STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN TOIMIALA

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

VERKKOPELIN SYNKRONOINTI

Työn tekijä: Juuso Savolainen
Työn valvoja: Matti Luukkainen
Työn ohjaaja: Matti Luukkainen

Työ hyväksytty: 25. 1. 2007

Matti Luukkainen
Yliopettaja

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Juuso Savolainen

Työn nimi: Verkkopelin Synkronointi

Päivämäärä: 25.1.2007 Sivumäärä: 37 s.

Koulutusohjelma: Tietotekniikka Suuntautumisvaihtoehto: Ohjelmistotekniikka

Työn valvoja: Matti Luukkainen

Työn ohjaaja: Matti Luukkainen

Monen pelaajan nettipeleissä on ongelmana yhteyksien hitaudesta johtuva viive. Tästä on haittaa erityisesti reaaliaikaisissa peleissä, koska pelaajan liikkeet eivät päivity tarpeeksi nopeasti toisille pelaajille: kun pelaaja liikkuu johonkin suuntaan ja päättää kääntyä toiseen suuntaan niin muut pelaajat saavat tiedon viiveellä. Tästä aiheutuu pelaajien liikkeiden töksähtelyä.

Työssä selvitetään, miten nämä ongelmat saadaan ratkaistua. Aluksi kerrotaan yleisesti nettipeleistä ja UDP-protokollaan liittyvistä ongelmista. Sitten esitetään erilaisia ratkaisuja synkronointiin, kuten delta encoding ja dead reckoning. Lopuksi esitellään itse suunniteltu rajapinta, joka helpottaa moninpelin toteutusta peleihin.

Avainsanat: verkkopeli, internet, UDP, synkronointi, delta encoding, dead reckoning

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Juuso Savolainen

Title: Network game synchronization

Date: 25.1.2007 Number of pages: 37

Department: Information technology Study Programme: Software engineering

Instructor: Matti Luukkainen

Supervisor: Matti Luukkainen

In multiplayer internet games there is a problem, result from slowness of internet connections. This is a disadvantage especially for real time games, because players' movements aren't updated to other players quickly enough: when player moves to some direction and decides to turn to another direction, then other players get the information about that with lag. This causes that player movements aren't smooth but jumping.

In this work it is find out how these problems are solved. First is told generally about internet games and about problems concerning UDP protocol. Then is explained some solutions for synchronization, like delta encoding and dead reckoning. Lastly is introduced self-designed programming interface, which helps the implementation of multiplayer game.

Keywords: network game, internet, UDP, synchronization, delta encoding, dead reckoning

SISÄLLYS

1 JOHDANTO

Verkkopelit, jotka tunnetaan myös moninpeleinä tai nettipeleinä, ovat suosittua ajanvietettä, koska niitä voi pelata muiden ihmisten kanssa internetin välityksellä, ja ihmispelaaja tuo peliin toisenlaista haastetta kuin tekoäly. Kaikki peliin liittyneet pelaajat ovat samassa pelimaailmassa. Lisäksi pelissä voi kehittyä jatkuvasti paremmaksi.

Verkkopelit alkoivat yleistyä 90-luvun puolivälissä, kun Id Software julkaisi suositun Quake-räiskintäpelin quake world –muunnoksen. Tätä pelatessa saattoi kokea olevansa oikeasti jossain toisessa maailmassa, koska tekniikka oli niin uutta ja muiden ihmisten kanssa pelaamista internetin välityksellä ei ollut ennen kokenut. Kuvassa 1 on ruudunkaappaus kyseisestä pelistä.

Kuva 1. Id softwaren tekemän Quaken grafiikka oli huikeaa siihen aikaan.

Nykyään on tarjolla monenlaisia nettipelejä, kuten web-selaimella pelattavat pokeri ja muut rahapelit. Suosittuja ja addiktoivia pelejä ovat internet-roolipelit eli MMORPG:t (Massive Multiplayer Online Role-playing Game.) Roolipelin pelaamiseen tarvitsee monesti paljon aikaa, koska pelimaailma ja hahmot kehittyvät koko ajan, eikä pelillä ole selvää päättymistä.

Edellä esitellyt pelityypit eivät välttämättä vaadi monimutkaista tietoliikennepakettien käsittelyä. Esim. roolipelit ovat monesti vuoropohjaisia ja hitaasti eteneviä, joten niissä pelaajan tekemisiä ei tarvitse saada hetkessä palvelimelle tai palvelimelta muille pelaajille. Jotakin roolipelejä pelataan jopa sähköpostin välityksellä.

Kehittyneempää verkkopelitoteutusta tarvitaan yleensä nopeatempoisiin peleihin, joissa pelin tapahtumat pitää saada välitettyä reaaliajassa kaikille pelaajille. Pelaamisen pitäisi vaikuttaa siltä, että kaikki pelaajat ovat pelimaailmassa samassa ajassa.

Monet verkkopelit perustuvat siihen, että on palvelin, joka on yhdistetty internetiin. Pelaajat ottavat yhteyden palvelimeen ja lähettävät sille tiedot omista tekemisistään. Palvelin määrää kaikki pelin tapahtumat ja välittää ne pelaajille. Internetyhteyksien hitaus on ollut alusta asti ongelmana, joka haittaa verkkopelien pelattavuutta. Kaikki pelaajan liikkeet pitää lähettää muille pelaajille palvelimen kautta. Jos yhteydessä on pitkä viive ja paketin kulkeminen palvelimelta asiakkaalle kestää pitkään, niin pelaajan liikkeen muutokset eivät välity muille pelaajille tarpeeksi nopeasti. Jos pelaaja on paikallaan ja lähtee liikkeelle, niin liikkeelle lähtö näkyy muilla pelaajilla myöhemmin. Pahimmillaan tämä voi aiheuttaa pelaajan hyppimistä paikasta toiseen.

Monissa reaaliaikaisissa verkkopeleissä käytetään UDP-protokollaa sen nopeuden takia. Esim. pokeriin sopii paremmin TCP-protokolla, koska pelin tapahtumat on saatava aina perille, mutta nopeus ei ole oleellista. Protokollat ovat tiedonsiirtotapoja, joita on helppo käyttää esim. socket-rajapinnalla.

Työn tavoitteena on kehittää algoritmi pelin tilan muutosten siirtämiseen palvelimelta asiakkalle mahdollisimman nopeasti ja kehittää rajapinta, jolla saa toteutettua peliin helposti verkkopeliominaisuuden. Aluksi kerrotaan harrastuksena tehdystä pelistä, jota varten rajapinta on kehitetty. Sitten kerrotaan UDP-protokollaan liittyvistä ongelmista ja esitetään erilaisia ratkaisuja synkronointiin. Näitä ovat delta encoding ja dead reckoning. Lopuksi esitellään itsesuunnitellun ohjelmointirajapinnan toteutus.

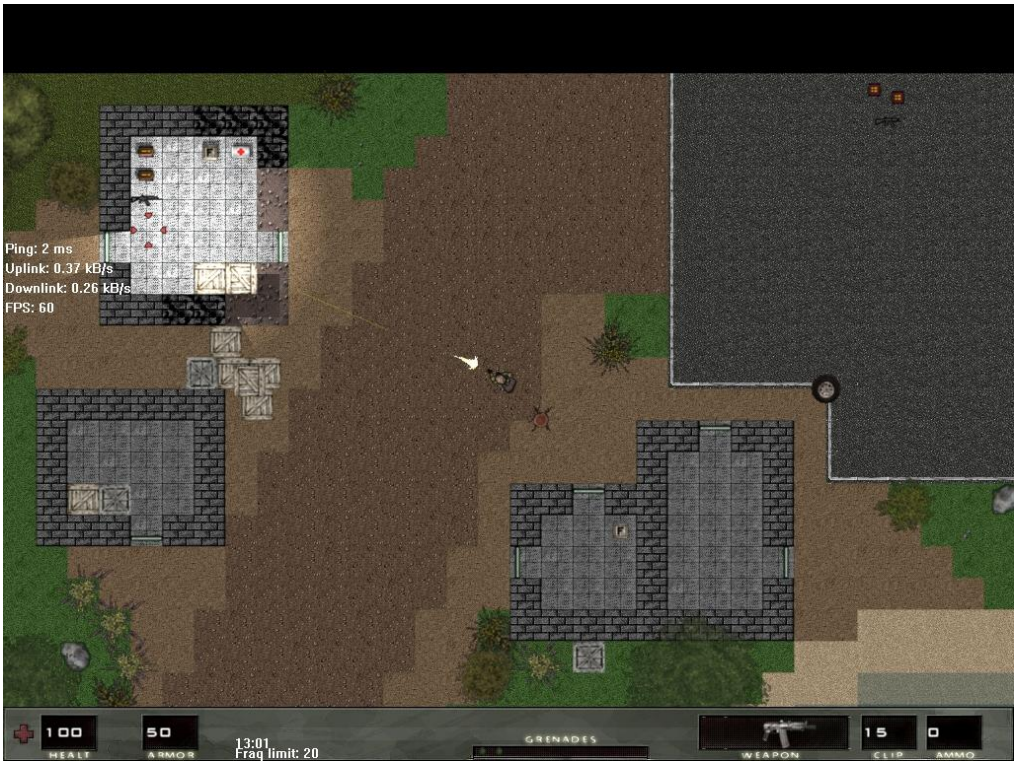
2 PELINTEKOHARRASTUS

Pelien tekeminen on ollut minulle tärkeä harrastus, joka alkoi yläasteaikana, kun pelasin johdannossa mainittua Quakea ja innistuin oman pelin tekemisestä sen hienon grafiikan takia. Olin jo ohjelmoinut basic-ohjelmointikielellä yksinkertaisia tekstiseikkailuja, joten ohjelmointi ei ollut ihan vieras asia. Harrastus on vienyt mukanaan, koska siinä saa käyttää luovuutta ja ratkoa monimutkaisia teknisiä ongelmia. Lisäksi tekniikka, kuten 3D-kiihdyttimien nopeus ja ominaisuudet kehittyvät koko ajan ja aina pystyy tekemään entistä hienompia grafiikkamoottoreita.

Viimeisin peli, jota olen tehnyt on Killing Engine -niminen ylhäältä päin kuvattu internetissä pelattava ammuskelupeli. Pelin ideana on sotiminen internetin välityksellä muita pelaajia vastaan tiimeissä. Pelin on tarkoitus olla hauskaa ajanvietettä urbaanilla ja koomisella pelimaailmalla ja hienolla grafiikalla. Pelin grafiikkaan on käytetty uusimmista 3D-peleistä tuttua pikselintarkkaa valaistusta ja dynaamisia varjoja. Kuvasta 2 näkee, kuinka reunassa olevan rakennuksen seinä on hajonnut ja rakennuksessa oleva valo tulee ulos hajonneen seinän kohdalta. Myös varjon paikka on muuttunut.

Nimi Killing Engine saattaa johtaa harhaan raa'asta pelistä, joka on vain armotonta tappamista, mutta oikeasti nimen on tarkoitus herättää mielenkiintoa ja kertoa pelin koomisuudesta. Peliä on tehty kahden vuoden ajan internetistä kasatun kolmen jäsenen tiimillä. Myös pelin kehitys ja yhteydenpito hoituu internetin välityksellä.

Tällä hetkellä peli on viimeistelyä vaille valmis pelimoottori, josta puuttuu vielä sisältö, eli erilaiset tapahtumapaikat ja tehtävät. Koska osalta tiimin jäsenistä on loppunut mielenkiinto pelin tekemistä kohtaan niin suunnitelmissa on, että peliin lisättäisiin skriptit, joilla valmiin pelin eri osiin voi ohjelmoida uutta toiminnallisuutta. Tämä mahdollistaisi sen, että kuka tahansa voisi kehittää peliin uutta sisältöä ja käyttä skriptejä omien muunnosten, eli modien tekemiseen.



Kuva 2. Killing Enginen pelimaailma koostuu palikoista ja pelaaja näkee maailman ohjaamansa hahmon kohdalta, ylhäältä päin.

Työn verkkopelimoduli on suunniteltu tätä peliä varten. Peliä varten on tehty myös kaksi muuta verkkopelimodulia, jotka eivät kuitenkaan ole toimineet tarpeeksi hyvin. Ensimmäiset versiot lähettivät kaikki objektien muutokset asiakkalle paketteina, joiden perillemeno varmistettiin. Tämä ratkaisu käytti nykyisen modulin tapaan UDP-protokollaa, jossa paketit eivät välillä mene perille. Pakettien siirtoa ei ollut suunniteltu kunnolla, ja yhteys tukkeutui jossain tapauksissa niin, että peliä ei pystynyt enää pelaamaan ollenkaan.

Ensimmäisissä versioissa ei ollut ollenkaan rajapintaa, jonka avulla verkkopelimoduli olisi liitetty siistimmin peliin, vaan verkkopelikoodia oli hajallaan pelin lähdekoodissa. Nykyisen verkkopelimoduulin toteutus on erotettu pelilogiikasta rajapintaluokkien avulla.

3 INTERNET

Tässä luvussa kerrotaan internetin rakenteesta ja eri menetelmistä, jolla tietoa voi siirtää internetissä.

3.1 Pakettien reititys

Internet on epäluotettava tietoverkko, koska lähetettävät IP-paketit voivat kadota matkalle: bittejä siirtyy virheellisinä, jokin reititin ei toimi oikein tai paketti kulkee liian pitkän matkan. Paketti voi myös viipyä matkalla niin pitkään, että sen luullaan kadonneen.

Tietoa lähetetään internetissä paketteina IP-protokollan avulla. Paketti lähetetään johonkin kohdeosoitteeseen, joka on määritetty paketin otsikossa. Paketit kulkevat internetissä reitittimien kautta. Lähettävä osapuoli lähettää paketin sille reitittimelle, johon se on kytketty. Reititin tarkistaa IP-paketista mihin paketti ollaan lähettämässä ja lähettää paketin oikeaan suuntaan seuraavalle reitittimelle. Pakettia lähetetään näin eteenpäin, kunnes se on lopulta saavuttanut kohteen.

Pelkällä IP-protokollalla ei kuitenkaan lähetetä tietoa vaan siihen käytetään ylemmän tason (kuljetuskerros) protokollia, kuten TCP:tä ja UDP:tä. Nämä lisäävät IP-pakettiin porttinumeron, jolla paketteja voi lähettää kohdeosoitteessa olevalle ohjelmalle. Lisäksi protokollissa on muita pakettien siirtoon vaikuttavia ominaisuuksia.

3.2 TCP-protokolla

TCP-protokollassa internetin puutteet on huomioitu ja hävinneet tai virheelliset paketit lähetetään automaattisesti uudestaan. TCP:n huono puoli verkkopelien suhteen on, että hävinnyt paketti voi sisältää vanhaa tietoa. Pelin objektin tila on voinut muuttua kokonaan toiseksi, jolloin on turha lähettää hävinnyttä pakettia uudestaan, koska se saattaisi vain tukkia yhteyden.

3.3 UDP-protokolla

UDP-protokolla eroaa TCP-protokollasta siten, että se on yhteydetön, paketteja ei lähetetä uudestaan niiden hävitessä eikä käyttöjärjestelmä lähetä vastausta paketteihin. Siinä ei myöskään ole vuonohjausta (flow control), kuten TCP-protokollassa, vaan paketit lähetetään heti. Tämä aiheuttaa sen, että jos paketteja tulee liikaa, eikä sovellus ehdi niitä käsitellä, niin käyttöjärjestelmän pakettien jonolle varaama muistialue voi täyttyä ja paketti joudutaan hylkäämään. [3.]

UDP-paketti lähetetään käyttöjärjestelmän palvelulla johonkin osoitteella ja porttinumerolla määrättyyn kohteeseen. Jos paketti menee perille, niin käyttöjärjestelmä tarkistaa paketissa olevan tarkistussumman avulla, onko paketti ehjä. Ehjä paketti välitetään eteenpäin sille sovellukselle, jonka paketin porttinumero määrää. Paketti voi kuitenkin saapua väärässä järjestyksessä ja useampaan kertaan.

UDP-paketteja kutsutaan myös datagrammeiksi, koska niiden perillemenoa ei ole taattu.

3.4 Soveltuvuus verkkopeleihin

Verkkopeleissä käytetään UDP-protokollaa, koska pakettien hävitessä ei tarvitse lähettää samaa pakettia uudestaan vaan tällöin lähetetään uusi paketti, jossa on objektien uudet tiedot. TCP-protokollalla yritettäisiin lähettää paketin kadotessa aina turhaan samat tiedot uudestaan. UDP-protokollalla saa myös tarvittaessa toteutettua pelin muuta tiedonsiirtoa varten samanlaisen häviöttömän tiedonsiirron, joka TCP-protokollassa on.

3.5 Socket–sovellusrajapinta

Tiedonsiirto on toteutettu käyttöjärjestelmään, ja sovellus voi käyttää näitä palveluita käyttöjärjestelmän funktioiden avulla, joita kutsutaan socket-rajapinnaksi. Standardi rajapinta tähän on Berkley sockets, jonka mukainen myös Windowsin toteutus, winsock on [5].

Socketin luonti

Rajapinta käyttää deskriptoreita, eli kokonaislukuja, joita sovellus käyttää kertomaan rajapinnalle mihin socketiin funktion kutsu kohdistetaan. Ensin on siis luotava socketi, johon voidaan sen jälkeen asettaa erilaisia ominaisuuksia ja lopuksi siirtää tietoa socketin kanssa. Socketi luodaan funktiolla `int socket(int domain, int type, int protocol)`. Domain-parametrina käytetään PF_INET:ä, joka tarkoittaa, että socketilla käytetään IPv4-internet protokollaa.

Socketin sitominen osoitteeseen

Sitomisella määrätään, mistä osoitteesta socketilla lähetettävät paketit ovat tai mitä osoitetta palvelin kuuntelee. Sitominen tehdään funktiolla `int`

`bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen)`. Sockaddr-tietorakenteella määrätään, mihin porttiin ja ip-osoitteeseen socketi sidotaan.

IPv4:llä sockaddr korvataan sockaddr_in:llä, jonka määrittely on kuvassa 3.

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

Kuva 3. Tietorakenne, jota käytetään IPv4:llä kohteen määrittämiseen.

Tietorakenteen sin_family-muuttujan arvoksi asetetaan PF_INET, joka kertoo, että kyseessä on IPv4-tietorakenne. sin_port-muuttuja määrää, mihin porttiin socketi sidotaan ja sen arvon täytyy olla oikeassa bittijärjestyksessä. Porttinumeron saa muutettua oikeaan bittijärjestykseen (*network byte order*) htons-funktiolla tai takaisin alkuperäiseen järjestykseen (*host byte order*) ntohs-funktiolla. sin_addr-muuttujan arvoksi astetetaan INADDR_ANY, jolloin käyttöjärjestelmä valitsee oikean osoitteen.

Jos samaan osoitteeseen on jo sidottu jokin toinen socketi, niin sitominen epäonnistuu ja bind-funktio palauttaa arvon -1.

TCP–socket

Jos halutaan käyttää TCP-protokollaa niin täytyy luoda yhteydellinen socketi socket-funktion type-parametrin arvolla SOCK_STREAM ja protocol-parametrin arvolla IPPROTO_TCP. SOCK_STREAM tarkoittaa, että socketi on yhteydellinen ja se täytyy aina ensin yhdistää connect-funktiolla tai aloittaa kuunteleminen listen-funktiolla. SOCK_STREAM myös takaa sen, että kaikki tieto menee perille muuttumattomana ja samassa järjestyksessä kuin se on lähetetty.

Ennen kuin tietoa voi siirtää, niin palvelimen täytyy sitoa socketi johonkin paikalliseen osoitteeseen ja aloittaa kuunteleminen. Asiakkaan täytyy yhdistää socketi palvelimen osoitteeseen.

Sitominen tehdään luvun 0 ohjeiden mukaan bind-funktiolla, määrittämällä tietorakenteeseen jokin osoite ja portti. Jotta asiakas voisi ottaa yhteyden palvelimeen, niin socketi on asetettava kuuntelutilaan funktiolla `int listen(int s, int backlog)`, jossa s-parametri on socket-deskriptori ja backlog-parametrilla kerrotaan, kuinka monta yhteydenottoa enintään pidetään jonossa. Tämän jälkeen palvelin voi kuunnella uusia yhteydenottoja funktiolla `int accept(int s, struct sockaddr *addr, socklen_t *addrlen)`. Funktio käyttää addr-parametrina luvussa 0 esiteltyä tietorakennetta. addrlen-parametrilla kerrotaan tietorakenteen koko. Kun yhteys on hyväksytty, niin accept-funktion palauttaman socketin kanssa voi siirtää tietoa palvelimen ja asiakkaan välillä. Asiakas voi muodostaa yhteyden palvelimeen funktiolla `int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen)`. serv_addr-parametrina käytetään luvussa 0 esiteltyä tietorakennetta. Jos yhteys saadaan niin funktio palauttaa arvon 0. Tämän jälkeen asiakas voi lähettää ja vastaanottaa tietoa palvelimelta.

UDP–socket

UDP-protokollaa varten luodaan datagram-socketi socket-funktion type-parametrin arvolla SOCK_DGRAM ja protocol-parametrin arvolla IPPROTO_UDP. Datagram socketi on yhteydetön ja paketteja voidaan lähettää heti mihin vaan osoitteeseen. Paketteja voi kadota, mennä perille väärässä järjestyksessä ja monistua. UDP-socketi täytyy ainakin palvelimella sitoa johonkin paikalliseen osoitteeseen ennen kuin tietoa voi siirtää. Asiakkaan ei ole pakko tehdä tätä vaan socketi sidotaan automaattisesti johonkin vapaaseen porttiin, kun tietoa lähetetään ensimmäisen kerran. UDP-socketia ei tarvitse yhdistää palvelimeen, eikä palvelimen tarvitse kuunnella socketia vaan paketteja voi heti lähettää mihin vaan osoitteeseen tai lukea socketia, jolloin lukufunktio odottaa, että paketti saapuu siihen osoitteeseen, mihin socketi on sidottu.

Tiedonsiirto

TCP-protokollaa käytettäessä tietoa voi siirtää funktioilla `int send(int s, const char* buf, int len, int flags)` ja `int recv(int s, char* buf, int len, int flags)`.

UDP-protokollaa käytettäessä tietoa voi siirtää funktioilla `int sendto(int s, const char* buf, int len, int flags, const struct sockaddr* to, int tolen)` ja `int recvfrom(int s, char* buf, int len, int flags, struct sockaddr* from, int* fromlen)`. Funktioissa on parametrit osoitetta varten. Vastaanotettaessa osoite kertoo mistä paketti on tullut ja lähetettäessä mihin paketti ollaan lähettämässä.

4 SYKRONOINTI UDP-PROTOKOLLALLA

Tässä luvussa kerrotaan, miten internetin aiheuttamat, verkkopelejä haittaavat ongelmat ratkaistaan sopivan algoritmin avulla. Seuraavana on tekstissä käytettyjä määritelmiä.

- *Objektilla* tarkoitetaan pelissä olevaa hahmoa tai esinettä.
- Objektilla on sijainti, asento, kävelysuunta ja muita tietoja, joita kutsutaan tässä *tietokentiksi*.
- Sovellusta jolla pelaaja pelaa peliä kutsutaan *asiakkaaksi*.
- *Palvelin* on sovellus johon kaikki pelaajat ovat yhteydessä ja joka määrää pelin tapahtumien kulun.
- *Pelin tila* tarkoittaa kaikkien objektien sijaintia, asentoa ja muita tietoja.
- *kiertoaika* on aika, joka kestää, kun paketti lähetetään ja siihen saadaan vastaus.

4.1 Yhteyksiin liittyvät ongelmat

Yksinkertainen verkkopelin toteutus päivittäisi kaikki asiakkaalle näkyvät objektit tarpeeksi monta kertaa sekuntissa, eikä välittäisi siitä, kuinka paljon tietoa siirretään. Peleissä riittävä ruudunpäivitusnopeus on 50 ruutua sekuntissa, koska alhaisemmalla nopeudella pelaaja huomaa, että ohjaus ei anna välitöntä vastetta ruudulla, eli pelaajan ohjaama hahmo liikkuu tahmeasti.

UDP-paketti menee ADSL-yhteydellä perille noin viidessä millisekuntissa riippuen reitittimien ja palvelimen kuormasta. Silloinkin paketin pitää olla tarpeeksi pieni. Jos yhteyden nopeus on 8 Mbps sisään, niin tietoa voi vastaanottaa palvelimelta 1024 kt/s. Paketin perillemenoajan voi laskea tästä kaavalla p+x/s, jossa p on paketin kiertoaika, x paketin koko ja s yhteyden nopeus.

Jotta pelin tila päivittyisi palvelimelta asiakkaalle tarpeeksi nopeasti niin kaikki tiedot pelistä pitää lähettää 50 kertaa sekuntissa. Tällöin yhden päivityksen koko voisi olla enintään 1024kt / 50 = 20,48 kt. Jos peli lähettäisi päivityksessä kaikkien pelaajien tiedot ja pelaajia olisi 32 kpl, niin yhden pelaajan tieto voisi viedä enintään 20,48 kt / 32 = 655 tavua.

Monissa palvelimissa on käytössä vain 100 Mbps:n yhteys, jolloin se riittäisi 12800 kt/s / 32 = 320 kt/s. Tällöin yhden pelissä olevan pelaaja-objektin tietoja varten olisi vain 205 tavua. 320 kt/s pelaajaa kohti riittäisi hyvin peliä varten, koska pelaajan sijainnin ilmoittamiseen tarvitsee vain 10-20 tavua.

Peli pitää kuitenkin saada toimimaan niin, että kuka tahansa voi pitää omaa palvelinta ADSL-yhteydellä, jossa paluukaistan nopeus on 1 Mbps. 32 pelaajalla palvelin voi lähettää enintään 4 kt/s. Tässä tapauksessa alkaa ilmetä ongelmia, jos pelin kaikki tedot lähetetään koko ajan.

Tavumäärä joka voidaan lähettää yhtä ruutua kohden, saadaan kaavalla 4 kt / 50 = 82 tavua. Tällöin paketti menee perille ajassa, jonka saa kaavalla 5 + 82 t / 4096 t/s = 25 ms. Matkalla voi olla edellisiä paketteja, jolloin aika pitenee.

Jos palvelimen ja asiakkaan välimatka on tarpeeksi pitkä, niin paketin kiertoaika on kymmeniäkin millisekunteja. Esim. 30 ms:n paketin kiertoajalla pelin tiedot voi edelleen päivittää 50 kertaa sekuntissa, mutta asiakkaan vastaanottaessa päivityksen palvelin on jo 30 ms edellä. Lähettämällä mahdollisimman pieniä paketteja kiertoaikaa saa tavallaan pienemmäksi, koska yksittäinen paketti menee tällöin nopeammin perille. Toisaalta, jos ollaan lähettämässä vielä edellistä päivitystä, niin kestää pitempään.

Suurempi latenssi toimii vielä esim. autopeleissä, joissa auton kulkusuunta ei muutu nopeasti. Räiskintäpeleissä suurempi latenssi vaikeuttaa tähtäystä, kun objekti ei ole palvelimella samassa kohdassa kuin pelaajalla.

Edellä kerrotun perusteella voi todeta, että verkkopelin toteutukseen liittyy seuraavia seikkoja:

- Jos tiedot kaikista pelissä olevista objekteista lähetetään, tietoliikenneyhteyksien kaista ei riitä kaiken tiedon siirtämiseen tarpeeksi nopeasti ja peli voi jähmettyä kokonaan. Jos kaistaa olisi loputtomasti, niin tämä tekniikka olisi riittävä eikä edes pakettien katoaminen olisi ongelma.

- Koska kaistanleveys on rajoitettu, lähetetään vain se tieto, mikä on muuttunut, mutta ei kuitenkaan liian usein. Tässä täytyy kuitenkin ottaa huomioon, että päivitys on edellistä tapausta tärkeämpää saada perille, koska UDP-yhteyksillä pakettien katoaminen on yleistä.
- Pakettien saapumisjärjestys on huomioitava.

Koska kaistanleveys on rajoitettu, joudutaan lähettämään vain kullekin pelaajalle näkyvien objektien tärkeimmät muuttuneet tiedot ja kehittämään algoritmi, joka hallitsee pakettien häviämisen ja järjestyksen sotkeutumisen. Seuraavissa luvuissa kerrotaan, miten tarkoitukseen sopiva algoritmi toimii.

Verkkopelissä on tärkeää lähettää paketteja mahdollisimman harvoin. Jos pelaaja päättää lähteä liikkeelle, niin tieto tästä pitää saada mahdollisimman nopeasti muille pelaajille. Kun kaistaa on säästetty, niin paketteja ei ole lähetettävänä, jolloin kaista on kriittisenä hetkenä vapaana ja tilan muutos saadaan nopeasti perille. Parhaiten tämä toimii jos paketit lähetetään tasaisissa jaksoissa. Silloin tärkeä objektin päivitys saadaan lähetettyä sitä nopeammin mitä pienempiä paketit ovat. Liian pienet paketit kuitenkin hukkaavat kaistaa, koska jokaisessa paketissa on otsikkodataa 40 tavua.

4.2 Muuttuneen tiedon lähettäminen

Verkkopelissä joudutaan lähettämään vain objektien muuttuneet tietokentät, jotta tietoliikenteen määrä ei kasvaisi liian suureksi. Jos liikennettä on liikaa, niin tärkeät viestit pelin tapahtumista viipyvät pitempää matkalla muiden tietojen tukkiessa yhteyden eivätkä pelin tapahtumat välity pelaajille reaaliajassa.

Moni pelaaja käyttää verkkopelaamisessa erilaisia huijauskeinoja pärjätäkseen pelissä paremmin. Näitä voivat olla esim. peliin rakennettu lisäosa, jolla näkee missä, muut pelaajat ovat, tai automaattinen tähtäys. Pelkkien välttämättömien tietojen lähettäminen ratkaisee samalla myös tämän ongelman.

ID	Sijainti	Suunta
1	42	100
1	83	100

Kuva 4. Objektin kentät.

Muuttuneiksi tiedoiksi voidaan ajatella eri asioita. Yksinkertaisimmillaan yksittäinen objekti voi olla muuttunut, jos sen joku kenttä on muuttunut ja tällöin

koko objekti lähetetään. Myös objektin kenttiä voidaan vertailla erikseen ja lähettää vain muuttuneet kentät. Jos halutaan mahdollisimman vähän tiedonsiirtoa, niin muuttuneista kentistä voi lähettää vain muuttuneet osat tai muutoksen suuruuden.

Muuttuneiden kenttien tai kentän muutossuuruuden lähettämistä kutsutaan nimellä delta encoding.

Kuvassa 4 on muuttunut objektin sijainti. Tässä voisi lähettää kokonaan lukeman 83 tai muutossuuruuden +41.

4.3 Objektien lisäys ja poisto

Uusia objekteja, kuten pelaajia, luotien valojuovia ja erilaisia esineitä voi ilmestyä pelimaailmaan kesken pelin. Objekti voi myös hävitä, kun esim. esine poimitaan maasta tai pelaaja poistuu pelistä. Näistäkin tapahtumista täytyy lähettää tiedot pelaajalle. Jotta tietoa siirrettäisiin mahdollisimman vähän, objekteista lähetetään vain ne, jotka pelaajan on mahdollista nähdä. Tähän liittyy monia ongelmia, osa UDP-protokollan puutteiden takia.

Vastaanottajalle voi saapua väärässä järjestyksessä paketti, jossa on tieto uudesta objektista ja sen tyypistä. Jos tätä pakettia ennen vastaanotettu uudempi paketti sisältää objektin tietoa, niin sitä ei voida käsitellä, koska objektin tyyppi ei ole vielä selvillä. Objektin tyyppi täytyy tietää, jotta vastaanotetut tiedot osattaisi kopioida objektin kenttiin oikein.

Jos väärässä järjestyksessä saapunut paketti sisältää tietoa objektista, jota ei vielä ole luotu, niin paketissa tämän jälkeen olevia muidenkaan objektien tietoja ei voi vielä käsitellä, koska kenttien sijainti paketissa ei ole selvillä. Tällöin paketti on tallennettava ja käsiteltävä puuttuneen objektin luonnin jälkeen, kun oikea paketti on vastaanotettu.

Myös objektien poistamisessa on huomioitava väärässä järjestyksessä saapuneet paketit. Poistettuun objektiin voi tulla vielä päivitys, jonka takia objektia on pidettävä muistissa jonkin aikaa poistamisen jälkeen.

Kun pelaaja siirtyy tarpeeksi kauas objektista, niin objekti ei enää näy pelaajan ruudulla ja objektin tietoja ei tarvitse enää päivittää pelaajalle. Objekti voi kuitenkin liikkua palvelimella, jolloin pelaajalla säilyy objektista vanha sijainti. Jos pelaaja liikkuu takaisin siihen kohtaan missä objekti on sillä, ja palveli-

mella objekti on siirtynyt kauas siitä, niin objekti näkyy väärässä kohdassa. Tämän takia objekti täytyy piilottaa jos se siirtyy tarpeeksi kauas pelaajasta. Pelaajalle lähetetään objektien päivitysten yhteydessä tieto siitä, että objekti on siirtynyt pois ruudulta. Myös tässä on huomioitava, että objektia ei saa piilottaa, jos paketti on vastaanotettu väärässä järjestyksessä, ja uudempi paketti on asettanut objektin uudelleen näkyväksi.

4.4 Liikkeiden pehmennys

Tiedonsiirtokaistaa voidaan säästää myös sillä, että objektin liikkeistä lähetetään vain liikesuunnan ja nopeuden muutokset, eikä koko objektin koordinaatteja aina, kun ne muuttuvat. Objektin tarkka sijainti ja katselusuunta lähetetään vain tietyin väliajoin.

Objektin liike alkaa kuitenkin tökkiä, jos sijaintia ei päivitetä tarpeeksi usein. Tähän auttaa interpolointitekniikka, jota kutsutaan myös nimellä dead reckoning. Tekniikalla yritetään ennustaa objektin liikerata. Räiskintäpelissä objektin oletetaan jatkavan liikettä samaan suuntaan vaikka paketteja ei kulkisi välillä yhteyden jumituttua. [1, 2.]

Pelaaja voi kuitenkin päättää kääntyä toiseen suuntaan ja tällöin objekti jatkaakin vanhaan suuntaan. Koska tiedot pelaajan ohjauksesta lähetetään koko ajan, niin palvelimella objekti liikkuu juuri sinne, mihin pelaaja sitä ohjaa. Palvelin voi lähettää tiedot muille pelaajille vaikka vasta puolen sekuntin päästä, jolloin objektin liike jatkuu vanhaan suuntaan ja lopulta objekti hyppää uuteen paikkaan. Tämä aiheuttaa ikävää töksähtelyä ja vaikeuttaa tähtäämistä.

Objektin liike ja liikesuunnan muutokset saa näyttämään tasaiselta, jos tiedonsiirtokaista yritetään pitää vapaana ja tiedot uudesta suunnasta saadaan lähetettyä heti.

4.5 Algoritmi objektien tietokenttien päivittämiseen

Algoritmin suunnittelussa on otettava huomioon seuraavat asiat.

- Vain muuttuneet kentät lähetetään.
- Paketteja voi hävitä.
- Paketti voi tulla väärässä järjestyksessä, jolloin vanha paketti ei saa sotkea kenttää, jonka uudempi paketti on jo päivittänyt.

- Palvelin lähettää niiden kenttien päivitykset uudestaan, joihin asiakas ei ole vastannut.

Algoritmin pitäisi toimia niin, että palvelin lähettää kaikki muuttuneet kentät tietyin väliajoin. Jos lähetettyihin kenttiin ei tule vastausta, niin ne lähetetään uudestaan. Kentistä lähetetään aina sen hetkiset arvot, eikä enää vanhoja arvoja, jos lähetettyyn pakettiin ei tule vastausta.

Olen päätynyt ratkaisuun, jossa asiakkaalla on jokaista objektin kenttää varten paikka puskurissa. Verkkopelin lukusäie päivittää vastaanotetut tiedot tähän, josta ne käydään lukemassa pelin olioiden jäsenmuuttujiin update-funktiolla aina pelisilmukan alussa. Lisäksi jokaista kenttää varten on id toisessa puskurissa. Vastaanotettu kenttä kirjoitetaan kenttäpuskuriin vain jos vastaanotetun paketin id on suurempi kuin puskurissa oleva id. Eli jos vastaanotettu id on suurempi, niin vastaanotettu tieto ei ole vanhentunutta. Tällä vältytään väärässä järjestyksessä saapuneitten pakettien sotkemasta kenttiä.

Kenttien lähettämisessä käytetään bittiä, joka ilmaisee, onko kenttä muuttunut. Kun kenttää ei lähetetä, niin säästytään turhan tiedon lähettämiseltä. Jos kenttä on muuttunut niin lähetettävään tietovirtaan kirjoitetaan merkiksi bitti 1. Asiakas tietää vain mitä kenttiä objektilla on, mutta ei sitä mitkä kentät ovat muuttuneet. Objektin kentät käsitellään paketista käymällä silmukassa läpi kaikki objektin kentät. Kun asiakas lukee 1-bitin, niin se tietää, että seuraavaksi bittivirrassa on muuttuneen kentän tiedot. Jos kenttä ei ole muuttunut, niin palvelin kirjoittaa 1-bitin sijasta 0-bitin. 0-bitin kohdalla asiakas ei muuta objektin kentän tietoa mitenkään.

Asiakas lähettää jokaisesta paketista palvelimelle vastauksen, jossa on vastaanotetun paketin id. Palvelin tietää tästä, mitkä objektien kentät asiakkaalla on. Palvelimella on jokaista asiakkaalle päivitettyä objektia varten taulukko, jossa on id jokaista objektin kenttää varten. Kun objektin kenttä päivitetään asiakkaalle niin kenttää vastaavaan taulukon kohtaan kirjoitetaan paketin id, jossa kenttä lähetetään. Jos palvelimen lähettämään pakettiin ei tule vastausta, niin paketin id:tä vastaavat taulukon id:t muutetaan ykkösiksi, josta palvelin tietää seuraavan kerran objektin päivitystä lähetettäessä, että kenttä täytyy lähettää uudestaan vaikka se ei olisi muuttunut. Id:tä ei kuitenkaan muuteta ykköseksi jos kenttä on ehtinyt muuttua ja muutos on lähetetty uu-

dessa paketissa, koska myös taulukossa oleva id on sama kuin uuden paketin id. Tällä tavalla lähetetään aina vain objektin kenttien uusimmat tiedot ja jos objektin kenttä ei muutu, eikä paketti mene perille niin vasta sitten lähetetään kenttä uudestaan.

Tämä algoritmi ratkaisee kaikki ylempänä olevat neljä ongelmaa. Verkkopelin toteutukseen joutuu kuitenkin lisäämään muutakin logiikkaa, jotta uusien objektien päivittäminen toimisi. Objektin päivittämisessä lähetetään objektin id, jolla asiakas tietää, mihin objektiin vastaanotetut kentät pitää kirjoittaa ja kuinka monta ja minkä tyyppisiä kenttiä vastaanotetussa tiedossa on. Uuden objektin luonnissa välitetään myös objektin tyyppi, jonka avulla asiakas tietää, mitä kenttiä objektilla on ja osaa lukea niiden perusteella vastaanotetusta paketista oikeat tiedot.

Lisäksi objekteja voi hävitä pelistä (esim. pelaaja ottaa jonkin esineen), jolloin pitää varmistaa, että objektin poistamisen jälkeen jokin väärässä järjestyksessä vastaanotettu paketti ei päivitä jo poistettua objektia. Tästä kerrotaan suunnitteluosassa.

Tiedon lähettämisessä täytyy lähettää vain muutama paketti kerrallaan, jotta kaista ei tukkeudu. Palvelin lähettää paketin vasta, kun edellisiin paketteihin on tullut vastaus tai paketin lähettämisestä on kulunut tarpeeksi pitkä aika.

Objektien päivityksestä ja paketin rakenteesta on kerrottu enemmän luvussa 5.8.

5 TOTEUTUKSEN ARKKITEHTUURI

Tässä luvussa annetaan ensin yleiskuva verkkopelimoduulista luokkakaavion ja rajapintaluokkien avulla. Sitten kerrotaan, miten objektien rakenteen saa kerrottua rajapinnalle ja luokkien toiminnasta tarkemmin. Lopuksi kerrotaan, miten pakettien ja tietovirran lähetys sekä objektien kenttien synkronointi toteutetaan käytännössä.

5.1 Toimintaperiaate

Verkkopelimoduulissa on palvelinta ja asiakasta varten oma toiminnallisuus, mutta myös molemmille yhteisiä osia. Molemmilla on säie, joka käynniste-

tään verkkopelimoduulin käynnistysfunktiota kutsuttaessa. Tätä tarvitaan siihen, että paketit voidaan vastaanottaa heti ja lähettää niihin vastauksen mahdollisimman nopeasti, jolloin pakettien siirto on nopeaa. Paketteja lähetetään vain muutama kerrallaan, joten vastaanottajan on tärkeää vastata niihin mahdollisimman nopeasti, että voidaan lähettää seuraavat paketit.

Palvelin

Palvelimen tarkoitus on käsitellä pelaajan ohjauskäskyt, liikuttaa pelaajan hahmoa pelimaailmassa ja raportoida pelaajille pelimaailman tapahtumista, kuten pelaajien liikkeistä. Palvelimella käsitellään tekoäly ja liikutetaan kaikkia pelimaailman objekteja. Objektien muutokset lähetetään erikseen jokaiselle asiakkaalle. Asiakas ei voi päättää, millä tavalla jokin objekti liikkuu vaan kertoa ainoastaan, mitä haluaa oman pelaajan tekevän. Tämä estää pelaajia huijaamasta erilaisilla viritelmillä, jotka esim. liikuttavat pelaaja tavallista nopeammin.

Asiakas

Palvelin kertoo aisiakkaalle pelimaailmassa olevista objekteista niiden sijainnin ja mikä objekti on kyseessä. Asiakkaan tarkoitus on piirtää sen listassa olevat palvelimen lähettämät objektit ruudulle, ja lähettää palvelimelle tieto siitä, mihin suuntaan pelaaja katsoo ja mitä ohjausnappeja on painettuna.

Verkkopelimoduuli ei ole tarkoitettu vain yhdenlaiseen peliin vaan asiakas voi piirtää objektit ruudulle ja jättää kaiken muun palvelimen vastuulle. Asiakas voi myös liikuttaa objekteja itse, jotta liike olisi tasaisempaa, ja suorittaa tekoälyä, jolloin palvelin päivittää välillä objektien tilan oikeaksi.

Esim. Killing Enginessä (ks. luku 2) asiakas piirtää objektit ruudulle ja liikuttaa joitain objekteja ja animaatioita. Palvelin päivittää välillä asiakkaalle animaation oikean kohdan ja objektin sijainnin. Jos palvelin huomaa pelaajan olevan tarpeeksi lähellä jotain esinettä niin se lähettää asiakkaalle tietovirtana viestin, jossa kerrotaan minkä esineen pelaaja sai. Poimitusta esineestä kertovan objektin kentän arvoksi muutetaan true, ja palvelin päivittää kentän muutoksen asiakkaalle, josta asiakas tietää, että objektia ei enää näytetä ruudulla.

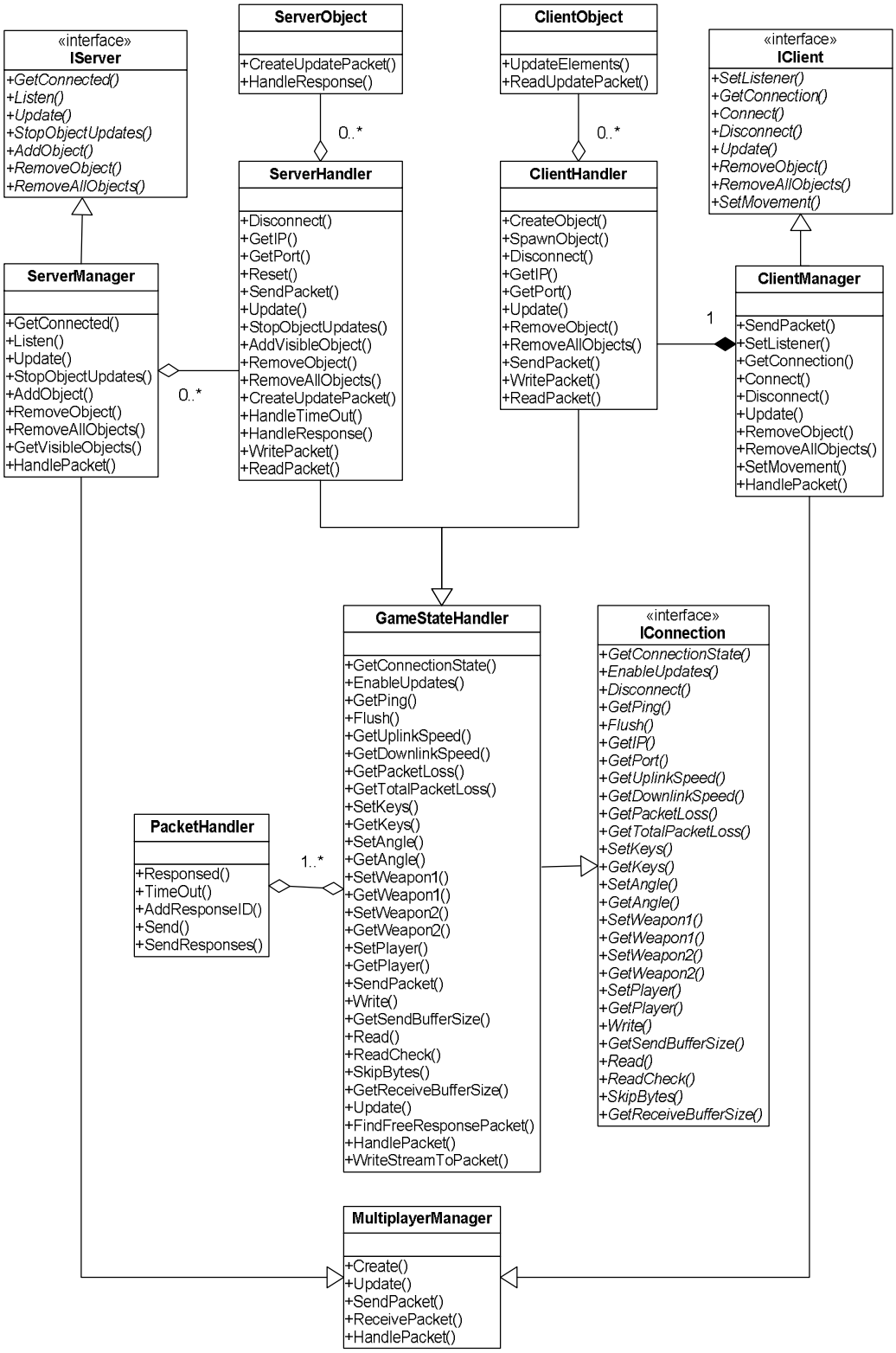
Kun pelaaja ampuu niin palvelin huomaa, että pelaajan ampumisnappi on painettuna ja lisää luotia tai muuta ammusta vastaavan objektin pelimaailmaan. Objekti päivitetään niille asiakkaille, jotka näkevät sen. Asiakkaat liikuttavat objektia ja piirtävät sen näytölle. Myös palvelin liikuttaa objektia ja tarkistaa tämän lisäksi osumat pelaajiin.

5.2 Luokkakaavio

Luokkakaaviossa MultiplayerManager-, ServerManager- ja ClientManager-luokat on tarkoitettu yhteyden hallintaan. Luokat hoitavat yhteyden muodostamisen ja pakettien vastaanottamisen.

Vastaanotetut paketit siirretään GameStateHandler-luokkaan, joka lukee paketista tietovirran ja hoitaa vastausten lähettämisen. Vastaanotetun paketin käsittely siirretään lopuksi ServerHandler- tai ClientHandler- luokkaan, joka lukee paketista objektien päivitykset ServerObject- tai ClientObject-luokan avulla.

Toteutusluokkien lisäksi on rajapintaluokat IServer, IClient ja IConnection, joilla verkkopelimoduuli erotetaan pelilogiikasta. Luokkakaavio on kuvassa 5.



Kuva 5. Verkkopelimoduulin luokkakaavio.

5.3 Rajapintaluokat

Rajapintaluokat erottavat verkkopelimoduulin toteutuksen itse pelistä ja niiden avulla verkkopelimoduulia on helppo käyttää.

5.3.1 IConnection

Tätä rajapintaluokkaa käytetään palvelimella ja asiakkaalla yhteyksien hallintaan. Osa funktioista on sellaisia, että niitä käytetään vain palvelimella tai asiakkaalla. Tietovirtaa varten tarkoitetut funktiot ovat käytössä molemmilla puolilla. Write-funktiolla lähetetty tieto voidaan lukea vastaanottajalla Read-funktiolla.

Palvelimella on jokaista asiakasta varten IConnection-olio, jolla asiakkaalle saa lähetettyä ja vastaanotettua tietovirtaa, sekä haettua tiedon pelaajan ohjauksesta. Näiden lisäksi rajapinnassa on funktioita, joilla saa tietoa yhteyden tilasta, kuten latenssin, tiedonsiirtonopeuden ja hävinneiden pakettien lukumäärän.

Kun asiakas katkaisee yhteyden, GetConnectionState-funktio palauttaa arvon DISCONNECTED. Jos yhteys katkeaa muusta syystä, funktio palauttaa arvon DROPPED. Jos funktio palauttaa jonkun näistä arvoista, sen jälkeen rajapinta vapautetaan uutta yhteyttä varten. Myös palvelin voi katkaista yhteyden Disconnect-funktiolla, jonka jälkeen rajapinta vapautetaan muuhun käyttöön.

Alimpana rajapinnan määrittelyssä ovat Read- ja Write-funktiot, joilla voi lähettää ja vastaanottaa tietovirtaa. Näiden avuksi ovat GetSendBufferSize-, GetReceiveBufferSize-, ReadCheck- ja SkipBytes-funktiot, joilla voi hakea tietoa lähetyspuskurin vapaasta tilasta, vastaanotettujen tavujen määrän ja lukea jotakin tarkistusta varten vastaanottopuskurista tietyn määrän tavuja jättäen kuitenkin tiedon puskuriin.

Rajapinnassa on myös funktioita, jolla saa tietoa yhteyden tilasta. GetPing-funktio palauttaa paketin kiertoajan millisekunteina, eli ajan, joka kuluu, kun paketti lähetetään ja siihen saadaan vastaus. GetUplinkSpeed-funktio palauttaa sekuntin aikana lähetetyn tietomäärän tavuina ja GetDownlinkSpeed-funktio palauttaa vastaanotettujen tavujen määrän sekuntin aikana. GetPacketLoss-funktio palauttaa hävinneiden pakettien lukumäärän sekuntissa ja GetTotalPacketLoss-funktio palauttaa hävinneiden pakettien määrän koko yhteyden aikana.

EnableUpdates–funktiolla asiakas voi estää palvelinta lähettämästä objektien päivityksiä. Kun asiakas kutsuu funktiota arvolla false, verkkopelimoduuli

lähettää palvelimelle komennon. Sen jälkeen palvelin ei enää lähetä objektien päivityksiä. Funktio palaa vasta, kun palvelimen lähettämä vastaus on saatu. Jos vastausta ei tule, funktio palauttaa false ja yhteys katkeaa. Kun funktiota kutsuu arvolla true, palvelin alkaa taas lähettää objektien päivityksiä.

EnableUpdates-funktiota tarvitaan esim. kentän latauksessa, jolloin päivitykset voivat sotkea objektien lataamisen. Kuvassa 6 on luokan määrittely.

```
class IConnection
{
public:
	enum ConnectionState
	{
	CONNECTED = 0,
	CONNECTING,
	DISCONNECTED,
	INTERRUPTED,
	DROPPED
	};

public:
	virtual ~IConnection(){};

	virtual ConnectionState GetConnectionState() = 0;
	virtual bool EnableUpdates(bool enable) = 0;
	virtual void Disconnect() = 0;
	virtual int GetPing() = 0;
	virtual bool Flush() = 0;
	virtual int GetIP() = 0;
	virtual int GetPort() = 0;

	virtual int GetUplinkSpeed() = 0;
	virtual int GetDownlinkSpeed() = 0;
	virtual float GetPacketLoss() = 0;
	virtual int GetTotalPacketLoss() = 0;

	virtual void SetKeys(int keys) = 0;
	virtual int GetKeys() = 0;
	virtual void SetAngle(float angle) = 0;
	virtual float GetAngle() = 0;
	virtual void SetWeapon1(int weapon1) = 0;
	virtual int GetWeapon1() = 0;
	virtual void SetWeapon2(int weapon2) = 0;
	virtual int GetWeapon2() = 0;
	virtual void SetPlayer(Object *object) = 0;
	virtual Object *GetPlayer() = 0;

	virtual bool Write(void *data, int len) = 0;
	virtual int GetSendBufferSize() = 0;
	virtual int Read(void *data, int maxlen) = 0;
	virtual int ReadCheck(void *data, int maxlen) = 0;
	virtual int ReadCheck(void *data, int start, int maxlen) = 0;
	virtual bool SkipBytes(int numBytes) = 0;
	virtual int GetReceiveBufferSize() = 0;
};
```

Kuva 6. Yhteysrajapinta.

5.3.2 IServer

IServer-rajapinnalla pelin palvelin voi luoda UDP-socketin, johon asiakkaat voivat luoda yhteyden IClient-rajapinnan avulla. Palvelin käynnistetään Listen-funktiolla.

Kun pelilogiikka luo uusia objekteja, se lisää synkronoitavat objektit verkkopelimoduuliin AddObject-funktiolla. Kun objekti häviää pelistä, se poistetaan RemoveObject-funktiolla ja verkkopelimoduuli välittää tästä viestin asiakkaille. Kaikki objektit voi poistaa RemoveAllObjects-funktiolla. Tämä kuitenkin välittää viestit pelaajille, jota ei pelin päätyttyä haluta enää tehdä. StopObjectUpdates-funktio tyhjentää kaikki objektien päivitykseen käytetyt tietorakenteet, jonka jälkeen asiakkaalle ei enää lähetetä mitään viestejä.

Pelilogiikka kutsuu jokaisessa iteraatiossa Update-funktiota, joka siirtää tietovirtaa ja päivittää objektien muutokset asiakkaille. Uuden yhteyden saa selville GetConnected-funktiolla, joka palauttaa IConnection-rajapinnan tai 0 jos uutta yhteyttä ei ole. Kuvassa 7 on luokan määrittely.

```
class IServer
{
public:
	virtual ~IServer(){};

	virtual IConnection *GetConnected() = 0;

	virtual bool Listen(int port) = 0;

	virtual void Update(float tick) = 0;
	virtual void StopObjectUpdates() = 0;

	virtual void AddObject(Object *object) = 0;
	virtual void RemoveObject(Object *object) = 0;
	virtual void RemoveAllObjects() = 0;
};
```

Kuva 7. Palvelinrajapinta.

5.3.3 IClient

Rajapinnalla asiakas luo UDP-socketin ja ottaa yhteyden palvelimeen. Ennen yhteyden luomista verkkopelimoduulille on kerrottava SetListener-funktiolla IClientListener-rajapinnan toteuttavan olion osoite. Sen jälkeen yhteys voidaan muodostaa Connect-funktiolla, joka lähettää palvelimelle yhteydenavauspyynnön. Jos palvelin vastaa tähän niin funktio palauttaa osoittimen IConnection-rajapintaan, jonka kautta yhteyttä voi hallita. Jos palvelin

ei vastaa pyyntöön niin funktio yrittää uudelleen ja palauttaa lopulta nollan jos yhteyttä ei saada muodostettua. Yhteyden voi katkaista Disconnect-funktiolla.

Pelilogiikka kutsuu jokaisessa iteraatiossa Update-funktiota, joka siirtää vastaanotettujen objektien kentät pelin omiin objekteihin ja suorittaa objektien liikkeiden ennustamisen. SetMovement-funktiolla kerrotaan verkkopelimoduulille ohjaukseen käytettävien nappien tila, pelaajan katselusuunta ja käytössä olevat aseet. Kuvassa 8 on luokan määrittely.

```
class IClient
{
public:
      virtual ~IClient(){};

      virtual void SetListener(IClientListener *listener) = 0;

      virtual IConnection *GetConnection() = 0;

      virtual IConnection *Connect(const wchar_t *address) = 0;
      virtual IConnection *Connect(const wchar_t *address,
                                   int port) = 0;
      virtual void Disconnect() = 0;

      virtual void Update(float tick) = 0;

      virtual void RemoveObject(Object *object) = 0;
      virtual void RemoveAllObjects() = 0;
      virtual void SetMovement(int keys, float angle, int weapon1,
                               int weapon2) = 0;
};
```

Kuva 8. Asiakasrajapinta.

5.3.4 IClientListener

Verkkopelimoduulin täytyy pystyä luomaan peliin uusia objekteja. Tarkoitusta varten on IClientListener-rajapinta, jonka jokin pelin luokka toteuttaa. Rajapinnan määrittely on kuvassa 9. IClient-rajapinnan SetListener-funktiolla kerrotaan verkkopelimoduulille osoitin rajapinnan toteuttavaan olioon.

Kun verkkopelimoduuli vastaanottaa uuden objektin, se kutsuu CreateObject-funktiota. Pelin funktion toteutus tarkistaa tyypin perusteella mikä objekti on kyseessä ja palauttaa sen tyyppisen uuden objektin. Objektia ei kuitenkaan vielä lisätä peliin, koska vastaanotto on eri säikeessä missä pelilogiikka on ja muuten pelin objektien käsittely saattaisi sotkeutua.

Kun pelilogiikka kutsuu IClient-rajapinnan Update-funktiota niin verkkopelimoduuli lisää uudet objektit peliin SpawnObject-funktiolla.

```
class IClientListener
{
public:
	virtual ~IClientListener(){};

	virtual Object *CreateObject(int type, int resid) = 0;
	virtual void SpawnObject(Object *object) = 0;
};
```

Kuva 9. Kuuntelijarajapinta, jolla luodaan uusia objekteja.

5.3.5 Object

Kaikki pelin objektit, jotka täytyy saada päivitettyä verkon yli, perivät Object-luokan. Verkkopelimoduuli osaa käsitellä rajapinnan kautta pelin objekteja. Kuvassa 10 on rajapinnan määrittely.

```
class Object
{
	friend class ServerObject;
	friend class ClientObject;
	friend class ServerManager;
	friend class ClientManager;
	friend class ServerHandler;

protected:
	typedef struct
	{
		int type, offset;
		bool once;
	} element_t;


	virtual element_t *GetElements() = 0;
	int id;

public:
	Object() { id = 0; }
	virtual ~Object(){};

	virtual int Type() = 0;
	virtual int ResourceId() = 0;
	virtual float GetX() = 0;
	virtual float GetY() = 0;
	virtual bool CanSee(float x, float y) = 0;
	virtual void Show() = 0;
	virtual void Hide() = 0;
	virtual void Move(float x, float y, float dx, float dy) = 0;
};
```

Kuva 10. Objektiluokka, jonka pelin objektiluokat perivät.

5.4 Objektin kenttien määritteleminen

Jokaisella objektilla täytyy olla lista tietokentistä, jotka objektilla on. Näin verkkopelimoduuli osaa hakea muistista oikeasta paikasta objektin kunkin

kentän. Verkkopelimoduuli tarkistaa onko kenttiä muuttunut ja lähettää muutokset asiakkaalle. Kenttien määrittelyyn käytetään makroja, jotka lisäävät objektin luokkaan määriteltyyn kenttätaulukkoon tiedot kenttien tyypistä. Taulukkoon tulee myös kenttien sijainti luokassa, joka saadaan C++:n offsetof-funktiolla.

- BEGIN_DEFINE_ELEMENTS(_class) aloittaa kenttien määrityksen ja lisää _class parametrin määräämään luokkaan __elements-taulukon.
- END_DEFINE_ELEMENTS lopettaa kenttien määrityksen.
- DEFINE_ELEM_BYTE(_class, var) lisää elementtitaulukkoon var-parametrin osoittaman char-tyyppisen muuttujan.
- DEFINE_ELEM_WORD(_class, var) lisää elementtitaulukkoon var-parametrin osoittaman short-tyyppisen muuttujan.
- DEFINE_ELEM_DWORD(_class, var) lisää elementtitaulukkoon var-parametrin osoittaman int-tyyppisen muuttujan.
- DEFINE_ELEM_FLOAT(_class, var) lisää elementtitaulukkoon var-parametrin osoittaman float-tyyppisen muuttujan.
- DEFINE_ELEM_BOOL(_class, var) lisää elementtitaulukkoon var-parametrin osoittaman bool-tyyppisen muuttujan.
- DEFINE_ELEM_INTERPOLATE_FLOAT(_class, var) lisää elementtitaulukkoon var-parametrin osoittaman float-tyyppisen muuttujan, jonka arvoa asiakas yrittää ennustaa eri ajanhetkillä sen mukaan, mihin suuntaan muuttujan arvo on muuttunut ja kuinka nopeasti.
- DEFINE_ELEM_INTERPOLATE_POSITION(_class, var_x, var_y, var_dx, var_dy) lisää taulukkoon var-parametrien osoittaman sijainnin ja suunnan. X- ja y-parametreja yritetään ennustaa sen mukaan, mihin suuntaan objekti on liikkumassa dx- ja dy-parametrien mukaan.
- DECLARE_MP_OBJECT määrittää luokkaan __elements-taulukon ja GetElements-funktion, joka palauttaa osoitteen siihen.

Esimerkiksi pelaajan objektit määritellään seuraavasti. Otsikkotiedoston luokkamäärittelyyn tulee DECLARE_MP_OBJECT. Tämä lisää luokkaan staattisen __elements–taulukon johon määritetään toteutustiedostossa muilla makroilla objektin kenttien tyypit ja sijainnit. Lisäksi luokkaan lisätään GetElements-funktion toteutus, jolla verkkopelimoduuli hakee kenttien määrittelyn Object-rajapintaluokan kautta. Kuvassa 11 on esimerkki, miten luokka määritellään.

```
class Player : public Object
{
     DECLARE_MP_OBJECT

     float dx, dy;
     int team;
     short score;
     unsigned short flags;
     ...
```

Kuva 11. Synkronoitavan objektin määrittely.

Toteutustiedostoon määritetään makroilla jokainen objektin kenttä kuvan 12 tavalla.

```
BEGIN_DEFINE_ELEMENTS(Player)
DEFINE_ELEM_INTERPOLATE_POSITION(Player, x, y, dx, dy)
DEFINE_ELEM_FLOAT(Player, angle)

DEFINE_ELEM_DWORD(Player, team)
DEFINE_ELEM_WORD(Player, flags)
DEFINE_ELEM_FLOAT(Player, health)
DEFINE_ELEM_DWORD(Player, weapon)
DEFINE_ELEM_FLOAT(Player, dispersion)
DEFINE_ELEM_WORD(Player, score)
END_DEFINE_ELEMENTS
```

Kuva 12. Objektiluokan kenttien määrittely.

5.5 Toteutusluokat

Tässä luvussa selitetään verkkopelimoduulin toteutusluokkien toiminta. Luokkien rakenteen näkee luvun 5.1 luokkakaaviosta, ja nimet ovat tässä samoja.

MultiplayerManager

MultiplayerManager-luokka käsittelee pakettien lähettämisen ja vastaanottamisen. Käynnistysfunktiolla luodaan säie, joka vastaanottaa kaiken tiedon kuuntelemalla UDP-sockettia. Vastaanotetut paketit välitetään virtuaalifunktiolla ylemmälle luokalle, ServerManager:lle tai ClientManager:lle.

ServerManager

ServerManager-luokka käsittelee MultiplayerManager-luokan vastaanottamat paketit ja jos paketissa on yhteydenmuodostuspyyntö niin palvelin alustaa asiakasta varten ServerHandler-luokan tietorakenteen ja lähettää asiak-

kaalle vastauksen yhteyden luonnin onnistumisesta. Tietorakenteeseen tallennetaan mm. IP-osoite ja portti, josta paketti on tullut. Jos palvelimella on niin paljon asiakkaita, että tietorakenteita ei ole vapaana niin asiakkaalle lähetetään paketti, jossa kerrotaan, että palvelin on täynnä. Jos palvelimen tietorakenteista löytyy sama IP-osoite ja portti, josta paketti on tullut niin paketti siirretään suoraan ServerHandler-luokan käsiteltäväksi. Tämä luokka toteuttaa myös IServer-rajapinnan, jonka kautta uusia asiakasyhteyksiä voi hyväksyä ja lisätä tai poistaa objekteja verkkopelimoduulista.

ClientManager

ClientManager-luokka toteuttaa IClient-rajapinnan, jonka kautta asiakas voi muodostaa yhteyden palvelimeen ja välittää tiedon pelaajan ohjauksesta. Luokan ainut tehtävä on lähettää yhteydenmuodostuspaketti ja käsitellä palvelimen vastaus. Kun yhteys on muodostettu, niin kaikki vastaanotettu tieto välitetään suoraan ClientHandler-oliolle, joita on määritelty yksi ClientManager-luokassa.

GameStateHandler

GameStateHandler on verkkopelimoduulin tärkein luokka, joka hoitaa yhteyden muodostuksen jälkeen pakettien lähettämisen oikeaan aikaan niin, että yhteys ei tukkeutuisi ja vastausten lähettämisen vastaanotetuista paketeista mahdollisimman nopeasti. Luokka käsittelee myös tietovirran lähettämisen ja vastaanottamisen. Tietovirtaa (ks. luku 5.7) lähetetään aina tietty enimmäismäärä objektien tilatietojen lisäksi. Jos objekteja ei ole päivitettävänä, niin koko paketti voidaan käyttää tietovirran lähettämiseen. Jos lähetettyyn pakettiin ei tule vastausta tarpeeksi nopeasti, paketin objektiosa muodostetaan uudestaan, ja paketti lähetetään uudestaan tietovirtaosan säilyessä muuttumattomana.

ServerHandler

ServerHandler-luokka toteuttaa ReadPacket- ja WritePacket-funktiot, joilla palvelin lukee vastaanotetusta paketista pelaajan lähettämät ohjaustiedot ja lisää lähetettävään pakettiin objektien päivitykset. ServerManager-luokassa on lista, jossa on ServerHandler-olio jokaista asiakasta kohti.

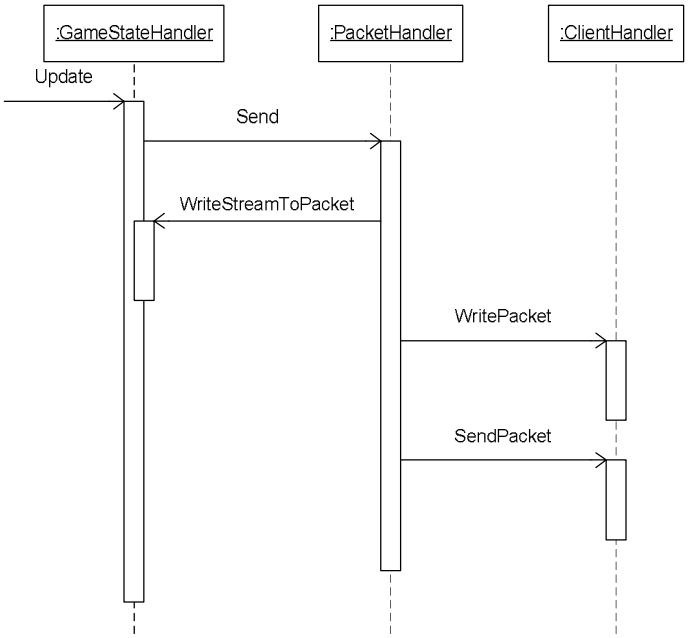
Pakettia lähetettäessä PacketHandler-luokan Send-funktio lisää pakettiin ensin vastaukset ja tietovirran. Sen jälkeen se kutsuu GameStateHandler-luokan WritePacket-funktiota, jonka ServerHandler-luokka toteuttaa. Funktio etsii kaikki uudet pelaajalle näkyvät objektit ja lisää ne ServerHandler-luokan listaan. Kaikille listan olioille kutsutaan ServerObject-luokan CreateUpdate-Packet-funktiota, joka lisää objektin päivityksen pakettiin.

Asiakkaan lähettämä paketti käsitellään GameStateHandler-luokan Handle-Packet-funktiossa, joka lukee paketista ensin vastaukset ja tietovirran. Sitten se kutsuu ReadPacket-funktiota, jonka ServerHandler-luokka toteuttaa. Funktio lukee paketista pelaajan lähettämän ohjaustiedon ja lisää nappien tiedot ja pelaajan katselusuunnan ServerHandler-luokan muuttujiin.

ClientHandler

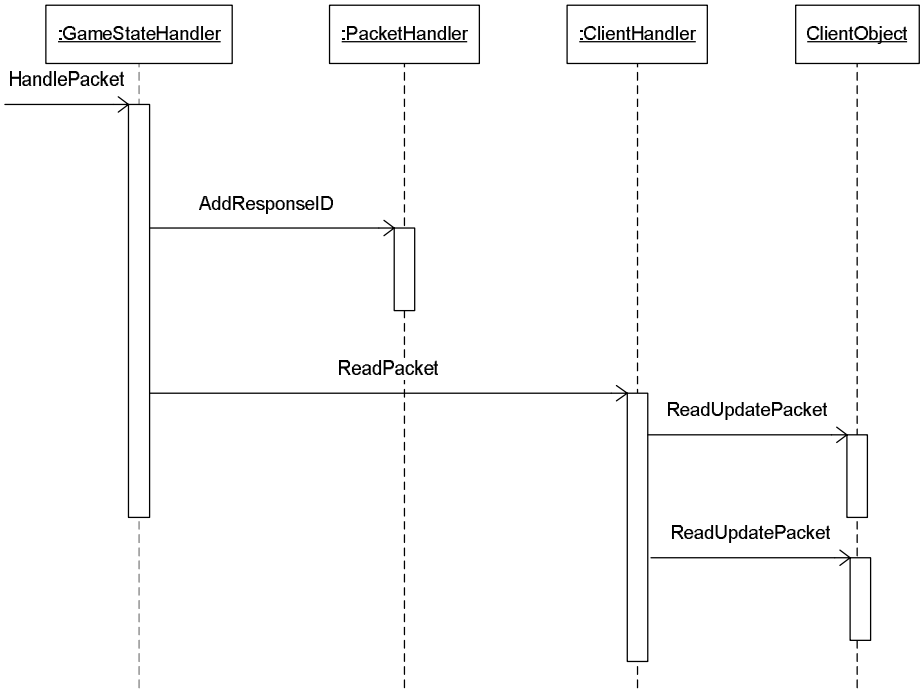
ClientHandler-luokka toteuttaa ReadPacket- ja WritePacket-funktiot, joilla asiakas välittää ohjaustiedot palvelimelle ja lukee objektien päivitykset vastaanotetusta paketista. Luokasta on yksi ilmentymä ClientManager-luokassa.

Kun asiakas lähettää paketin niin GameStateHandler-luokan Update-funktio kutsuu PacketHandler-luokan Send-funktiota, joka muodostaa paketin. Tietovirran pakettiin lisäämisen jälkeen Send-funktio kutsuu GameStateHandler-luokan WritePacket-funktiota, jonka ClientHandler-luokka toteuttaa. ClientHandler-luokan WritePacket-funktio lisää pakettiin ohjaukseen käytettävien nappien tiedot ja pelaajan katselusuunnan. Luokkien funktiokutsut pakettia lähetettäessä ovat kuvassa 13.



Kuva 13. Paketin lähettäminen.

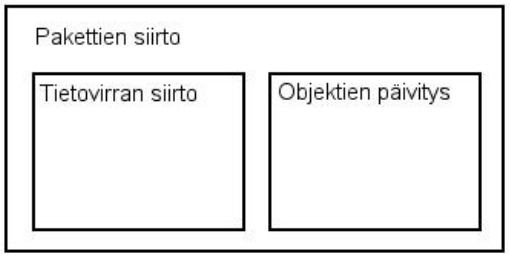
Paketin vastaanotossa GameStateHandler-luokan HandlePacket-funktio käsittelee ensin paketista vastaukset ja tietovirran ja kutsuu sen jälkeen ReadPacket-funktiota, jonka ClientHandler-luokka toteuttaa. Tämä lukee paketista objektin id:itä niin kauan, kun id ei ole nolla. Ensin id:llä etsitään ClientObject-oliota, jolla on sama id. Jos olio löytyy niin kutsutaan ClientObject-luokan ReadUpdatePacket-funktiota, joka lukee objektin päivityksen paketista. Jos oliota ei löydy, luodaan uusi objekti id:llä ja vasta sen jälkeen luetaan paketista päivitys luotuun objektiin. Luokkien funktiokutsut paketin vastaanotossa ovat kuvassa 14.



Kuva 14. Paketin vastaanotto.

5.6 Pakettien välitys

Tässä luvussa kerrotaan tarkemmin edellisissä luvuissa esitellyn ohjelmistoarkkitehtuurin toiminnasta pakettien siirrossa. Verkkopelimoduulin pakettien käsittelyn voi jakaa eri kerroksiin, jotka ovat kuvassa 15.



Kuva 15.Verkkopelimoduulin eri kerrokset.

Verkkopelimoduuli lähettää paketteja tietyin väliajoin ja jokaiseen pakettiin sisällytetään puolet tietovirtaa ja jäljelle jäävä osa objektien päivityksiä. GameStateHandler-luokka ja sen apuluokka PacketHandler huolehtivat pakettien siirrosta ja tietovirran lisäämisestä pakettiin. GameStateHandler toteuttaa IConnection-rajapinnan Read, Write ja muut tietovirran lähetykseen tarkoitetut funktiot.

Pakettien lähettämistä varten pelilogiikan on kutsuttava jokaisessa iteraatiossa IClient- tai IServer-rajapinnan Update-funktiota, joka tarkistaa sopivin aikavälein, onko lähetyslistassa vapaita tietorakenteita paketin lähettämistä varten. Tietorakenteeseen tallennetaan paketin sisältämä tietovirta ja paketin id. Jos paketti ei mene perille, niin tietorakenteessa oleva laskuri ylittää rajan, ja paketti lähetetään uudestaan. Uudessa paketissa lähetetään kadonneen paketin tietovirta ja objektiosa muodostetaan uudestaan. Lähetyslistaa tarvitaan myös siihen, että paketteja osattaisiin lähettää vain tietty määrä kerrallaan.

Jos pakettilistasta löytyy vapaa tietorakenne niin kutsutaan PacketHandler-luokan Send-funktiota, joka muodostaa paketin. Jokaiselle paketille annetaan yksilöivä id, joka kasvaa aina yhdellä, kunnes palaa lopulta ykköseen. Jos lähetetty paketti ei mene perille, niin uudelleen lähetettäväkin paketti saa uuden id:n. Tämä estää sen, että jos kadonneeksi luultuun pakettiin tuleekin vastaus, niin uudenkin paketin luullaan menneen perille.

GameStateHandler-luokassa on virtuaaliset ReadPacket- ja WritePacket-funktiot, jotka ServerHandler- ja ClientHandler-luokat toteuttavat. PacketHandler-luokan Send-funktio kirjoittaa pakettia muodostettaessa ensin pakettiin tietovirtaa ja kutsuu sitten WritePacket-funktiota. Palvelin kirjoittaa tässä funktiossa pakettiin objektien päivityksiä tärkeysjärjestyksessä niin paljon kuin pakettiin mahtuu. Asiakas puolestaan kirjoittaa pakettiin ohjaustiedot.

ServerManager- ja ClientManager-luokat vastaanottavat lukusäikeessä paketit ja kutsuvat GameStateHandler-luokan HandlePacket-funktiota. Tämä etsii lähetyslistasta paketin, joka ollaan seuraavaksi lähettämässä ja lisää vastaanotetun paketin id:n saman luokan listaan. Kun paketti seuraavan kerran lähetetään, niin PacketHandler-luokan Send-funktio lisää vastaus-id:t lähetettävään pakettiin. Kun vastaus on mennyt perille, niin sen vastaanottaja tietää, että sen lähettämä paketti on mennyt perille ja lähetyslistan tietorakenne vapautetaan uutta pakettia varten.

Kun vastaukset on käsitelty niin paketista luetaan tietovirtaosa, joka lisätään GameStateHandler-luokan listaan. Lopuksi kutsutaan ReadPacket-funktiota, jonka ClientHandler-luokka toteuttaa asiakkaalla ja jossa objektien päivityk-

set luetaan paketista. Palvelimella kutsutaan ServerHandler-luokan ReadPacket-funktiota, joka lukee paketista pelaajan ohjaustiedot.

5.7 Tietovirran välitys

Pelissä on pystyttävä lähettämään muutakin kuin pelkkiä objektien tietoja. Esim. pelin aloituksessa täytyy välittää palvelimelta asiakkaalle pelattavan kentän tiedostonimi ja muita pelin tietoja. Pelaajien lähettämät viestit lähetetään tietovirtana palvelimen kautta muille pelaajille ja esineiden, aseiden ja ammusten poimimisesta lähetetään pelaajalle viesti, jotta peli osaa näyttää oikean ilmoituksen tästä ja lisätä poimitun tavaran pelin tietorakenteisiin. Tietovirran siirtoa varten GameStateHandler-luokassa on IConnection-rajapinnan Read-, Write- ja muiden tietovirran käsittelyfunktioiden toteutukset.

Kun uusi paketti muodostetaan, niin PacketHandler-luokan Send-funktio hakee GameStateHandler-olion puskurista tietoa, joka lisätään paketin alkuun. Jos lähetettyyn pakettiin ei tule vastausta tarpeeksi nopeasti niin paketin oletetaan kadonneen ja se lähetetään uudestaan. Asiakkaan on kuitenkin huomioitava se, että paketti voi tulla useamman kerran joko sen takia, että IP-paketit voivat monistua, tai ensimmäinen paketti on voinut tulla perille viivellä.

Tämän takia asiakas pitää muistissa sen paketin id:tä, joka on käsiteltävä seuraavaksi. Jos vastaanotetun paketin id on pienempi niin pakettia ei käsitellä. Paketin id voi olla myös suurempi, jolloin paketin tietovirtaosa tallennetaan muistiin ja käsitellään sen jälkeen, kun oikea paketti on vastaanotettu.

5.8 Objektien päivitys

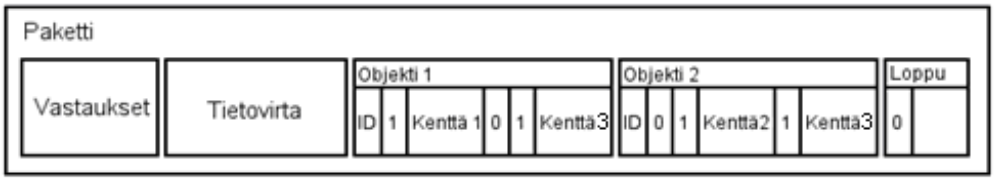
Objektien päivityksessä palvelimella tapahtuvien objektien tietokenttien muutokset välitetään asiakkaalle. Nämä muutokset lisätään pakettiin tietovirran jälkeen, kun PacketHandler-luokan Send-funktio kutsuu lopuksi ServerHandler-luokan WritePacket-funktiota. Tämä etsii ne objektit, jotka on tärkeintä saada lähetettyä, ja kutsuu jokaisen objektin CreateUpdatePacket-funktiota, joka on ServerObject-luokassa. Funktio kirjoittaa pakettiin kaikkien muuttuneitten kenttien tiedot. Objektit ovat paketissa kuvan 16 osoittamassa muodossa.

Objektien kentille voi myös määrittää prioriteetin ja korkeimman prioriteetin kentät lähetetään ensin. PacketHandler-luokaan on toteutettu algoritmi, joka osaa hakea ne objektit, joissa on tärkeimpiä kenttiä päivitettävänä. Myös ne objektit päivitetään, joilla on pienemmän prioriteetin kenttiä, mutta joita ei ole päivitetty vähään aikaan.

Palvelimella on jokaista asiakasta kohti ServerHandler-olio, jossa on lista objekteista, jotka näkyvät tai ovat olleet näkyvissä asiakkaalle. Kun palvelin lisää objektit pakettiin niin ServerHandler-luokan CreateUpdatePacket-funktio merkitseen ensin kaikki ServerHandler-luokan listassa olevat objektit näkymättömiksi. Sen jälkeen se kutsuu ServerManager-luokan GetVisibleObjects-funktiota, joka etsii kaikki asiakkaalle näkyvät objektit luokassa olevasta, kaikki palvelimen objektit sisältävästä listasta. Näkyvät objektit lisätään ServerHandler-luokan listaan sen ServerHandler-luokan AddVisibleObject-funktiolla.

Kun objektin tilaa päivitetään, niin ServerObject-luokan CreateUpdate-Packet-funktio osaa käsitellä objektin näkyvyyden sen perusteella, onko objekti merkitty näkyväksi vai näkymättömäksi.

Pelilogiikka voi lisätä ServerManager-luokan listaan IServer-rajapinnan AddObject-funktiolla objekteja, jotka halutaan synkronoida asiakkaille.



Kuva 16. Pakettien muoto.

5.8.1 Objektien tilan päivitys

Objektien tilan päivityksellä tarkoitetaan uudesta objektista kertomista, objektin piilottamista ja poistamista. Objektien tiedot kirjoitetaan ServerObject-luokan CreateUpdatePacket-funktiossa, jota ServerHandler-luokan Write-Packet-funktio kutsuu jokaiselle objektille. CreateUpdatePacket-funktio tarkistaa ensin, onko objektin tila uusi, näkymättömissä vai poistettu pelistä. Jos objektin tila ei ole mikään näistä, niin kyseessä on tavallinen kenttien päivitys ja siirrytään suoraan tietokenttien päivittämiseen.

Objektipäivitys on paketissa kuvan 16 osoittamassa muodossa. Esimerkissä molemmilla objekteilla on kolme kenttää, joista kaksi on muuttunut. Pakettiin lisätään aina ensin 32 bitin id, joka kertoo mihin objektiin päivitys on. Sen jälkeen on 2 bitin koodi, joka kertoo päivityksen tyypin. Jos objektin tila on uusi niin koodina käytetään ykköstä, jonka perusteella asiakas tietää, että kyseessä on uusi objekti. Uudesta objektista täytyy kertoa, koska objekteja on erilaisia ja niillä on erilaisia kenttiä. Muuten asiakas ei tiedä, kuinka monta ja minkätyyppisiä kenttiä paketissa on.

Uuden objektin koodin jälkeen pakettiin lisätään objektin tyyppi, jonka perusteella asiakas osaa luoda objektia varten oikean luokan ilmentymän. Ilmentymän kautta se saa Object-rajapinnan GetElements-funktiolla selville objektin elementtitaulukon, josta se näkee mitä kenttiä objektilla on. Objektin tyypin voisi kertoa myös jokaisen tavallisen päivityksen mukana, mutta se veisi enemmän tilaa.

Kun asiakas vastaanottaa objektien päivityksen, se hakee paketista ensin objektin id:n ja etsii sen perusteella listasta kyseisen objektin. Koska UDP-protokollaa käytettäessä paketeja saatetaan vastaanottaa eri järjestyksessä, kuin niitä on lähetetty, niin objektia ei välttämättä löydykkään listasta. Tällöin paketti tallennetaan listaan, josta se käsitellään uudelleen oikean paketin käsittelyn jälkeen. Objektin id:n jälkeen paketista haetaan päivityksen tyyppi ja käsitellään paketti sen mukaan.

Kun objekti häviää pelistä, se poistetaan palvelimelta IServer-rajapinnan RemoveObject-funktiolla. Verkkopelimoduuli huomaa seuraavassa Update-funktion kutsussa, että objekti on poistettu ja lähettää objektin päivityksessä asiakkaalle tiedon tästä. Palvelin pitää poistetun objektin tietorakennetta vielä muistissa sen aikaa, kunnes asiakas on vastannut lähetettyyn pakettiin.

Myös asiakas pitää objektia varten varattua tietorakennetta muistissa hetken. Jos jokin paketti sattuu vielä sisältämään päivityksen kyseiseen objektiin niin pakettia ei osata käsitellä jos objektia ei löydy, eikä ole tietoa siitä, mitä kenttiä paketissa on.

5.8.2 Objektien kenttien päivitys

Objektien kenttien muutokset lisätään lähetettään pakettiin ServerObject-luokan CreateUpatePacket-funktiossa sen jälkeen, kun objektin tilatieto on

lisätty pakettiin, jos objekti on uusi ja pelaajalle näkyvissä. Funktio käy läpi kaikki objektin tietokentät ja vertaa tietoa ServerObject-luokan puskurissa olevaan vanhaan tietoon.

Jos kenttä on muuttunut tai lähetettyä pakettia ei ole saatu perille, pakettiin lisätään tästä merkiksi ykkösbitti ja sen perään muuttuneen kentän tiedot. Jos kenttä ei ole muuttunut, niin pakettiin lisätään pelkkä nollabitti. Kentät ovat paketissa kuvan 16 osoittamassa muodossa.

Paketin perillemenon seuraamista varten ServerObject-luokassa on taulukko, jossa on jokaista objektin kenttää varten sen paketin id, jolla kenttä on viimeeksi lähetetty. Jos paketti ei mene perille niin ServerHandler-luokan HandleTimeOut-funktio asettaa taulukon ne id:t ykkösiksi, jotka ovat samoja kuin hävinneen paketin id. Jos taulukossa on ykkönen, niin kenttä lisätään pakettiin, vaikka se ei olisi muuttunut.

5.8.3 Liikkeen pehmennys

Pelin tiettyjen objektien halutaan liikkuvan tasaisesti. Näitä ovat Esim. pelaajat ja ajoneuvot. Esim. esineiden siirtely on paljon hitaampaa, joten siinä voi vain lähettää harvemmin objektin sijainnin. Liikkeen pehmennystä tarvitaan, koska objektien sijaintia ei voi päivittää koko ajan rajoitetun kaistanleveyden takia.

Objektin tietokenttä, jonka muutoksia verkkopelimoduuli yrittää ennustaa dead-reckoning algoritmilla, määritellään objektiin DEFINE_ELEM_INTERPOLATE_POSITION-makrolla. Verkkopelimoduuli käsittelee sijainnin ja liikesuunnan X- ja Y-komponentit erikseen.

Jos objektin liikesuunta on muuttunut palvelimella, niin palvelin päivittää objektin sijainnin ja liikesuunnan asiakkaalle mahdollisimman nopeasti. Jos pelkkä sijainti muuttuu niin palvelin ei päivitä muutosta asiakkaalle heti vaan odottaa hetken. Tällä aikaa asiakas yrittää ennustaa sijaintia ja näin säästetään tiedonsiirtoa. Vaikka sijainti pitäisi päivittää, voidaan sitä lykätä vielä hetki jos jotain korkeamman prioriteetin kenttiä on lähetettävänä, kuten objektin suunnan muutos.

Kun asiakas vastaanottaa sijainnin tai suunnan muutoksen se tallentaa nämä ClientObject-luokan puskuriin ja merkitsee lisäksi puskuriin, että sijainti on muuttunut.

Pelin koodi kutsuu jokaisella iteraatiolla IClient-rajapinnan kautta ClientHandler-luokan Update-funktiota, joka kutsuu jokaisen objektin UpdateElements-funktiota. Jos puskurissa oleva sijainti on muuttunut niin objektin sijainniksi muutetaan tämä. Myös peli-objektin suunnaksi päivitetään puskurissa oleva suunta. Tämän jälkeen UpdateElements liikuttaa aina objektia siihen suuntaan, minkä palvelin on kertonut ja sen verran, kun aikaa on kulunut.

Pelin täytyy tarkistaa objektia liikutettaessa, että se ei mene seinien läpi. Ilman liikkeen pehmennystä tätä ei tarvitsisi, koska silloin ainoastaan palvelin liikuttaa objekteja. Tarkistusta varten Object-rajapinnassa on Move-funktio, jonka pelin objektiluokka toteuttaa. Tämä aiheuttaa kuitenkin uuden ongelman, koska palvelin ja asiakas eivät liikuta objektia samaa liikerataa pitkin. Objekti voi esim. joutua asiakkaalla talon sisälle ja palvelimella jäädä ulkopuolelle. Lopulta objekti hyppää pelaajalla takaisin talon ulkopuolelle, kun palvelin päivittää objektin sijainnin kokonaan.

Tähän auttaa se, että jos objektin suunta muuttuu palvelimella niin tieto siitä pitää saada heti asiakkaille, jolloin asiakas ei ehdi jatkaa objektin liikettä vanhaan suuntaan liian kauan. Oleellista siis on, että tietoa ei tarvitse lähettää jos liikesuunta ei ole muuttunut. Näin säästetään kaistaa ja liikesuunnan muuttuessa se saadaan päivitettyä mahdollisimman nopeasti asiakkaille. Jos yhteydessä on suuri latenssi niin objektien tökkimistä voi kuitenkin esiintyä.

6 YHTEENVETO

Tässä pinnäytetyässä selvitettiin internetin aiheuttamia ongelmia verkkopelin tiedonsiirtoon. Alussa kerrottiin, mitä verkkopelit ovat ja minkälaiseen peliin työn verkkopelimoduli sopii. Sitten kerrottiin internetin toiminnasta ja pakettien siirtämisestä paikasta toiseen internetissä. Lopuksi kehitettiin ratkaisu, jolla verkkopelin saa toimimaan internetin rajoitukset huomioon ottaen.

Työn tuloksena syntyi verkkopelimoduuli, joka hallitsee internet-yhteyksiin liittyvät ongelmat: pakettien katoamisen ja pakettien järjestyksen sotkeutumisen. Moduulin toteutus toimii niin hyvin, että peliä pystyy pelaamaan, mutta joissain tilanteissa jotkin objektit eivät tule näkyviin pelaajalla. Esim. objekti voidaan pakettien järjestyksen sotkeutumisen takia poistaa ennen kuin asiakas on vastaanottanut tiedon uudesta objektista.

Verkkopelimoduliin on tehtävä vielä monia parannuksia, jotta kadonneet ja väärässä järjestyksessä vastaanotetut paketit hallutaan kunnolla. Esim. tieto objektin poistamisesta voi tulla ennen, kun tieto samasta uudesta objektista on vastaanotettu.

VIITELUETTELO

[1] Caldwell, Nicholas. Defeating Lag With Cubic Splines [verkkodokumentti]. 2000 [viitattu 25.4.2008]. Saatavissa: http://www.gamedev.net/reference/articles/article914.asp.

[2] Aronson, Jesse. Dead Reckoning: Latency Hiding for Networked Games [verkkodokumentti]. 19.9.1997 [viitattu 25.4.2008]. Saatavissa: http://www.gamasutra.com/features/19970919/aronson_01.htm.

[3] Postel, J. User Datagram Protocol [verkkodokumentti]. 28.8.1980 [viitattu 25.4.2008]. Saatavissa: http://www.ietf.org/rfc/rfc0768.txt.

[4] Network Working Group. TCP Congestion Control [verkkodokumentti]. 1999 [viitattu 25.4.2008]. Saatavissa: http://www.ietf.org/rfc/rfc2581.txt.

[5] Microsoft. Winsock Reference [verkkosivusto]. 27.3.2008 [viitattu 25.4.2008]. Saatavissa: http://msdn2.microsoft.com/en-us/library/ms741416%28VS.85%29.aspx.