur večigralskega sistema. Arhitektura a) ni omrežna, ampak je zgolj večigralska. V nadaljevanju si bomo podrobneje



Slika 4.1: Možne arhitekture večigralskega sistema [1].

pogledali mrežne arhitekture b) vsak z vsakim, c) odjemalec-strežnik in e) večstrežniška arhitektura. Arhitekturi d) in e) sta razširitvi arhitektur b) in c).

4.2.1 Vsak z vsakim

Arhitekturo vsak z vsakim (angl. Peer to peer) se je uporabljalo vrsto let, dandanes pa se jo le redko kdaj. Ta arhitektura odpade tam, kjer se omrežna komunikacija pogosto uporablja. Deluje tako, da vsak računalnik v omrežju (angl. peer)[1] pošlje podatke vsem svoje drugim vrstnikom, ki so povezani med seboj. V tem primeru računa vsak računalnik svoje podatke in jih pošilja ostalim. Tako lahko nastanejo velike matematične napake pri fiziki, ki jih je nemogoče odpraviti. Arhitektura deluje brez napak v primeru potezne igre, medtem ko pri novejših igrah, ki zahtevajo vedno več poslanih podatkov

[1]Računalnik v omrežju za izmenjavo podatkov, ki je na istem nivoju kot drugi računalniki.

preko omrežja, pogosto nastajajo težave. Pri arhitekturi vsak z vsakim je neljuba naloga enakovrednih računalnikov v omrežju tudi odkrivaje goljufov. Mehanizmi igre so lahko uničeni zaradi takega goljufivega odjemalca. Na primer: če obstaja v igri funkcija, ki omogoča, da igralec brcne drugega igralca iz igre zaradi suma goljufije, potem lahko tudi goljuf izkorišča to funkcijo tako, da brcne iz igre igralca, ki ni goljuf. V tem primeru je potrebno uporabiti protokol varnega in anonimnega glasovanja, tako več odjemalcev sprejme odločitev preden pride do prepovedi igranja igralca. Če povzamemo je arhitektura vsak z vsakim zelo požrešna kar se tiče omrežnega prometa, pa tudi nevarna [3].

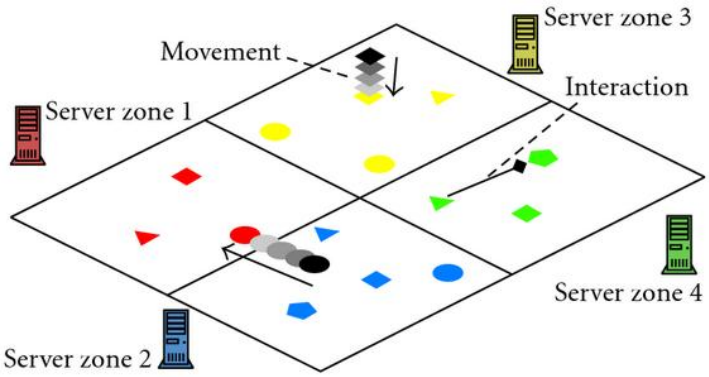
4.2.2 Odjemalec-strežnik

Po prihodu igre Doom vedno več omrežnih iger uporablja arhitekturo odjemalec-strežnik (angl. Client/server), kjer ima glavni strežnik ali poslušalec-strežnik (odjemalec in strežnik hkrati) glavno besedo pri stanju igre [1]. Od odjemalca dobi vhodne parametre vsakega igralca, izračuna stanje na podlagi vhodnih podatkov in rezultate pošlje nazaj vsem povezanim odjemalcem. Z varnostnega vidika je glavna prednost te arhitekture, da za strežnik vemo, da je vreden zaupanja. Strežnik ima tudi nalogo odkrivanja in kaznovanja goljufov [5].

Unreal Engine uporablja splošni odjemalec-strežnik, kjer strežnik vsebuje dokončno stanje igre, odjemalci pa delujejo s približkom oziroma pomanjkljivim poznavanjem sveta. Te informacije se sinhronizirajo v ustreznih časovnih presledkih [5].

4.2.3 Večstrežniška arhitektura

Razvoj masivnih večigralskih spletnih iger je precej bolj zapleten v primerjavi z razvojem priložnostne spletne igre za majhno število uporabnikov. Ker en strežnik ne more streči visokim številom uporabnikov, postane večstrežniška arhitektura potrebna. Na sliki 4.1 je tak sistem strežnikov prikazan pod



Slika 4.2: Štirje strežniki, ki strežejo vsak svoj del sveta [2].

oznako e). Povezava več strežnikov vodi do številnih problemov z usklajevanjem podatkov med njimi. Tudi razhroščevanje postane oteženo. Potrebni so tudi posegi zaradi preobremenitev posameznih strežnikov [2].

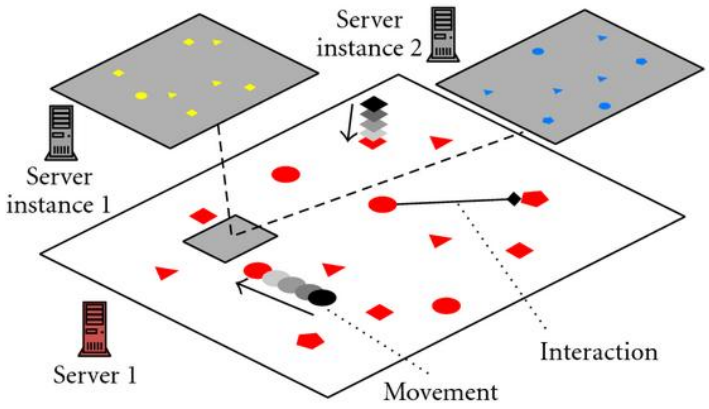
Poleg izmenjave podatkov o igralcih strežniki tipično strežejo še njihove pozicije. V masivnih večigralskih spletnih igrah so strežniki običajno specializirani glede na tri vrste delitve oz. nedelitve sveta:

- območna distribucija sveta (angl. zoning),
- instanciranje delov sveta (angl. instancing) (slika 4.3),
- podatkovno kopiranje sveta (angl. replication) (slika 4.4).

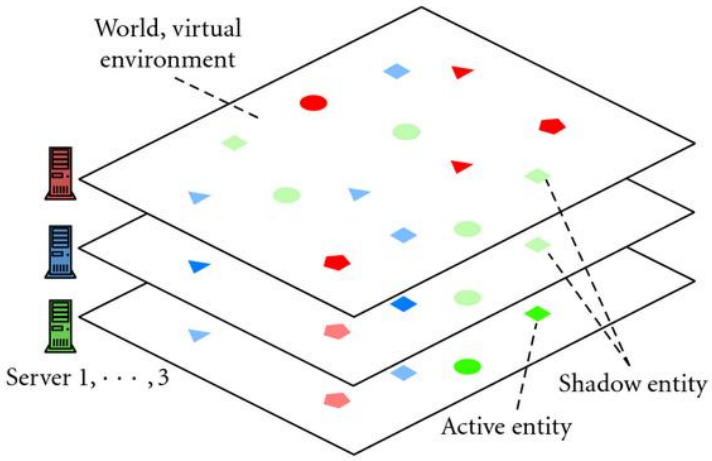
Pri območni distribuciji sveta si strežniki običajno strežejo primerno razdeljene dele sveta, kot je prikazano na sliki 4.2.

Pri instanciranju delov sveta (slika 4.3) instančni strežniki strežejo vsak svoj primerek (angl. instance) istega območja igre, vendar z drugimi igralci v njej, ki se med seboj ne vidijo.

Pri podatkovnem kopiranju sveta (slika 4.4) imajo vsi strežniki enako funkcionalnost in si delijo obremenitev. Strežniki si porazdelijo entitete označenega območja. Posamezen strežnik prekopira obdelane podatke na ostale strežnike.



Slika 4.3: En strežnik, ki streže območje in dva instančna strežnika [2].



Slika 4.4: Podatkovno kopiranje sveta s tremi strežniki [2].

4.3 Protokol

Naslednja velika odločitev pri načrtovanju omrežnega dela igre je izbira mrežnega protokola. Izbiramo med uporabo počasnejšega TCP za zagotovljeno dobavo ali hitrejšega UDP. Potrebno je zagotoviti, da izbrane varnostne metode delujejo z izbranim protokolom. Na primer: če so paketi šifrirani na način, ki zahteva, da se dostavi vse pakete, potem nam bo UDP povzročal glavobole. Načeloma bi v računalniški igri potrebovali protokol TCP za komunikacijo med dvema igralcema in UDP za prestavljanje pozicij objektov [6]. Razne omrežne knjižnice, kot je ENet (podrobneje opisan v podpoglavju 5.2), nam ponujajo dodatne izboljšave na protokolu UDP in popolnoma opuščajo protokol TCP.

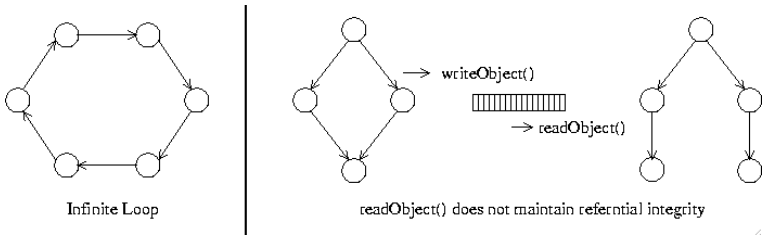
4.4 Serializacija

Pomembnost serializacije v večigralskih igrah preko mreže

Cilj izdelave računalniške igre, ki podpira igranje preko mreže, je čim manjša obremenitev omrežja. Tako je serializacija podatkov zelo pomembna pri pošiljanju paketkov preko mreže, saj nam zagotovi enako stanje spremenljivk, ki ga je imel *odjemalec1* na računalniku pri *odjemalec2*. Po potrebi se serializirane podatke tudi stisne, če ugotovimo, da ima naša igra ozko grlo pri pošiljanju in prejemanju paketkov.

Delovanje serializacije

V računalništvu (v kontekstu shranjevanja podatkov in prenosa) je serijalizacija proces pretvarjanja podatkovnih struktur ali stanja objektov v zapis, ki ga je mogoče shraniti (na primer v datoteko ali pa prenesti preko omrežne povezave) ter kasneje ponovno uporabiti na istem ali drugem računalniku. Posledično lahko potem, ko niz bitov spet preberemo, ustvarimo vsebinsko popolnoma enako kopijo izvirnega objekta. Za veliko kompleksnih objektov,



Slika 4.5: Možne težave pri serializairanju.

kot so npr. tisti, ki uporabljajo veliko kazalcev, ki se sklicujejo na druge objekte, ta proces ni enostaven. Pri serializaciji takega objekta je potrebno najprej serializirati podatke, na katere se sklicujemo. Ta proces serializacije objekta se imenuje tudi izpraznjevanje objekta(angl. deflating). Nasprotna operacija, rekonstrukcija podatkovne strukture, se imenuje tudi deserializacija (angl. deserialization) [20].

Slika 4.5 prikazuje možne težave pri serializiranju podatkov, pri katerem imamo lahko ciklične sklicne povezave na objekte, pri katerih je treba poskrbeti, da so podatki po serializaciji takšni, kot so bili pred njo.

Kljub temu lahko serializacija izpostavi podrobnosti implementacije. Da bi podjetja odvrnila izdelavo združljivih proizvodov pred tekmovalnimi podjetji, založniki lastniške programske opreme pogosto hranijo podrobnosti o svojih programskih serijskih formatih kot poslovno skrivnost. Nekateri namerno prikrijejo ali celo šifrirajo serializirane podatke [20].

Formati serializacije

Poznamo mnogo formatov serializacije, pogostejši so denimo:

- XML,
- JSON,
- YAML,

- seznam lastnosti (angl. Property list - na operacijskih sistemih Mac OS X) itd.

V našem primeru mislimo predvsem na bitno serializacijo, torej na zaporedja enk in ničel, ki se kot najmanjši možni paketki prenesejo preko omrežja. Enke in ničle je tako tudi lažje stisniti.

4.5 Varnost in napadi

Omrežne igre morajo preprečiti goljufom uporabo orodij za goljufanje in vdiranje v igro, saj s tem uničijo zabavo in povzročajo škodo drugim igralcem. Goljufamo lahko s pomočjo programske opreme, dodatkov ali datotečnih sprememb. Goljufi tako pogosto postanejo premočni za druge udeležence, povzročajo, da igra postane neodzivna ali pa skušajo oropati račune drugih igralcev. Ker je mogoče razstaviti vso kodo, ki teče na odjemalcu, se je naučiti in spremeniti, je potrebno sprejeti varnostne odločitve pri izdelavi večigralskega sistema. Vsak tak sistem mora biti zasnovan z dvema ciljema [5]:

- omogočiti enostavno odkrivanje goljufov in
- preprečiti goljufanje v igri.

Goljufe bi lahko odkrili in preprečili s primernimi pastmi v kodi. Na primer: ko bi goljuf spremenil določen del kode, kar bi posledično spremenilo določene podatke, bi se samodejno sprožila metoda pošiljanja sporočila glavnemu strežniku, nakar bi sledili ukrepi. Take pasti bi lahko napisali v svoji kodi tako, da bi posebej naredili neko metodo (na primer metodo spreminjanja denarja v igri) in bi preverjali, če se vrednost spremenljivke denarja spremeni v napisani metodi. Če se vrednost spremeni kje drugje je torej igralec goljufal. Še lažje je ugotavljati goljufe, če imamo glavni strežnik, ki vsebuje vse podatke igralca. Tako bi glavni strežnik takoj vedel, ali nekdo goljufa, ker se vrednosti ne bi ujemale z igralčevimi na strežniku. Nekaj uporabnih trikov lahko najdemo na forumu "Gamedev Stack Exchange" [27].

Gradnja varnosti igre ali varnostnega omrežja je pogosto dolgotrajno in težko delo, saj morajo razvijalci zaščititi igro pred vsemi možnimi napadi, medtem ko je dovolj, da goljuf izkoristi le eno luknjo.

Stopnja razvitosti napadov v igrah je neposredno sorazmerna s priljubljenostjo in dolžino življenjske dobe igre. Zato je varnost bolj potrebna pri visokokakovostnih računalniških igrah z visokim proračunom (angl. Triple A oziroma AAA title games) kot pri igrah neodvisnih izdelovalcev (angl. indie games). Na primer: World of Warcraft uporablja protigoljufsko orodje Warden. Warden deluje tako, da vsakih 15 sekund pošilja podatke na Blizzardove strežnike, kjer potem Blizzardovo osebje preveri pristnost podatkov [5]. Podatki so običajno imena trenutno odprtih programov na uporabnikovem računalniku. Če osebje zazna znan program za goljufanje nemudoma sledijo ukrepi. Zaradi negativnih odzivov javnosti zaradi neupoštevanja zasebnosti je Blizzard program prenovil, tako da Warden sledi samo spremembe na pomnilniških lokacijah, kjer je nahaja Blizzardova igra [26]. Podobno delujeta tudi Valvova aplikacija proti goljufanju (angl. Valve Anti-Cheat (VAC)) in PunkBuster[2] [25]. Na splošno ni veliko javnih podatkov o programih za zagotavljanje varnosti, saj bi tako goljufom olajšali goljufanje.

Kot zanimivost je dobro ločiti:

- vdiranje(angl. hacking), ki v tem kontekstu pomeni spreminjanje igre, običajno z negativnim prizvokom (spreminjanje programa za nekaj zlonamernega) in

- goljufanje, ki v tem kontekstu pomeni dejanje izkoriščanja vdiranja v igro za nepošteno pridobitev prednosti.

Poznamo več vrst goljufij, najbolj priljubljena je razdiranje igre v kodo zbirnega jezika na odjemalčevem računalniku. Tako lahko popravimo nekaj vrednosti in igra lahko deluje drugače. Nekateri razvijalci iger postavljajo v igre pasti, da ko se pomembni podatki spremenijo, popolnoma zmedejo potek igre v upanju, da se potem goljufi odvadijo spreminjanja kode. Zelo pogosti

[2]http://www.evenbalance.com/

goljufanji na tak način sta uporaba "aimbotov", kjer igralec (goljuf) spremeni kodo do takšne mere, da ko npr. v streljaški igri strelja in zadene vse ostale igralce v igri; in "wallhack", kjer igralec (goljuf) brez težav hodi ali strelja skozi zidove. Med znanimi metodami je tudi zakasnjevanje prenosa podatkov. To pomeni, da bodo vsi odjemalci občutili zelo veliko zamudo prihajanja paketkov do svojega računalnika. Pri tem mora igralec biti strežnik oziroma poslušalec strežnik[3]. Druga možnost je, če igralec pridobi IP-naslove ostalih igralcev v igri in jim začne pošiljati večje količine neuporabnih podatkov, da tako postane igra ostalih igralcev zelo počasna.

Kot protiukrep je potrebno pisati izvorno kodo z uporabo polimorfizma. Primerna je tudi uporaba raznih šifriranj. Šifriranje lahko upočasni delovanje igre, vendar se lahko še vedno uporablja kot občasno odkrivanje goljufij [7]. Dobro je razviti dovolj pameten strežnik, ki zaznava goljufe po statistiki. Za statistiko igralcev je potrebno hraniti podatke o prijavljenih igralcih z uporabniškimi imeni, časi igranja, statistiko igre itd. Avtomatiziran sistem ali moderator pregleduje statistiko vseh igralcev in išče odstopanja. Da bi preprečili poškodbe igralskega okolja, je zelo pomembno redno načrtovati in izvajati varnostne kopije sistema [5]. Še eno samostojno orodje, ki ga lahko uporabimo proti goljufanju, je PunkBuster, ki se integrira v izvorno kodo igre [19].

Ko je goljuf ujet, mora igra imeti natančno opredeljen kazenski sistem. Večina protigoljufskih sistemov začasno ali za vedno prepove (angl. ban) goljufov račun. Pri igrah, kjer ni centralnega strežnika za uporabniške račune, smo žal nemočni in običajno odnehamo s trenutno igro. Za tovrstno varnost mora torej biti prisoten centralni strežnik [5].

4.6 Praktični primeri

V nadaljevanju si bomo pogledali nekaj primerov iger, ki so bile ustvarjene s pomočjo prej omenjenih večigralskih sistemov.

[3]Po Valvovi definiciji: igralec v igri, ki je hkrati tudi strežnik

Različne računalniške igre imajo različne potrebe pošiljanja podatkov pri večigralstvu, da bi se prenos podatkov preko mreže čimbolj zmanjšal. Spodaj bomo našteli nekaj priljubljenih zvrsti računalniških iger in opisali njihove potrebe pošiljanja podatkov preko omrežja v igranju.

MMORPG

Zadnjih nekaj let so med najbolj priljubljenimi zvrstmi igre z množičnim večigralskim spletnim igranjem vlog (angl. Massively Multiplayer Online Role Playing Game ali MMORPG). Najbolj znana igra te zvrsti je World of Warcraft.

Na posameznem strežniku v World of Warcraftu je približno pet tisoč igralcev. Arhitektura, ki jo World of Warcraft uporablja, je odjemalec-strežnik. V igri je potrebno posodabljati samo pozicijo igralcev, ki so v vidnem polju, in stanja določenih dogodkov. Interpolacija bitij v igri je preprosta (linearna), saj se vsaka premika z neko že v naprej določeno hitrostjo. Na sliki 4.6 je prikazano prehajanje igralčevega lika v naslednjo stopnjo. Tudi podatek o dogodku se pošlje na strežnik, ki ga nato slednji razpošlje vsem igralcem.

Potezne igre in realnočasovne strategije

Potezne igre imajo najmanj podatkovnega prometa po mreži. Vse, kar je potrebno storiti v takšnih zvrsteh je, da ob določenih dogodkih, na primer klikih, posodobijo stanje igre. Zaradi majhnega prenosa podatkov lahko v takšnih igrah uporabimo kakršnokoli večigralsko arhitekturo. Pogosto je uporabljena kar najpreprostejša "vsak z vsakim".

Med priljubljenimi igrami te zvrsti sta seriji Heroes of Might and Magic (4.7) in Civilization (4.8) ter družabne igre s kartami ali brez. Med zelo priljubljenimi zvrstmi so tudi realnočasovne strategije, ki so na začetku običajno delovale na arhitekturi "vsak z vsakim", dandanes pa vedno več takšnih iger uporablja odjemalec-strežnik. Ena takšnih iger je Starcraft. Na sliki 4.9 je

Slika 4.6: V igri World of Warcraft se pošiljajo podatki igralca, kot so njegova stopnja, stanje zdravja, orientacija in njegova pozicija, ki je nato linearno interpolirana, ter napad, ki ga je sprožil.

prikazano posodabljanje pozicij enot v tej realnočasovni strategiji.

Realnočasovne strateške igre delujejo v večigralskem načinu enako kot potezne igre z dodatkom sprotnega posodabljanja pozicij vsake vojaške enote. V poteznih igrah posodabljanje pozicij ni potrebno, saj se v njih zgodi le animacija (angl. animation), ki je vedno enaka.

Prvoosebne streljačine in dirkaške igre

Pri prvoosebnih streljačinah in dirkaških igrah je pomembno zelo hitro posodabljanje podatkov, saj se stvari dogajajo zelo hitro. Ena takšnih iger je Call of duty (4.10). Dober primer hitrega posodabljanja podatkov je tudi dirkačina, pri kateri avtomobili vozijo zelo hitro in jih je potrebno prestaviti vsako sličico. Pri takšnih igrah želimo doseči, da posodabljamo čim manjšo količino podatkov v določenem času, saj je tako interpolacija lažja in gladkejša. Pri dirkaških igrah se dodatno avtomobili ne vozijo s konstantno

Slika 4.7: Ko je igralec na potezi v igri Heroes of might and magic 3, se po vsakem premiku v potezi pošiljajo podatki o pozicijah enot in njihovih ukazih.

Slika 4.8: Ko je igralec na potezi v igri Civilization 5, se po vsakem premiku v potezi pošiljajo podatki o prestavljenih enotah in navodilih za gradnjo.

Slika 4.9: V igri Starcraft se skozi celo igro v večigralskem načinu neprestano pošiljajo podatki o prestavljenih enotah in navodilih za gradnjo.

Slika 4.10: V igri Call of duty 4 se skozi celo igro v večigralskem načinu pošiljajo podatki o poziciji in orientaciji igralcev skupaj s podatki o akcijah, kot je streljanje.

Slika 4.11: Posodabljanje pozicije avtomobila se izvaja ves potek večigralske igre v dirkaški igri Need for speed shift.

hitrostjo, zato je zelo pomembna tudi napoved vnosa (da se avtomobil ob pritisku gumba ne obrne šele po določeni zakasnitvi) in kompenzacija zakasnitve (da se ob trku z drugim vozilom vozili pravilno odbijeta). Primer dirkaške igre je Need for speed (4.11). Z veliko pogostostjo posodabljanja podatkov preko spleta je postal problem ozkega grla. Rešitvi te težave sta v zmanjšanju podatkov, ki jih mora vsak igralec poslati, ter zmanjšanju števila igralcev na strežniku. Tako redkokdaj vidimo več kot 32 igralcev prvoosebnih streljaških ali pa dirkaških iger. Pri takšnih igrah uporabljamo arhitekturo odjemalec-strežnik. Po dolgih letih prevlade arhitekture vsak z vsakim je bila prva igra, ki je uporabljala arhitekturo odjemalec-strežnik, prav prvoosebna streljačina Doom [1].

4.7 Posebnosti pri realizaciji omrežnih iger

4.7.1 Iskanje lokalnih strežnikov

Po realizaciji jedra večigralskega sistema je potrebna funkcionalnost iskanja strežnikov v lokalnem omrežju.

To izvedemo tako, da vsak strežnik veže podatkovni vtič UDP na točno določena vrata, samo za preverjanje odziva (angl. pinging). Promet igralnih paketkov naj se odvija na drugih vratih, ki pa se razlikujejo od strežniških. Informacijski vtič mora imeti ista vrata pri vseh strežnikih.

Za sejo v lokalnem omrežju odjemalec razpršeno oddaja sporočilo na IP-naslov 255.255.255.255, kjer se to sporočilo pošlje vsem aktivnim strežnikom igre, ki v lokalnem omrežju poslušajo na določenih vratih. Strežnik potem pošlje odgovor. Odgovor vsebuje med drugim IP-naslov strežnika, na katerega se mora odjemalec naslednjič, če se želi povezati nanj, odzvati, ter na katerih vratih se pošiljajo podatki o stanju igre.

Priporočljivo je tudi, da se to pošiljanje sporočila dogaja vsakih nekaj sekund, tako da odjemalec ve, kateri strežnik je aktiven in pri tem ne ovira omrežnega prometa zaradi zelo nizke frekvence preverjanja odziva. Tako bo tudi v primeru izgube sporočila zaradi protokola UDP v nekaj sekundah prispelo novo.

ENet nudi možnost povezave na IP naslov 255.255.255.255, s pomočjo katere se bo odjemalec povezal na prvi strežnik, ki mu bo odgovoril. To je primerno takrat, ko smo prepričani, da je samo en strežnik igre v lokalnem omrežju, ampak mora imeti igra to opcijo tudi implementirano.

Razvijalci knjižnice ENet odsvetujejo uporabo svoje knjižnice za ta namen in svetujejo uporabo običajnega protokola UDP [14].

4.7.2 Glavni strežnik

Po realizaciji jedra večigralskega sistema in funkciji iskanja primerkov igre v lokalnem omrežju nam preostane še realizacija povezljivosti z glavnim

strežnikom. V večigralskih igrah obstaja običajno glavni strežnik, na katerem so vsi podatki o strežnikih (če obstajajo), igralcih, ki igrajo igro, primerkih iger, ki so na voljo ipd. Skratka vsi podatki so na enem mestu.

Razširljivost je odločilnega pomena na današnjem trgu omrežnih iger. Tako se je v večigralskih igrah pojavila centralizirana storitev za gostovanje iger. Tudi če en primerek igre na primer podpira le majhno število igralcev, ima igra sposobnost za zagon dodatnih primerkov igre na enem strežniku. Storitev ima lahko velike stroške obratovanja. Poleg začetnega nakupa strojne in programske opreme ter licenčnine vsak fizični strežnik zavzame tudi veliko prostora, energije, potrebno je tudi hlajenje.

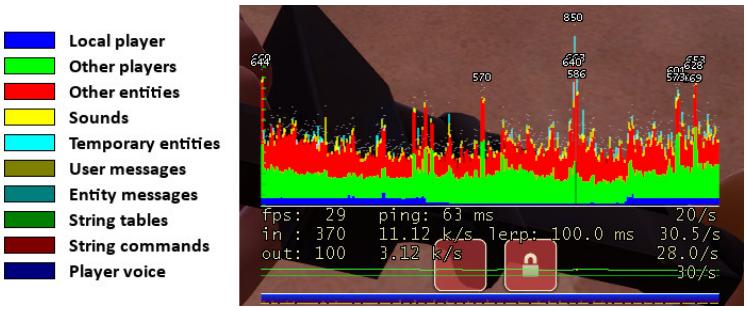
Upravlanje z več odjemalci in problem C10K

Igralni strežnik mora znati upravljati z več povezavami. Preprost strežnik mora iterirati preko vseh povezav odjemalcev in izvajati vhodno-izhodne operacije brez blokiranja za vsakega odjemalca posebej. Vendar pa to vključuje veliko izpraševanja, še posebej, če je večina povezav nedejavnih, kot se pogosto zgodi pri branju podatkov odjemalcev. Ker se število povezav veča, se tako veča tudi dolžina iteracije, kar zmanjšuje odzivnost strežnika za vsakega odjemalca.

Problem C10K[4] je omejitev, ki jo ima trenutno večina omrežnih strežnikov. Problem opisuje omejitve omrežnega strežnika, ki lahko upravlja z največ 10.000 sočasnimi povezavami. Ta omejitev je posledica kombinacije omejitev operacijskega sistema in omrežnih omejitev strežniške programske opreme. Čeprav obstaja nekaj specializiranih omrežnih strežnikov, ki zmorejo več kot 10.000 odjemalcev, jih večina omrežnih strežnikov trenutno obravnava največ 10.000 hkrati [9].

Cilj vsake igre je imeti čim več igralcev v istem primerku igre. Žal to ni mogoče zaradi prej omenjenih problemov. Rešitev je prehod na več porazdeljenih strežniških sistemov, ki sodelujejo med sabo (skupek fizičnih strežnikov, ki sodelujejo in si delijo odjemalce med seboj). Sistem je za-

[4] "deset tisoč odjemalcev"



Slika 4.12: Količina mrežnega prometa in delež posamezne vrste paketkov.

snovan tako, da se odjemalec poveže na strežnik v verigi strežnikov, ki mu je najbližji. Če ima strežnik preveč odjemalcev, ga ta preusmeri na drugega najbližjega. Glavne ovire strojne opreme so preozka pasovna širina za različne dele strežnika in neučinkovito ravnanje operacijskega sistema z velikim številom odjemalcev.

4.8 Problem zakasnitve paketkov

Z uporabo za zdaj najboljše arhitekture odjemalec-strežnik iz poglavja 4 pride do nekaterih težav, ki jih bomo opisali v tem poglavju. Žal se podatki po mreži ne pošiljajo s hitrostjo, s katero operira računalnik. Mreža poti, po katerih potujejo paketki, je tudi ozko grlo, saj prav tako ni mogoče, da bi preko žične povezave po Ethernet kablu ali po zraku z brezžično lokalno povezavo potovalo zadostno število podatkovnih paketkov. Tako se podatki pošiljajo na določen časovni interval, npr. desetkrat, dvajsetkrat ali tridesetkrat v sekundi. Na takšen način razbremenimo mrežno linijo, a problem zakasnitve paketkov še poslabšamo, ker nekaterih podatkov tako ne pošljemo. Tako težavo z zakasnitvijo paketkov rešujemo s tremi pomembnimi metodami, opisanimi v naslednjem poglavju.

Slika 4.12 prikazuje mrežni graf, prenosi kakšnih paketkov in koliko paketkov določene vrste se pošilja v mrežnih igrah.

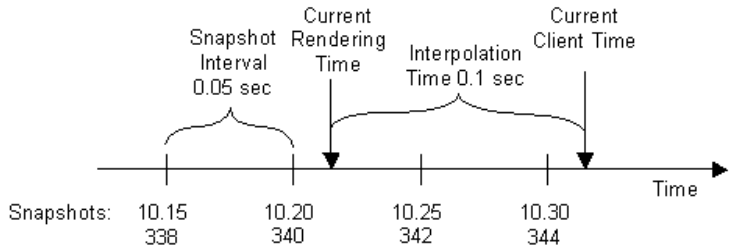
Napoved vnosa

Predpostavljajmo, da igralec igra prvoosebno streljačino, ima mrežno zakasnitev 150 milisekund in se začne premikati naprej. Informacija, da je igralec pritisnil tipko "naprej", se shrani v uporabniški ukaz, ki je poslan strežniku. Na strežniku se ukaz igralca obdela in ugotovi, da je to ukaz, ki služi za premik igralca. Strežnik izračuna, da se je igralec premaknil naprej v svetu. To novo stanje igralca se prenese tudi na vse odjemalce. Zaradi hitrosti podatkov, ki potujejo po mreži, bi torej posamezni igralec videl svojo spremembo gibanja s prej omenjeno 150 milisekundno zamudo. Vizualna zamuda velja za vsa igralčeva dejanja, kot so gibanje, streljanje z orožjem itd. in postane slabša z višjo mrežno zakasnitvijo.

Zamik med igralčevim vhodnim podatkom (pritisnjeno tipko) in ustrezno vizualno povratno informacijo ustvarja čuden, nenaraven, zamaknjen pomik, zaradi česar se je težko natančneje premakniti ali ciljati. Odjemalčeva napoved vnosa je način za odstranitev te zamude, po katerem naj bi se igralčevo dejanje občutilo kot bolj takojšnje. Namesto čakanja na podatke posodobitve svojega položaja, ki jih je strežnik poslal, odjemalec le napoveduje premikanje ob pritisku tipk. Odjemalec deluje s popolnoma enako kodo in pravili kot strežnik, zato ne obstaja možnost za presenetljivo velike razlike. Po končani napovedi se lokalni igralec takoj premakne na novo mesto, medtem ko ga strežnik še vedno vidi na starem.

Po 150 milisekundah odjemalec prejme strežnikovo izračunano stanje z izračunano novo pozicijo, ki temelji na pred tem poslanih vhodnih podatkih odjemalca. Odjemalec primerja s strežnikom svoj položaj in predvideno mesto. Če se položaj razlikuje, je pri napovedi prišlo do napake. To kaže, da odjemalec, ko je prišlo do obdelave igralčevega ukaza, ni imel pravilne informacije o drugih objektih in okolju v igri. Nato je potrebno, da odjemalec popravi svoj položaj, saj ima strežnik končno besedo nad odjemalčevo napovedjo.

Predvidevanje vedenja nekega objekta deluje le, če deluje odjemalec z enakimi pravili kot strežnik. To običajno ne drži, saj ima strežnik več infor-



Slika 4.13: Potek interpolacije med različnimi posnetki [21].

macij o objektih v igri kot odjemalci. Odjemalci vidijo le majhen del sveta hkrati, pa dobijo dovolj informacij, da narišejo tistih nekaj glavnih objektov. Tako napoved deluje samo za svojo podobo igralca v igri. Pravilna napoved drugih igralcev ali interaktivnih objektov trenutno ni mogoča na odjemalcu, ker bi tako prišlo do velikih napak v igri [21].

Interpolacija

Na primer: odjemalec prejme približno 20 posnetkov stanja na sekundo. Če bi bili objekti v svetu samo izrisani na položajih, ki jih je strežnik odposlal v obliki paketkov, bi se premikajoče se objekte in njihove animacije videlo utripajoče. Izgubljanje paketkov bi povzročilo opazno večje napake. Trik za rešitev tega problema je interpolacija med dvema posnetkoma. Po prej omenjenih 20 posnetkih na sekundo pomeni, da pride do nove posodobitve približno vsakih 50 milisekund. Če je odjemalčev izrisovalni čas premaknjen nazaj za 50 milisekund, se lahko objekti vedno interpolirajo med zadnjim prejetim posnetkom in posnetkom pred tem. Prihod prihajajočih posnetkov ponazarja slika 4.13.

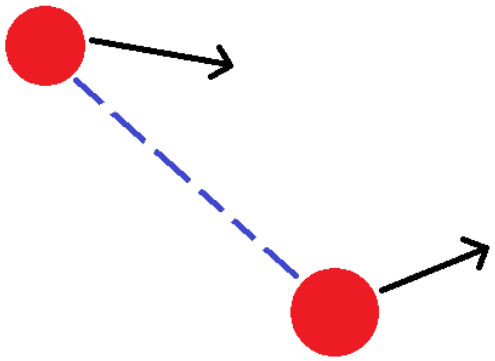
Slika 4.13 natančneje prikazuje zadnji prejeti posnetek odjemalca v ciklu 344 v času $t = 10,30$ sekund. V času $t = 10,22$ je na vrsti izris naslednje sličice in animacije objektov se interpolirajo z uporabo posnetkov 340 in 342. Interpolacija med tema dvema posnetkoma pa traja 100 milisekund. Ker imamo interpolacijsko zamudo 100 milisekund, bi interpolacija delovala tudi,

če bi posnetek 342 manjkal zaradi izgube paketov. Potem bi lahko interpolacija uporabila posnetke 340 in 344. Če več kot en posnetek ne prispe do odjemalca, interpolacija ni uspešna. V tem primeru lahko uporabimo linearno ekstrapolacijo, s katero izračunamo naslednjo pozicijo objektov s pomočjo zgodovine pozicij. Ekstrapolacija ekstrapolira le za 0,25 sekunde izgube paketov, saj bi v nasprotnem primeru napake postale prevelike. Interpolacija tako povzroči konstantno izrisovalno zamudo 100 milisekund, tudi če igramo na poslušalnem strežniku (strežnik in odjemalec na istem računalniku). Posledično to ne pomeni, da je potrebno predvidevati že 100 milisekund prej, kam bomo v igri streljali na nasprotnika, saj na strežnikovi strani kompenzacija zakasnitve ve za interpolacijsko napako in jo popravi [21].

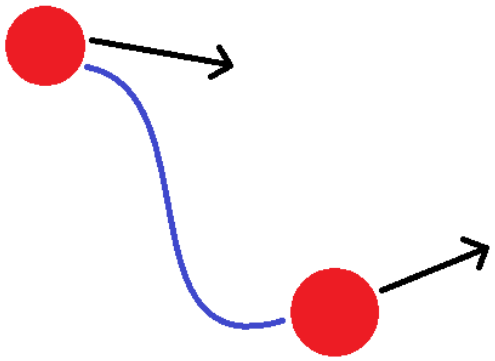
Interpolacijo se lahko izvede na več načinov. V odvisnosti od kompleksnosti igre lahko izbiramo med linearno interpolacijo in interpolacijo s kubičnimi zlepki (angl. spline). Lahko pa uporabimo kakšno svojo metodo. Z linearno interpolacijo interpoliramo igre, ki potrebujejo veliko procesorskega časa, da posledično ne proizvedejo zamud pri procesiranju paketkov. Pri interpolaciji s kubičnimi zlepki je pomemben podatek objektov tudi pospešek. Tako je interpolacija s kubičnimi zlepki zelo pomembna v dirkaških igrah. Podrobnosti interpolacije s kubičnimi zlepki so opisane v "Defeating Lag With Cubic Splines" [11]. Slika 4.14 prikazuje linearno interpolacijo s pomočjo hitrosti objekta, uporabljeno v igrah, kjer objekti nimajo pospeška in je njihova hitrost konstantna. Slika 4.15 prikazuje kubično interpolacijo s pomočjo hitrosti objekta in njegovega pospeška.

Kompenzacija zakasnitve

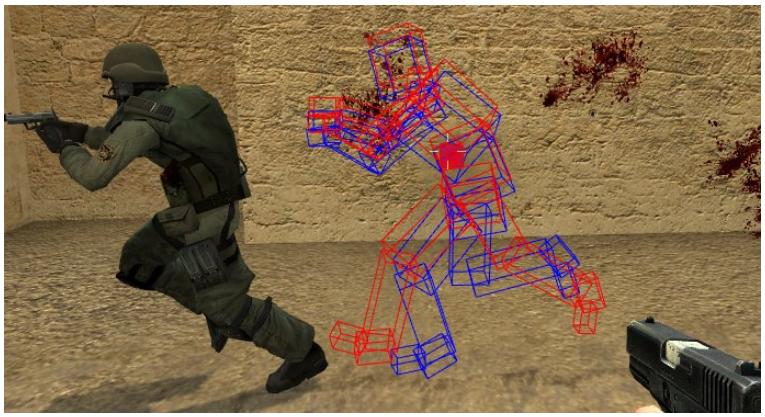
Recimo, da igralec strelja na cilj ob odjemalčevem času $t = 10,5$. Informacije streljanja so zapakirane v paketek, ki se pošlje strežniku. Medtem ko je paketek na poti preko omrežja, strežnik še vedno simulira potek dogajanja v svetu na podlagi prejšnjih podatkov. V tem času se cilj premakne na drugo mesto. Paketek z informacijo o streljanju prispe na strežnik v času $t = 10,6$ in strežnik ne zazna zadetka, čeprav je igralec ob času $t = 10,5$



Slika 4.14: Interpolacija premikanja objekta s pomočjo linearne interpolacije [11].



Slika 4.15: Interpolacija premikanja objekta s pomočjo kubične interpolacije [11].



Slika 4.16: Kompenzacija zakasnitve. Strežnik premakne vse podobe igralcev nazaj, kjer so bili v času izvedbe ukaza streljanja in preveri zaznavanje zadetka. [21].

ciljal natančno v tarčo. Ta napaka je popravljena z metodo kompenzacija zakasnitve. Sistem zakasnitvene kompenzacije hrani zgodovino vseh zadnjih igralčevih pozicij za eno sekundo. Če se izvede uporabnikov ukaz, strežnik oceni, kdaj je bil ukaz ustvarjen:

časovna oznaka izvedbe ukaza = trenutna časovna oznaka na strežniku - krožni čas paketka - odjemalčeva interpolacija

Strežnik nato premakne vse igralce (samo igralce oziroma podobe igralcev) nazaj, kjer so bili v času izvedbe ukaza streljanja. Ukaz se simulira še enkrat na strežniku in zadetek je pravilno zaznan. Po obdelavi ukaza strežnik prestavi igralce nazaj na zadnjo znano pozicijo.

Slika 4.16 je posneta na poslušalskem strežniku z 200 milisekund zamika, takoj ko je strežnik potrdil zadetek. Rdeča obroba tarče prikazuje ciljni položaj na odjemalcu pred 100 milisekundami. Odjemalec se je premikal v levo. Ko je na strežnik prišel uporabnikov ukaz, je strežnik obnovil položaj tarče (modra obroba tarče), ki temelji na predvidenem času izvedbe ukaza. Strežnik izračuna strel in potrdi zadetek (odjemalec vidi učinke krvi). Odjemalčeve in strežnikove obrobe tarče se ne ujemajo natančno zaradi manjših

napak natančnosti pri merjenju časa. Tudi majhna razlika v nekaj milisekundah lahko povzroči napako nekaj centimetrov za hitro premikajoči se objekt. Večigralsko zaznavanje zadetka ni do slikovne pike natančno in ima natančnostne omejitve, ki temeljijo na meri ciklov (angl. tickrate) in hitrosti premikajočih se objektov. Povečanje ciklov izboljša natančnost odkrivanja zadetka, zahteva pa tudi boljši procesor, več pomnilnika in večje pasovne širine za strežnik in odjemalce [21].

Če hočemo zagotoviti čim manj goljufanja, je potrebno, da rezultate zadetkov izračuna strežnik (več v poglavju 4). Če bi to izvajali odjemalci, bi lahko prišlo do nezaželene goljufije.

Mrežna zakasnitev in kompenzacija zakasnitve lahko ustvarijo paradokse, ki se zdijo nelogični v primerjavi z realnim svetom. Na primer: nas lahko zadene nasprotnik, ko se sami že skrijemo pred njim. Kar se zgodi je to, da strežnik preseli obrobe tarč nazaj v čas in izračuna zadetek, ko smo izpostavljeni napadalcu. Te težave nedoslednosti na splošno ni mogoče rešiti zaradi relativno počasnih hitrosti paketkov. V realnem svetu ne bomo opazili tega problema, ker luč (paketki) potujejo tako hitro, da ne zaznamo zamude [21].

Poglavje 5

Realizacija večigralske igre

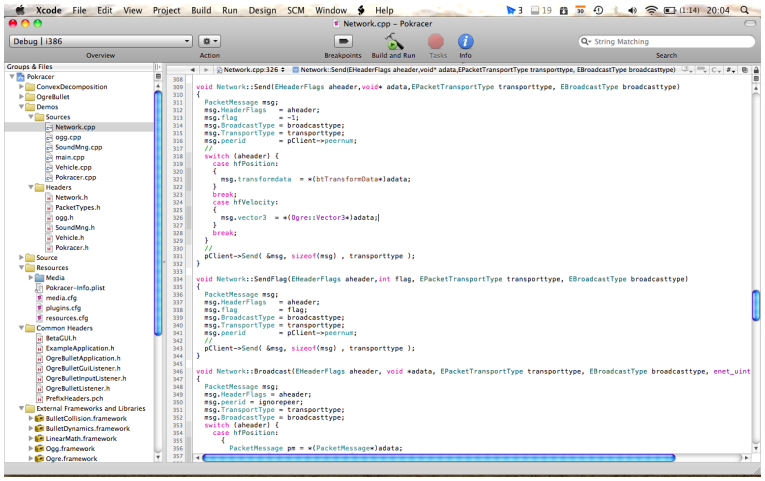
5.1 Realizacija igre

Prvotno sem izdelavo igre z večigralskim sistemom začel predvsem zaradi želje igranja igre s prijatelji in želje do programiranja zabavne igre. Igro oziroma začetni izdelek sem razvil na mojem prenosnem računalniku MacBook z nameščenim operacijskim sistemom Mac OS X 10.6 (Snow Leopard) v razvojnem okolju Xcode (prikazano na sliki 5.1).

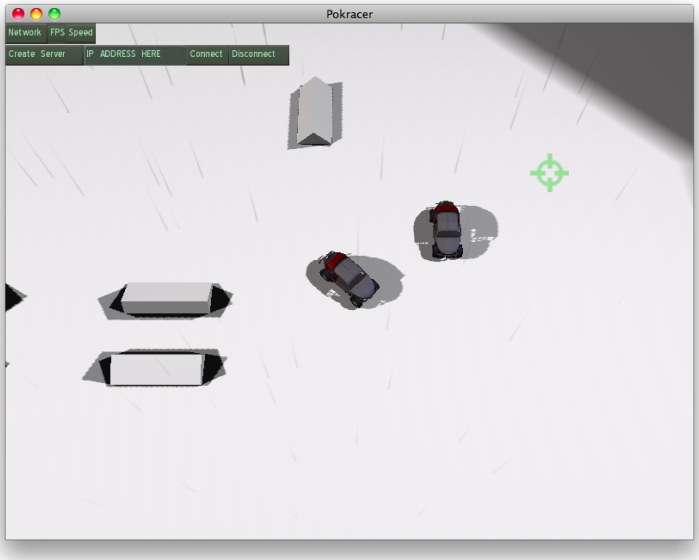
Za izrisovanje grafike sem uporabil knjižnico OGRE 3D z fizikalnim pogonom Bullet. Na spletni strani OGRE 3D ponujajo dodateko, ki združuje OGRE 3D in Bullet z imenom OGREBullet. Zaradi manjkajočega modelerja sem si 3D objekte prav tako izposodil iz OGREBulleta. Za grafični uporabniški vmesnik sem uporabil betaGUI[1], ki se prav tako dobi na spletni strani OGRE 3D. OGRE 3D ima zelo slabo podporo za operacijski sistem Mac OS X, saj si je potrebno projektno datoteko OGRE 3D za Mac OS X 10.6 narediti sam. Ostaja še ena slabost knjižnice OGRE 3D na Mac OS X – deluje samo v 32-bitnem načinu. Posledično je bilo potrebno zgraditi vse ostale knjižnice prav tako v 32-bitnem načinu. Zvočne učinke sem naredil z uporabo knjižnice OpenAL.

Na sliki 5.2 je prikazan izdelek z dvema primerkoma avtomobilov, pove-

[1] http://www.ogre3d.org/tikiwiki/BetaGUI



Slika 5.1: Razvojno okolje Xcode na operacijskem sistemu Mac OS X.



Slika 5.2: Prototip večigralske računalniške igre, kjer sta dva igralca (avtomobila).

zanih z večigralsko arhitekturo vsak z vsakim.

5.2 ENet

Po izdelavi preostalega dela igre smo začeli z izdelavo večigralskega sistema. Knjižnica ENet je jedro našega večigralskega sistema. Poznamo dva glavna protokola transportnega sloja:

- TCP in
- UDP,

ki žal nista preveč uporabna na področju računalniških iger (v podpoglavju 4.3).

Preden smo začeli z izdelavo, smo preverili, če že obstaja kakšna mrežna knjižnica, optimizirana za izdelavo iger. Iskali smo predvsem možnosti, ki bi jih bilo mogoče uporabiti v več računalniških okoljih. Seznam smo zmanjšali na 2 zadetka: Raknet[2] in ENet[3]. Raknet je zelo popularna knjižnica, saj jo uporablja tudi večnamenski pogon Unity. Odločili smo se za opcijo ENet predvsem zato, ker je brezplačen in zelo enostaven za uporabo, kar smo lahko predhodno preizkusili s pomočjo vadnice na njihovi spletni strani.

Knjižnica ENeta združuje dobre lastnosti obeh prej omenjenih protokolov transportnega sloja. Namen ENeta je zagotoviti relativno tanko, preprosto in trdno plast komunikacijskega omrežja nad UDP. Primarna funkcionalnost zagotavlja pogojno zanesljivo dostavo paketov. Knjižnica je primerna tako za izdelavo arhitekture vsak z vsakim kot za odjemalec-strežnik.

ENet izpušča nekatere višje funkcionalnosti mrežne ravni, kot so preverjanje pristnosti, čakanje v čakalni vrsti, odkrivanje strežnikov, šifriranje ali druge podobne naloge, ki so precej specifične za določene aplikacije, tako da ostane knjižnica prilagodljiva, prenosljiva in enostavno vgradljiva.

ENet je bil razvit posebej kot UDP mrežena plast za večigralnost prvoosebne streljačine Cube[4]. Cube je potreboval nizke zakasnitve podatkov, ki

[2]http://www.jenkinssoftware.com/
[3]http://enet.bespin.org/
[4]http://assault.cubers.net/

Slika 5.3: Igra Cube, ki uporablja omrežno knjižnico ENet [10].

bi se pošiljali zelo pogosto, tako da protokol TCP ni primerna izbira zaradi visoke zakasnitve in usmerjenosti toka. Protokol UDP pa potrebuje včasih potrebne funkcionalnosti protokola TCP, kot so zanesljivost, zaporedje, neomejena velikost paketka in upravljanje povezave. Tako niti protokol UDP sam po sebi ni bil primeren kot omrežni protokol igre. V takratnem času nastanka ENet ni obstajala nobena primerna, preprosta, dostopna omrežna knjižnica, ki bi zapolnila to vrzel [10].

Protokola UDP in TCP bi lahko skupaj bila uporabljena igri Cube in tako koristila oboje funkcij. Vendar pa je kombinacija protokolov še vedno daleč od želenega. TCP nima več tokov komunikacije brez zatekanja k odpiranju velikega števila vtičnic. UDP nima zaporednega prejemanja podatkov, upravljanja povezave, upravljanja pasovne širine virov in določa omejitve pri velikosti paketov.

ENet tako poskuša rešiti omenjene probleme in zagotavlja enoten večplastni protokol preko protokola UDP, da lahko razvijalec razpolaga z najboljšimi značilnostmi protokolov UDP in TCP, kot tudi z nekaj uporabnimi funkcionalnostmi, ki jih TCP in UDP ne zagotavljata. V nadaljevanju si bomo pogledali pomembne lastnosti knjižnice ENet [14].

Upravljanje s povezavo

ENet omogoča enostavno povezavo preko vmesnika, preko katerega komunicira s tujimi gostitelji. Aktivnost povezave aktivno spremlja s preverjanjem odziva tujih gostiteljev v pogostih intervalih, kot tudi spremlja stanje omrežja od lokalnega omrežja do tujih gostiteljev, pri tem pa vsebuje podatke o času potovanja in izgubi paketov.

Zaporedja

Namesto enotnega bajtnega toka podatkov, ki otežuje razmejitev paketov, ENet predstavlja več povezav pravilno zaporednih tokov paketov, s katerim se poenostavi prenos različnih vrst podatkov.

ENet zagotavlja zaporedje za vse paketke, tako da dodeli vsakemu poslanemu paketku zaporedno številko, ki se poveča po vsakemu poslanemu paketku. ENet zagotavlja, da paketek z večjo zaporedno številko ne bo prispel pred paketkom z nižjo zaporedno številko, kar zagotavlja, da so paketi poslani v pravem vrstnem redu.

Pri nezanesljivih paketkih bo ENet preprosto zavrgel paketek z nižjo zaporedno številko, če je paketek z večjo zaporedno številko že prispel. To omogoča pošiljanje paketkov takoj, ko so na voljo, in s tem zmanjšanje zakasnitev nezanesljivih paketov na absolutni minimum. Ko pa pri zanesljivih paketkih prispe paketek z višjo zaporedno številko in paketka z manjšo zaporedno številko še ni, bo ENet počakal z obdelavo paketkov dokler ne prispe manjkajoči paketek.

Kanali

Ker ENet zaustavlja dostavo zanesljivih paketkov, lahko zaradi zagotavljanja primernega zaporedja pride do zakasnitve pri dostavi ostalih paketkov, ki jim ni treba biti strogo zaporedni glede na ustavljen paketek.

Za boj proti tej zakasnitvi in zmanjševanju zaporednih omejitev za pakete ponuja ENet številne komunikacijske kanale za vsako povezavo. Vsak kanal

ima neodvisno zaporedje paketkov od drugega, tako stanje dostave paketka na enem kanalu ne bo ustavilo prihajanje drugih paketkov na drugih kanalih.

Zanesljivost

ENet zagotavlja dodatno zanesljivost dobave paketka, tako da zagotavlja, da tuji gostitelj potrjuje prejemanje vseh zanesljivih paketov. Če ne dobi potrditve, da je tuji gostitelj prejel paketek v določenem času, bo ENet poskušal ponovno poslati paketek znotraj primernega časovnega intervala. Vsaka časovna omejitev se stopnjuje in postane bolj prizanesljiva z vsakim neuspelim poskusom.

Razdrobljenost in ponovno sestavljanje

ENet bo pošiljal in dostavljal paketke ne glede na velikost. Veliki paketi so razdrobljeni v več manjših paketov primerne velikosti in se ponovno sestavijo na tujih gostiteljih. Postopek je popolnoma transparenten razvijalcu.

Združevanje

ENet združuje vse protokolne ukaze, vključno s potrjevanjem in paketnimi prenosi, v večje protokolne pakete, da se zagotovi pravilno uporabo povezave in omejuje možnost izgube paketov, kar bi lahko vodilo v še večjo zakasnitev pri dostavi.

Prilagodljivost

ENet zagotavlja statični mehanizem dodeljevanja pasovne širine za zagotavljanje skupnega obsega poslanih in prejetih paketov, da ne presegajo zmogljivosti gostitelja. Poleg tega ENet nudi dinamično dušenje, ki se odziva na odstopanja od običajnih omrežnih povezav za odpravo različnih vrst prezasedenosti omrežja z dodatnim krčenjem obsega poslanih paketov.

Prenosljivost

ENet deluje na operacijskem sistemu Windows in vseh drugih računalniških okoljih Unix z uporabo vtičničnih vmesnikov BSD. Knjižnica vsebuje kratko in stabilno kodno bazo, da lahko ustvarjalci na preprost način dodajo podporo tudi za druga računalniška okolja. ENet pa ne predpisuje pravila debelega oziroma tankega konca ali kakšna je velikost besede, tako da mora ta del razvijalec določiti in doprogramirati sam.

Svoboda

ENet ne zahteva nobenih licenčnin in je pod licenco MIT. Programsko opremo pod licenco MIT se sme prodajati in nismo dolžni objaviti izvorne kode programa javnosti. Licenca se pa ne prenese na zaključni proizvod [**?**].

5.3 Realizacija jedra večigralskega sistema

Ko smo se seznanili s knjižnico ENet, je bilo potrebno povezati njene metode v neko celoto. Kot prvi korak sem napisal upravljalca omrežja, ki skrbi za izvajanje ukazov strežnika in odjemalca brez napak.

Za preprosto povezavo je potrebno spisati naslednjih nekaj korakov v metode razreda s poljubnim imenom:

- inicializacija ENeta,
- ustvarjanje strežnika ENet,
- ustvarjanje odjemalca ENet,
- upravljanje gostitelja ENet,
- pošiljanje paketka ENet omrežnemu vrstniku,
- odklopiti ENet omrežnega vrstnika,
- povezava na gostitelja ENet.

Vadnica na spletni strani Eneta pri tem precej pomaga. Po teh korakih sem ustvaril še tri polja (eno za vsako vrsto paketkov) z vidika pošiljanja (nezaporedni, zanesljivi in nezaporedni zanesljivi). Polja je potrebno ob prejetju paketka vsaki okvir (angl. frame) ažurirati s podatki iz paketka. Podatke sem nato obdelal v glavni zanki programa.

Vsaki paketek sem sestavil s sledečimi sestavinami:

- podatek, od katerega omrežnega vrstnika je paketek,
- kanal, po katerem je paketek prispel,
- kazalec na sporočilo paketka,
- velikost sporočila paketka.

Sporočilo paketka vsebuje:

- glavo oziroma naslov sporočila,
- kazalec na podatke sporočila,
- način razpršenega oddajanja,
- vrsto paketka.

Glava oziroma naslov sporočila služi predvsem ločevanju, kateri podatki so v sporočilu (kazalec na podatke sporočila). Na primer: če ima glava sporočila ime matrika objekta, je potem podatek sporočila matrika. Nad sporočilo je potrebno serializirati kot prikazuje podpoglavje 4.4. Ponavadi se paketek tudi šifrira, da ne more noben goljuf pregledati paketka in ga spremeniti. Paketek se še stisne, da je čim manjši. Velikost paketka ne sme preseči velikosti, ki jo določa največja prenosna enota (angl. maximum transmission unit; MTU).

Način razpršenega oddajanja mora vsebovati tri možnosti:

- preskoči lastnika paketka,

- preskoči odjemalca 0,
- pošlji vsem.

Omenjene tri možnosti so pomembne pri odločanju, kakšno arhitekturo odjemalec-strežnik bomo uporabljali. Način razpršenega oddajanja je pomemben pri odločanju, kaj bo strežnik naredil s paketkom, ki ga je dobil od omrežnega vrstnika. Pri opciji "preskoči lastnika paketka" pošlje strežnik naprej paketek vsem omrežnim vrstnikom razen tistemu, od katerega ga je dobil. V takem primeru ima lastnik paketka že posodobljene podatke in ni potrebno, da dobi paketek, ki ga je pravkar poslal. Možnost "preskoči odjemalca 0" je potrebna, če je strežnik hkrati tudi eden od omrežnih vrstnikov, kar pomeni, da je strežnik in prvi odjemalec. Ta možnost se uporablja, če strežnik računa, kje bi moral biti posamezen objekt v igri in nato pošlje podatke vsem odjemalcem razen odjemalcu 0, saj je to dejansko strežnik. Opcija "pošlji vsem" je pomembna, če je strežnik posamezna enota, ki računa dobljene paketke od odjemalcev in sama ni odjemalec. V mojem primeru sem uporabil "preskoči lastnika paketka", saj sem uporabljal arhitekturo vsak z vsakim.

Z vidika pošiljanja ločimo naslednje vrste paketkov:

- nezaporedne paketke,
- zanesljive paketke in
- nezaporedne zanesljive paketke.

Vrste paketkov z vidika pošiljanja so zelo pomembne pri razbremenitvi ozkega grla večigralske mreže. Pošiljanje nezaporednih paketkov lahko interpretiramo kot protokol UDP. V tem primeru paketki lahko tudi ne prispejo do svojega cilja, lahko pa tudi prispejo v drugačnem vrstnem redu, kot so bili poslani. Zanesljive paketke si lahko interpretiramo kot protokol TCP, pri katerem paketki prispejo vsakič na cilj v takšnem vrstnem redu, kot so bili poslani. Slednja možnost je časovno gledano najdaljša in tudi paketki so

večkrat poslani, ker nekateri ne pridejo na cilj. Zadnja možnost je pošiljanje nezaporednih zanesljivih paketkov, ki uporablja del prve in del druge opcije. Paketki prispejo na cilj v vsakem primeru, lahko so pa pomešani med seboj.

Glavni del večigralskega sistema je tako končan, dva igralca se lahko povežeta. Odvisno od tipa igre lahko objekti v igri zelo utripajo zaradi pomanjkanja predvidevanja vnosa, interpolacije in pomanjkanja kompenzacije zakasnitve. Rešitev je vpeljava metod, ki smo jih opisali v poglavju 4.7. Pri realizaciji večigralskega sistema sem izbral arhitekturo vsak z vsakim, da bi pokazal napake pri večigralskem sistemu dirkaške igre. Pravilna odločitev bi bila uporaba arhitekture odjemalec-strežnik. Po vpeljavi teh metod je potrebna izdelava sistema za pregledovanje in obstoj lokalnih iger, nato pa še izgradnja sistema za povezavo na glavni strežnik igre. Izdelava igre z večigralskim sistemom je tako končana. Pri izdelavi večigralskega sistema sem se naslonil na knjigo "Algorithms and Networking for Computer Games" [8], v kateri so s praktičnimi primeri opisani postopki, potrebni za izdelavo sistema za mrežno večigralstvo.

5.4 Problemi pri izdelavi

Pri pripravi celotnega projekta sem imel manjše težave z dodatnimi knjižnicami, ki so bile dodane v projekt, saj imajo na spletni strani OGRE 3D zelo slabo in zastarelo podporo za Applove operacijske sisteme Mac OS X. Kar se tiče večigralskega sistema nas je motila relativno slabo dokumentirana knjižnica ENet, čeprav imajo na domači spletni strani ENeta dopisni seznam, na katerem redno odgovarjajo na vprašanja, ki jih zastavljajo uporabniki ENeta. Dodatno sem naletel na težave, kot so neujemanje poslanih podatkov s prejetimi, predvsem zato, ker sem imel kazalce v objektih, katere sem pošiljal preko mreže. Težave sem odpravil tako, da sem uporabil serializacijo podatkov.

Poglavje 6

Zaključek

Večigralstvo preko omrežja je postalo prava uspešnica v zadnjih letih. Na tak način so prijatelji tudi pri igranju računalniških iger v stalnem stiku, kar spodbuja družabnost in je včasih mnogo bolj zabavno kot da igramo igre sami. Dandanes obstajajo celo igre, ki brez povezave v medmrežje ne delujejo, kot je na primer zelo znana množična spletna večigralska igra World of Warcraft. Mislim, da lahko v prihodnosti pričakujemo še veliko več takšnih.

Izdelava večigralskega sistema je relativno preprosta, a pred tem je potrebno predelati nekaj literature, da smo seznanjeni z vsemi potrebnimi podrobnostmi. Pomemben del diplomske naloge je prikaz različnih arhitektur, dveh pomembnih protokolov ter varnosti in nevarnosti večigralskega omrežja. Možne nevarnosti žal zelo hitro naraščajo, zato je potrebno posredovati na včasih sporne načine, kot je to naredil Blizzard v World of Warcraftu s pomočjo Wardena. Med pomembnimi znanimi protigoljufskimi sistemi, opisanimi v diplomski nalogi, sta še VAC razvijalca Valve, v različnih priljubljenih igrah pa to delo opravlja tudi PunkBuster. Zelo pomembna težava, s katero se soočajo vsi razvijalci večigralskih sistemov, je težava zakasnitve, saj zaradi počasnega prenosa podatkov od enega do drugega računalnika nastajajo neskladnosti, ki se jih da odpraviti s pristopi, kot so napoved vnosa, interpolacija in kompenzacija zakasnitve. Po delujočem jedru večigralskega sistema je potrebno v igri izdelati tudi sistem iskanja lokalnih strežnikov, s

katerim poiščemo vse aktivne primerke iger in se nato na enega povežemo. S prihodom arhitekture odjemalec-strežnik je običajno v igrah tudi glavni strežnik, ki prav tako ponuja primerke iger, ki so na voljo, in na primer dodatne informacije igralcev. Pri izdelavi večigralskega sistema igre je zelo pomemben korak odločitev, ali želimo brezplačno rešitev. Če uporabljamo igralske pogone, kot sta Unity in UDK, imamo večigralski sistem že vgrajen v pogon. Izdelavo računalniške večigralske omrežne igre sem realiziral s pomočjo omrežne knjižnice ENet, ki je preprosta za uporabo, saj lahko že samo s pomočjo vadnice naredimo prototip delujočega večigralskega sistema.

Danes se izide le redko katera računalniška igra brez omrežnega večigralskega sistema. Implementacija večigralskega sistema popestri igro in ji doda nove izzive. Poleg dodatne zabave poskrbi večigralski sistem za dodaten način, kako smo lahko v stiku s prijatelji.

Seznam slik

Literatura

[1] G. Armitage, M. Claypool, P. Branch, *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*, John Wiley & Sons, 2006.

[2] F. Glinka, A. Ploss, S. Gorlatch, J. Müller-Iden, "High-Level Development of Multiserver Online Games", *International Journal of Computer Games Technology*, 2008.

[3] B. Hallberg, *Networking: A Beginner's Guide*, 5. izd., The McGraw-Hill Companies, 2010.

[4] G. Junker, *Pro OGRE 3D Programming*, Apress, 2006.

[5] A. Lake, *Game Programming Gems 8*, Course Technology, 2011.

[6] A. Mulholland, T. Hakala, *Programming Multiplayer Games*, Wordware Publishing, Inc., 2004.

[7] S. Rabin, *Introduction to Game Development*, 2. izd., Charles River Media, 2010, str. 603-640.

[8] J. Smed, H. Hakonen, *Algorithms and Networking for Computer Games*, John Wiley & Sons, 2006.

[9] C10K problem. Dostopno na:
http://en.wikipedia.org/wiki/C10k_problem
(datum zadnjega obiska: 25.8.2011)

[10] Cube. Dostopno na:
http://assault.cubers.net/
(datum zadnjega obiska: 24.8.2011)

[11] (2000) Defeating Lag With Cubic Splines. Dostopno na:
http://www.gamedev.net/page/resources/_/reference/programming/multiplayer-and-networking/268/defeating-lag-with-cubic-splines-r914
(datum zadnjega obiska: 19.8.2011)

[12] Diablo 3 to feature player-to-player real money auction house for virtual items. Dostopno na:
http://www.joystiq.com/2011/08/01/diablo-3-to-feature-player-to-player-real-money-auction-house-fo/
(datum zadnjega obiska: 24.8.2011)

[13] (2004) Dynamics Simulation. Dostopno na:
http://www.ode.org/slides/parc/dynamics.pdf
(datum zadnjega obiska: 25.8.2011)

[14] Enet. Dostopno na:
http://enet.bespin.org/
(datum zadnjega obiska: 24.8.2011)

[15] Game design document. Dostopno na:
http://en.wikipedia.org/wiki/Game_design_document
(datum zadnjega obiska: 28.8.2011)

[16] MIT license. Dostopno na:
http://www.opensource.org/licenses/mit-license.php
(datum zadnjega obiska: 11.9.2011)

[17] More People Playing The Altitude Demo Than Altitude. Dostopno na:
http://forums.steampowered.com/forums/showthread.php?t=1425900
(datum zadnjega obiska: 3.9.2011)

[18] Multiplayer - interpolation / prediction. Dostopno na:
http://www.racer.nl/tech/multiplayer.html
(datum zadnjega obiska: 22.8.2011)

[19] PunkBuster. Dostopno na:
http://www.evenbalance.com/
(datum zadnjega obiska: 5.9.2011)

[20] Serialization. Dostopno na:
http://en.wikipedia.org/wiki/Serialization/
(datum zadnjega obiska: 24.8.2011)

[21] Source Multiplayer Networking. Dostopno na:
http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking
(datum zadnjega obiska: 22.8.2011)

[22] Spacewar. Dostopno na:
http://en.wikipedia.org/wiki/Spacewar!
(datum zadnjega obiska: 2.9.2011)

[23] Tennis for two. Dostopno na:
http://en.wikipedia.org/wiki/Tennis_for_Two
(datum zadnjega obiska: 2.9.2011)

[24] High Level Networking Concepts. Dostopno na:
http://unity3d.com/support/documentation/Components/net-HighLevelOverview.html
(datum zadnjega obiska: 22.8.2011)

[25] Valve Anti-Cheat (VAC). Dostopno na:
http://en.wikipedia.org/wiki/Valve_Anti-Cheat
(datum zadnjega obiska: 27.8.2011)

[26] Warden. Dostopno na:
http://en.wikipedia.org/wiki/Warden_(software)
(datum zadnjega obiska: 27.8.2011)

[27] What are some ways to prevent or reduce cheating in online multiplayer games? Dostopno na:
http://gamedev.stackexchange.com/
(datum zadnjega obiska: 4.9.2011)