Justus Koponen

AUTORITAARINEN PALVELINLOGIIKKA REAALIAIKAISELLE MONINPELILLE

Opinnäytetyö

Ohjelmistotekniikan ko.

Kesäkuu 2015

KYAMK
University of Applied Sciences

Tekijä/Tekijät	Tutkinto	Aika
Justus Koponen	Insinööri	Kesäkuu 2015

Opinnäytetyön nimi

Autoritaarinen palvelinlogiikka reaaliaikaiselle moninpelille

31 sivua
1 liitesivu

Toimeksiantaja

Kymenlaakson ammattikorkeakoulu

Ohjaaja

Yliopettaja Paula Posio

Tiivistelmä

Opinnäytetyön tavoitteena oli luoda nopeatempoinen, reaaliaikainen moninpeli selaimeen käyttäen Node.js- sekä Socket.io-kirjastoja.

Opinnäytetyö lähti alkuun kokeilemalla ja ideoimalla eri tyylejä toteuttaa moninpeli. Lähtökohtina oli kuitenkin tehdä nopeatempoinen ja pelimekaniikoiltaan yksinkertainen peli, jota on vaivaton ruveta kaveriporukalla pelaamaan. Itse työ kulki eteenpäin ilman suurempia ongelmia. Opinnäytetyön tuloksena syntyi lähellä alun perin kaavailemaani ideaa oleva peli, johon on helppo liittyä ja jonka parissa on koukuttavaa pelata tuttavien kanssa.

Kaikkia käyttämiäni kirjastoja ja teknologioita käytin ensimmäistä kertaa, joten vaikka opettelu hidasti itse projektin toteutusta, oli projekti erittäin opettavainen kaikin puolin. Lopputuloksena havaitsin, että peliä varten toteutettava palvelinlogiikka kannattaa harkita käyttötapauksen, käyttäjäkunnan sekä alustan mukaan.

Asiasanat
javascript, html, moninpelit, palvelimet

Author (authors)	Degree	Time
Justus Koponen	Bachelor of Engineering	June 2015

Thesis Title

Authoritative Server Logic For a Real-Time Multiplayer Game

31 pages
1 page of appendices

Commissioned by

Kymenlaakso University of Applied Sciences

Supervisor

Paula Posio, Principal Lecturer

Abstract

The Subject of this thesis was to create a fast paced, real-time multiplayer game using Node.js and Socket.io.

The goal was to create a fast paced and mechanically simple game that's easy to jump in and out from. The project started off by experimenting with different ideas for a game and different ways to create the multiplayer architecture. After getting started the project progressed without much problems and the result was close to what I originally envisioned.

All the tools and libraries used in the project were new to me so despite learning everything while doing the project slowed things down a bit, it was naturally a massive learning experience. The biggest takeaway for me was that one should always plan their server logic with the use case, user base and the platform in mind.

Keywords

javascript, html, multiplayer, servers

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on toteuttaa reaaliaikainen moninpeli. Lähtökohtana ideoinnissa oli, että halusin tehdä nopeatempoisen pelin, johon on helppo liittyä ja lähteä pois, ja joka ei vaadi suurempaa ajallista investointia. Olen huomannut, että hektiset, nopeatempoiset, mekaniikoiltaan yksinkertaiset pelit aiheuttavat huvittavia tilanteita kaveriporukalla pelattaessa, joten tähtäsin samanlaiseen alkuasetelmaan.

Alustaksi valitsin verkkoselaimen, sillä käytännössä jokaisella on sellainen koneellaan, joten pelin voi aloittaa kaverin jakaman linkin klikkauksella. Selain oli mielenkiintoinen kehitysalusta, sillä pelin suorituskyky saattoi rampautua pienestäkin asiasta ja pelin toiminta vaihteli eri selainten välillä. Lisäksi eri kirjastojen ja kehitystyökalujen määrä selaimille on häikäisevä. Yritin kuitenkin pitää pelin mahdollisimman yksinkertaisena.

Palvelimelle valitsin Node.js-sovellusalustan, jota olen jo pitkään halunnut käyttää ja opiskella. Nodella työskentely oli jouhevaa, sillä saman ohjelmointikielen pyöriminen sekä selaimessa että palvelimella takaa niiden saumattoman yhteistyön.

2 TYÖKALUT JA TEKNOLOGIAT

2.1 Node.js

Node.js on avoimeen lähdekoodiin perustuva, verkko- ja palvelinsovelluksia varten toteutettu sovellusalusta. C-kielellä kirjoitettu ohjelmisto käyttää Googlen V8 JavaScript-moottoria koodin ajoon. Jatkossa käytän Node.js:stä yksinkertaisesti nimitystä Node. (Nodejs.org 2015.)

Node tarjoaa estämättömän I/O-mallin, joka käytännössä tarkoittaa sitä, ettei palvelin missään vaiheessa seisahdu ajaessaan koodia ja näin ollen estä muun koodin suoritusta odottaessaan aiempien komentojen loppuun pääsyä. (Souza 2014.)

Valitsin alustan projektia varten lähinnä oppimismielessä ja huomasinkin pian, kuinka kätevää on hyödyntää samaa koodipohjaa sekä selaimessa että palvelimella.

2.2 Socket.IO

Socket.IO on JavaScript-kirjasto, joka mahdollistaa selaimen ja palvelimen välisen kaksisuuntaisen, reaaliaikaisen kommunikoinnin. Se tarjoaa helppokäyttöisen implementaation WebSocket-protokollasta, mutta tuen puutteessa käyttää muita, hitaampia protokollia. (Kelleher 2015.)

WebSocketit käyttävät kommunikointiin TCP:tä, mikä käytännössä tarkoittaa sitä, että paketit saapuvat varmuudella perille (Kelleher 2015). Vaikka TCP on toimiva ja kenties jopa riittävä ratkaisu, projektin aikana opin, ettei se ole optimaalisin tyyli tekemäni kaltaiseen nopeatempoiseen peliin, jossa pakettien vauhti ja tiheys on tärkeämpää kuin niiden luotettava perille saapuminen.

Tästä johtuen pelissä on ajoittain havaittavissa lievää nykimistä, kun paketit saapuvat myöhässä perille.

2.3 Express

Express on Nodelle tarkoitettu rajapinta, joka tarjoaa paljon hyödyllistä toiminnallisuutta. (Expressjs.com 2015.)

Projektini tapauksessa Express pitää huolen, että pelin tiedostot päätyvät pelaajalle, joten se toimii käytännössä vain välikätenä Noden ja Socketin välillä.

2.4 P2.js

P2.js on projektissa käyttämäni fysiikkakirjasto. Jatkossa käytän P2.js:stä yksinkertaisesti nimitystä P2.

Fysiikkakirjaston valinnan kriteereinäni oli, että saan jokseenkin vaivattomasti pyöritettyä samaa kirjastoa selaimessa sekä palvelimella ja että kirjaston käyttö on mahdollisimman yksinkertaista.

P2 sisältää oman renderöintimoottorinsa, mutta sen saa myös helposti käyttämään ulkoista renderöijää kuten Pixiä. (Hedman 2015.)

2.5 Pixi.js

Pixi.js eli yksinkertaisesti Pixi on renderöintimoottori, jota usein pidetään parhaana vaihtoehtona selainpohjaisille projekteille. Se käyttää ensisijaisesti WebGL:ää, mutta tarvittaessa siirtyy canvaksen käyttöön. Pixi tukee myös mobiiliselaimia, kosketustapahtumia ja montaa muuta hienoa asiaa, joita en tässä projektissa päässyt hyödyntämään. (PixiJS 2013.)

Valitsin Pixin projektia varten lähinnä siksi, ettei minun itse tarvitsisi huolehtia siitä, miten eri selaimet tukevat mitäkin piirtotekniikkaa. Pixin avulla myös tekstin piirto on suoraviivaista.

Tarkoituksena oli käyttää Pixin tarjoamia efektejä jossain kohtaa pelissä, mutta ajanpuutteen vuoksi päätin jättää ne mahdolliseen jatkokehitykseen.

3 VERKKOPELIEN TEORIAA

3.1 Yleisesti

Palvelin on silloin malliltaan autoritaarinen, jos kaikki pelissä tapahtuva on loppukädessä palvelimen sanelemaa. Selaimet eivät liikuta omia hahmojaan vaan liike tapahtuu lähettämällä palvelimelle käyttäjän painamat syötteet, ja palvelin puolestaan laskee pelaajien liikkeet syötteiden perusteella ja lähettää hahmojen uudet koordinaatit selaimille takaisin. (Fiedler 2010.)

Etuina tällä mallilla on se, että selain ei pääse määrittelemään, mitä palvelimelle lähetetään ja näin ollen huijaamaan. Lisäksi koska kaikki pelaajat saavat päivityksensä yhdestä lähteestä, ohjelmoijan ei tarvitse ratkoa Peer2Peer eli P2P-yhteyden aiheuttamia ongelmia kuten sitä, että yhden pelaajan huonosta yhteydestä kärsii muut pelaajat. (Fiedler 2010.)

3.2 Latenssin aiheuttamat ongelmat ja ratkaisut

3.2.1 Käyttäjäpuolen ennustus

Jos selain ei liikuta pelaajan hahmoa heti syötteen jälkeen, vaan odottaa koordinaatteja palvelimelta, on pelin responsiivisuus täysin verkkoliikenteen

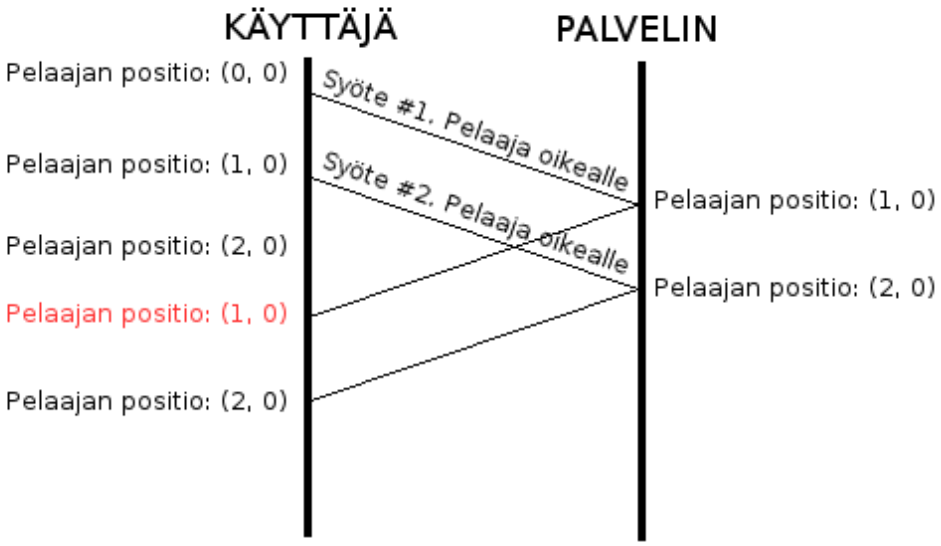
armoilla. Tämä johtaisi väistämättä viiveeseen käyttäjän näppäinpainalluksen ja ruudulla näkyvän toiminnon välillä.

Sen sijaan että nojattaisiin täysin palvelimen lähettämiin tilapäivityksiin, käyttäjän syötettä vastaava toimenpide voidaan toteuttaa selaimessa heti painamishetkellä. Näin selain ennustaa palvelimen lähettämän tilan itse toteuttamalla pelaajan liikkeen. Mahdolliset erot tilojen välillä korjaa palvelin, mutta koska selaimessa ja palvelimessa pyörivä pelilogiikka on käytännössä sama, tilat vastaavat useimmiten toisiaan. (Gambetta 2015B.)

Lopputuloksena saavutetaan käyttäjän näkökulmasta sulava pelattavuus, sekä autoritäärisen palvelimen päätavoite eli palvelimen sanelema lopputilanne pelimaailmaan.

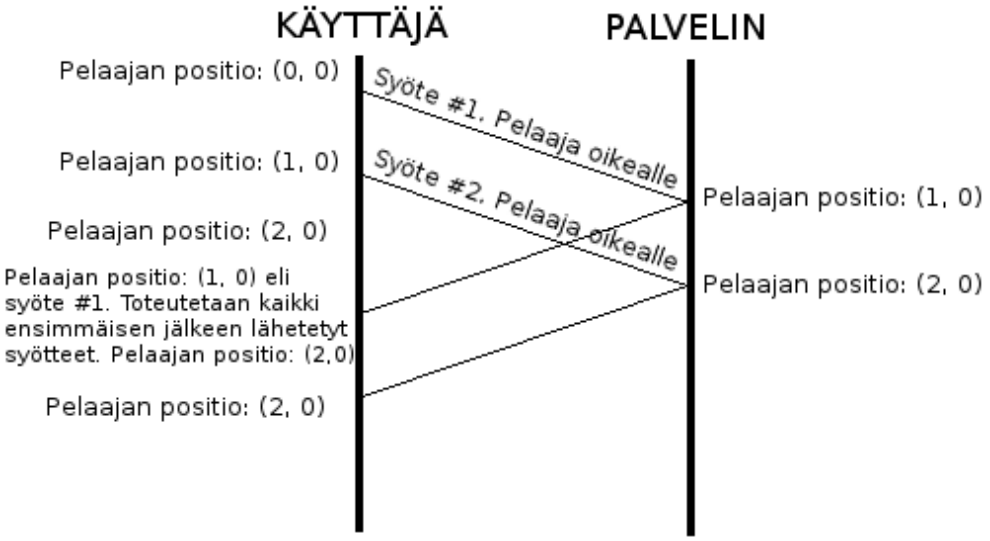
3.2.2 Palvelimen hitauden yhteensovittaminen pelitapahtumiin

Yksi mahdollinen skenaario on, että palvelimen sekä käyttäjän välillä on synkronointiongelma, jossa käyttäjä ennustaa liikkuvansa tietyn määrän tiettyyn suuntaan ja kun palvelimelta tulee kyseiseen liikkeeseen vaste, käyttäjä on jo ehtinyt liikkua eteenpäin seuraavaan sijaintiin. Tästä johtuen pelaajan liike näyttää nykivältä. (Kuva 1) (Gambetta 2015B.)



Kuva 1. Käyttäjän ja palvelimen välinen synkronointiongelma.

Jotta synkronointiongelma saataisiin korjattua, pidetään listaa painetuista syötteistä ja annetaan jokaiselle syötteelle sekvenssinumero. Palvelimen prosessoitua näppäinpainalluksen, se lähettää tilapäivityksen mukana käyttäjälle viimeksi prosessoidun syötteen sekvenssinumeron. (kuva 2) (Gambetta 2015B.)



Kuva 2. Synkrointiongelma korjattu.

Tämän jälkeen selain käy läpi paikallisen listansa, johon jokainen käyttäjältä palvelimelle lähtenyt syöte on lisätty ja vertaa palvelimen sekvenssinumeroa listalta löytyviin. Jos käyttäjän listalta löytyvän syötteen sekvenssinumero on pienempi tai yhtä suuri kuin palvelimen viimeksi prosessoiman syötteen sekvenssinumero, leikataan syöte listasta pois, sillä se on otettu jo huomioon. Kaikki syötteet, jotka löytyvät listasta mutta joita palvelin ei ole vielä prosessoinut, toteutetaan selaimessa ennustavasti. (kuva 2) (Gambetta 2015B.)

3.2.3 Entiteetin interpolaatio

Koska liikkeen ennustus toimii ainoastaan pelaajan oman hahmon kohdalla, kulkevat muut objektit vielä palvelimelta tulevien pakettien tahtiin. Alkuperäisessä toteutuksessa selain siirtää objektit suoraan palvelimen määrittämiin koordinaatteihin pakettien saapuessa. Tällöin objektien liike alhaisilla päivitysnopeuksilla on erityisen nykivää.

Ongelma korjaantuu tallentamalla palvelimen lähettämät tilat ja piirtämällä pelimaailma palvelimen näkökulmasta aina hieman jäljessä, interpoloiden kohti uusinta vastaanotettua tilapäivitystä. (Gambetta 2015A.)

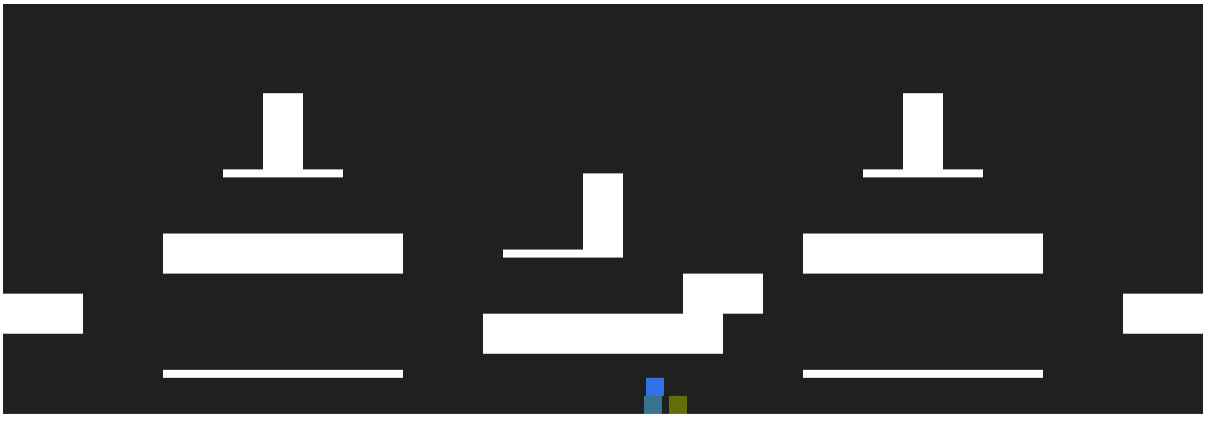
Esimerkkitilanteessa käyttäjä vastaanottaa toisen pelaajan position maailmassa joka on ajalla t0 koordinaateissa (15, 0). Seuraava paketti saapuu ajalla t1 ja sen mukaan pelaaja on siirtynyt koordinaatteihin (16, 0). Tässä vaiheessa ruvetaan piirtämään pelaajaa ajassa t0 ja liikuttamaan sitä kohti ajalla t1 saapuneen paketin koordinaatteja lineaarisesti interpoloimalla.

Tällä tavalla käyttäjän näkökulmasta muut objektit kulkevat sulavasti pelimaailmassa palvelimen päivitysnopeudesta riippumatta. Sulavuus tulee sillä hinnalla, että objektit ovat aina paketin saapumiseen kuluvan ajan lisäksi määritetyn interpolaatioajan, esimerkiksi 100 millisekuntia, verran myöhässä palvelimeen verrattuna. (Gambetta 2015A.)

4 PROJEKTIN TOTEUTUS

4.1 Pelin kuvaus

Tyyliltään peli on sivusta kuvattu, tasohyppelymäinen deathmatch-peli, jossa tavoitteena on tuhota vastustajapelaajat pommeilla. Pelikenttänä toimii alue, jossa on useita alustoja, joilla hyppiä ja joiden kautta pommit voivat kimmota. (kuva 3)



Kuva 3. Pelikenttä.

Pommeja heitetään välilyöntiä painamalla ja kullakin pelaajalla voi olla pelikentällä vain yksi pommi kerrallaan. Jos pelaaja painaa pomminheittonappulaa pommin ollessa jo kentällä, pommi räjähtää ennenaikaisesti. Pommin ajastin on kolme sekuntia, eikä pommia voi heittää useammin kuin kerran siinä ajassa. Pommin räjähtäessä tarkastetaan onko tietyn välimatkan sisällä muita pommeja tai pelaajia ja räjäytetään myös ne. (Kuva 4)

```
function explosionRadiusCheck(source) {
    var bombX = source.pos[0];
    var bombY = source.pos[1];
    var explosionRadius = 1.5;

    for(var i in players) {
        var playerX = players[i].body.position[0];
        var playerY = players[i].body.position[1];

        if(playerX > bombX - explosionRadius && playerX < bombX + explosionRadius) {
            if(playerY > bombY - explosionRadius && playerY < bombY + explosionRadius && !players[i].isDead) {
               players[i].destroy(source.id);
               if(players[source.id] != null)
                   players[source.id].kills++;
            }
        }

        if(bombs[i] != null) {
            var otherBombX = bombs[i].body.position[0];
            var otherBombY = bombs[i].body.position[1];

            if(otherBombX > bombX - explosionRadius && otherBombX < bombX + explosionRadius) {
                if(otherBombY > bombY - explosionRadius && otherBombY < bombY + explosionRadius) {
                    bombs[i].destroy(i);
                }
            }
        }
    }
}
```

Kuva 4. Räjähdysalueen tarkistus.

Pelillä ei ole varsinaista voitto- tai häviöehtoa, mutta pelin ideaa voi jalostaa eteenpäin. Pelissä pidetään kirjaa jokaisen pelaajan tapoista ja kuolemista, mutta mitään käytännön virkaa näillä ei kirjoitushetkellä vielä ole. Pelaajien määrää ei myöskään ole rajattu, vaan peli hyväksyy uusia pelaajia niin kauan kuin palvelinkapasiteettia riittää.

4.2 Palvelinympäristön pystytys

Projekti pyörii Amsterdamissa sijaitsevalla Debian-pohjaisella virtuaalipalvelimella, joten ohjelmat asennetaan komentorivin kautta, koska graafista käyttöliittymää palvelimella ei käytössä ole.

Ensin asennetaan Node.js käyttäen Debianin Apt-paketinhallintaa komennolla *sudo apt-get install nodejs*. Tämä asentaa sekä Noden että npm-paketinhallinnan, jonka kautta asennetaan osa projektin vaatimista kirjastoista. (NodeSource 2015.)

Palvelimella pyörivä Debian 7.0 ei oletuksena tarjoa Nodea paketinhallinnan kautta mutta palveluntarjoajani (Digital Ocean) on lisännyt käyttöön pakettivaraston, josta kyseinen paketti löytyy. Debianin uudemmilla versioilla tätä ongelmaa ei ole.

Seuraavaksi asennetaan loput tarvittavat paketit npm-paketinhallinnan kautta komennolla *npm install socket.io express node-uuid p2*. Tämä komento tallentaa kyseiset paketit projektikansion alle *node_modules* kansioon.

Lopuksi tarvitaan enää pixi.js, joka hoitaa pelin piirron käyttäjäpuolella. Kirjaston saa ladattua osoitteesta: https://github.com/GoodBoyDigital/pixi.js.

4.3 Pelaaja liittyy palvelimelle

4.3.1 Palvelimen toiminta

Pelaajan liittyessä peliin palvelin generoi käyttäjälle uniikin tunnisteen käyttämällä *node-uuid*-lisäosan UUID-funktiota. Tämän tunnisteen avulla palvelin ja käyttäjä pysyvät ajan tasalla siitä, kuka tekee mitäkin. Lisäksi pelaajalle luodaan oma väri generoimalla se satunnaisesti. (Kuva 5)

```
io.on('connection', function(socket){

    player = {
        id : UUID(),
        color : (Math.random()*0xFFFFFF<<0).toString(16),
    };

    players[player.id] = new Player(player.id);
    players[player.id].color = player.color;
```

Kuva 5. Pelaajan luonti.

Tämän jälkeen luodaan uusi Player-objekti, jolle annetaan äsken luodut ominaisuudet ja joka lisätään players-taulukkoon (kuva 5). Objektin luomisessa maailmaan lisätään neliön muotoinen pelaaja ja sille annetaan sen fyysiset ominaisuudet kuten massa ja positio (kuva 6).

```
var Player = function(id) {
    var rectangle= new p2.Rectangle(size,size);
    this.pid = id;
    this.isDead = false;
    this.ableToFire = true;
    this.timer = 0;
    this.kills = 0;
    this.deaths = 0;
    this.color = '0x#FFFFFF';

    var opts = {
        mass: 1,
        position: [0,-0.5],
    };

    this.lastProcessedInput = [];

    this.body = new p2.Body(opts);
    this.body.addShape(rectangle);
    this.body.fixedRotation = true;
    this.body.damping = 0.9;
    world.addBody(this.body);
};
```

Kuva 6. Uuden pelaajan luominen palvelimella.

Palvelimen luotua instanssin pelaajasta, lähetetään käyttäjälle oma tunnisteensa sekä lista jo palvelimella valmiiksi olevista pelaajista. (kuva 7)

Lisäksi peliin liittyneen uuden käyttäjän tunniste sekä väri lähetetään palvelimella oleville pelaajille. (kuva 7)

```
var connectedPlayers = [];
for(var i in players) {
    connectedPlayers.push({
        id: players[i].pid,
        color: players[i].color,
    });
}

socket.emit('onconnected',{pid: player.id, list: connectedPlayers});
socket.broadcast.emit('newconnected', player);
```

Kuva 7. Käyttäjälle itselleen sekä muille lähetettävät tiedot.

4.3.2 Selaimen toiminta

Liittyessään peliin käyttäjä vastaanottaa palvelimelta oman henkilökohtaisen tunnisteensa sekä listan muista pelaajista (Kuva 8).

Listan avulla selain luo kopiot palvelimella olevista pelaajista, lisää ne players-listaan ja tämän jälkeen vertaa luotujen pelaajien tunnisteita omaan tunnisteeseensa ja näin päättelee, mikä pelaajista on käyttäjän oma (Kuva 8).

```
socket.on('onconnected', function(pid, list) {
    myid = pid.pid;

    for(var i in pid.list) {
        var n = pid.list[i];
        if(n.id != null) {
            players[n.id] = new Player(n.id, n.color);

            if(myid === n.id) {
                myPlayer = players[n.id];
                drawText(n.color, "#000000", n.id,"CONNECTED", "");
            }
        }
    }
```

Kuva 8. Pelaaja liittyy palvelimelle ja luo pelaajien hahmot palvelimen tietojen mukaan.

Uuden pelaajan liittyessä peliin vanhat pelaajat saavat pelaajan tunnisteen sekä värin ja luovat näillä tiedoilla uuden pelaajaobjektin maailmaansa (Kuva 9).

```
function userConnected() {
    socket.on('newconnected', function(player) {
        drawText(player.color, "#000000", player.id,"CONNECTED", "");
        players[player.id] = new Player(player.id, player.color);
    });
}
```

Kuva 9. Pelaajat saavat tiedon uuden käyttäjän saapumisesta.

Peliobjektin luonti toimii samalla periaatteella kuin palvelimella (kuva 6). Erona on se, että selaimella pelaajalle annetaan graafiset ominaisuudet, jotta Pixi voi ne piirtää.

4.4 Pelin aikainen logiikka

4.4.1 Palvelimen toiminta

Pelin aikana palvelimella pyörii pelisilmukka, joka päivittää pelin toimintoja 64 kertaa sekunnissa. Näihin kuuluvat fysiikkamallinnus, pommin ajastimien laskeminen, pommien heittotiheyden viiveen laskeminen sekä pelimaailman tilan lähetys pelaajille.

Pelaajien näppäinpainallusten käsittely

Palvelin käsittelee käyttäjän lähettämät näppäinpainallukset niiden saapuessa (kuva 13). Pelaajan painaessa pomminheittonappulaa eli välilyöntiä, palvelin tarkastaa, onko pelaajalla jo pommeja pelikentällä. Jos pelaajalla on jo pommi kentällä ja pommin ajastin on pyörinyt puoli sekuntia, pommi räjähtää. Muuten palvelin kutsuu pelaajan fire-funktiota, joka luo ja ampuu uuden pommin pelaajan liikkumasuuntaan (kuva 11) sekä lähettää muille pelaajille viestin että pelaaja on ampunut (kuva 10).

```
Player.prototype.fire = function () {
    this.ableToFire = false;

    var pos = {
        x: this.body.position[0],
        y: this.body.position[1],
        vel: this.body.velocity,
    };

    bombs[this.pid] = new Bomb(pos);

    var data = {
        id: this.pid,
        bomb: {x: bombs[this.pid].body.position[0], y: bombs[this.pid].body.position[1]},
    };

    io.emit('fired', data);
};
```

Kuva 10. Funktio joka ampuu pommin.

```
var Bomb = function(pos) {
    var circle = new p2.Circle(size/2);
    this.ttl = 3;

    var opts = {
        mass: 5,
        position: [pos.x, pos.y],
        velocity: [pos.vel[0] * 3, pos.vel[1] + 3],
    };

    this.body = new p2.Body(opts);
    this.body.addShape(circle);
    this.body.fixedRotation = false;
    this.body.damping = 0;
    this.body.gravityScale = 0.6;
    world.addBody(this.body);
};
```

Kuva 11. Uuden pommin luonti.

Suuntanäppäimiä painettaessa palvelin lisää tai vähentää pelaajan nopeutta. Hyppy toimii samalla periaatteella, mutta ennen hyppyä palvelin tarkistaa, onko pelaaja maassa.

Jos pelaaja painaa r-painiketta, katsotaan, onko pelaaja kuollut vai ei. Jos pelaaja on kuollut, kutsutaan respawn-funktiota, joka lisää pelaajan takaisin maailmaan ja lähettää muille pelaajille viestin, jotta myös selaimet osaavat herättää kuolleen pelaajan (kuva 12).

```
Player.prototype.respawn = function () {
    this.body.position[0] = 0;
    this.body.position[1] = -0.5;
    world.addBody(this.body);
    this.isDead = false;
    var data = this.pid;
    io.emit('respawned', data);
};
```

Kuva 12. Funktio joka herättää pelaajan henkiin ja ilmoittaa siitä muille pelaajille.

Kun näppäinpainallus on käsitelty, merkitään se viimeiseksi käsitellyksi painallukseksi (kuva 13).

```
Player.prototype.processInputs = function(input) {
    var id = input.pid;
    if(this.isDead && input.respawn)
        this.respawn();

    if(!this.isDead) {
        if(input.up && checkIfCanJump(this.body))
            this.body.velocity[1] = jumpforce;

        if(input.fire && bombs[this.pid] == null && this.ableToFire)
            this.fire();
        else if(bombs[this.pid] != null && !this.ableToFire && input.fire && bombs[this.pid].ttl <= 2.5)
            bombs[this.pid].destroy(this.pid);

        if(input.right)
            this.body.velocity[0] = mvspd;
        else if(input.left)
            this.body.velocity[0] = -mvspd;

        this.lastProcessedInput[id] = input.seqNumber;
    }
};
```

Kuva 13. Syötteen prosessointi palvelimella.

Pelimaailman tilan lähetys selaimille

Pelisilmukan alussa lähetetään pelaajille maailman tämänhetkinen tila. Lähetettävään pakettiin kerätään jokaisen pelaajan tunniste, koordinaatit, nopeus, pelaajan pommin paikka, viimeksi käsitelty painallus sekä tapot ja kuolemat. Tämän jälkeen paketti lähetetään jokaiselle pelaajalle. (kuva 14)

```
var sendStateToClients =  function() {
    var state = [];
    for(var i in players) {
        var p = players[i];
        var bomb;
        if(bombs[p.pid] != null)
            bomb = {x: bombs[p.pid].body.position[0], y: bombs[p.pid].body.position[1]};

        state.push({
            id: p.pid,
            position: {x: p.body.position[0], y: p.body.position[1], vel: p.body.velocity},
            bomb: bomb,
            lastProcessedInput: p.lastProcessedInput[p.pid],
            kills: p.kills,
            deaths: p.deaths,
        });
    }
```

Kuva 14. Pelaajille lähetettävä paketti joka sisältää maailman tilan.

Pelin päivitysnopeutta sekä pakettien lähetystiheyttä kannattaa kokeilla palvelimen tehoista sekä pelin tarpeista riippuen. Itse valitsin molemmiksi 64 kertaa sekunnissa, koska tällä hetkellä muutaman pelaajan kuormituksella peli ei ole hirveän prosessori-intensiivinen eikä palveluntarjoajani ole asettanut kattoa käyttööni annetulle kaistalle.

4.4.2 Selaimen toiminta

Näppäinpainallusten käsittely

Käyttäjän painaessa näppäimiä, sallittuihin näppäinpainalluksiin liitetään sekvenssinumero sekä pelaajan tunniste ja tiedot lähetetään palvelimelle. Painallus lisätään myös listaan ”pendingInputs” myöhempää käyttöä varten. (kuva 15)

Lisäksi painallus syötetään applyInput-funktioon, jotta selain voi ennustavasti toteuttaa syötteen sen sijaan, että se odottaisi palvelimen vastausta (kuva 15). Jokainen syöte lisää pelaajan nopeutta liikesuuntaan samalla tavalla kuin palvelimella (kuva 13).

```
Player.prototype.processInputs = function() {

    var input = {
        fire: false,
        right: false,
        left: false,
        up: false,
        respawn: false,
    };

    if(buttons.fire)
        input.fire = true;

    if(buttons.up)
        input.up = true;

    if(buttons.left)
        input.left = true;

    if(buttons.right)
        input.right = true;

    if(this.isDead && buttons.respawn)
        input.respawn = true;

    for(i in input) {
        if(input[i])
        {
            input.seqNumber = this.seqNumber++;
            input.pid = this.pid;
            socket.emit('move', input);
            this.applyInput(input);
            this.pendingInputs.push(input);
        }
    }
};
```

Kuva 15. Painallusten käsittely ja lähetys palvelimelle.

Pelimaailman tilan vastaanotto

Kun pelaaja on lähettänyt näppäinpainalluksensa palvelimelle, palvelin laskee pelaajan uudet koordinaatit ja lähettää ne takaisin pelaajille. Ennustuksen ansiosta pelaajan ja palvelimen tilat ovat jo valmiiksi melko lähellä toisiaan, mutta tarpeelliset korjaukset tulevat kuitenkin palvelimelta.

Jotta käyttäjän ja palvelimen välinen synkronointiongelma saataisiin korjattua, pidetään listaa palvelimelle lähetetyistä syötteistä pendingInputs-nimisessä taulukossa.

Palvelimen tilapäivitysten mukana tulee aina palvelimen viimeksi prosessoiman painalluksen sekvenssinumero, jota verrataan pendingInputs-taulukosta löytyviin syötteisiin (kuva 16).

Jos palvelimen viimeksi prosessoima syöte on uudempi tai sama kuin käyttäjän listalta löytyvä, leikataan syöte listasta pois. Kaikki syötteet, jotka löytyvät listasta mutta joita palvelin ei ole vielä prosessoinut, syötetään applyInput-funktioon, jotta selain voi yrittää ennustaa palvelimelta tulevan tilan (kuva 16).

```
function receiveState() {
    socket.on('moved', function(state) {
        for(var i in state) {
            var p = state[i];
            if(players[p.id] == myPlayer) {

                myPlayer.body.position[0] = p.position.x;
                myPlayer.body.position[1] = p.position.y;
                var j = 0;
                while(j < myPlayer.pendingInputs.length) {
                    var input = myPlayer.pendingInputs[j];
                    if(input.seqNumber <= p.lastProcessedInput) {
                        myPlayer.pendingInputs.splice(j,1);
                    }
                    else {
                        myPlayer.applyInput(input);
                        j++;
                    }
                }
            }
            else if(players[p.id] != null) {
                players[p.id].applyState(p);
            }

            if(bombs[p.id] != null && p.bomb != null) {
                bombs[p.id].body.position[0] = p.bomb.x;
                bombs[p.id].body.position[1] = p.bomb.y;
            }
        }
    });
}
```

Kuva 16. Pelaaja vastaanottaa päivityksiä palvelimelta. (Gambetta 2015C.)

Valitettavasti ajanpuutteen vuoksi en saanut entiteetin interpolaatiota täysin toimivana implementoitua, mutta kompensoin puutetta korottamalla palvelimen päivitysnopeuden 64 kertaan sekunnissa, joka on riittävä määrä objektien sulavan liikkeen piirtoon.

Muiden palvelinviestien vastaanotto

Useaan kertaan lähetettävien tilapäivitysten lisäksi palvelin lähettää myös tiettyjen tilanteiden sattuessa ilmoituksen selaimille.

Pelaajan ampuessa pommin, on siitä lähetettävä viesti selaimille, jotta selaimet osaavat piirtää sen (kuva 17). Pommin instantiointi toimii samalla tavalla kuin palvelimella (kuva 11), erona se, että selaimessa pommille ei anneta suuntaa eikä nopeutta ja sille annetaan graafiset ominaisuudet, jotta Pixi voi sen piirtää.

```
function userFired() {
    socket.on('fired', function(data) {
        bombs[data.id] = new Bomb(data.bomb);
    });
}
```

Kuva 17. Käyttäjä saa palvelimelta viestin että joku on ampunut pommin.

Samalla tavalla kuin pommin ampumisessa, myös pommin räjähtäessä lähetetään viesti palvelimelta selaimille.

Viestin saapuessa selain luo yksinkertaisen for-silmukan kautta partikkeliobjekteja. Valitsin for-silmukan laskemaan muuttujalle *i*, luvut nollasta viiteenkymmeneen, jolloin partikkeleita syntyy 51 kappaletta. Kullekin partikkelille arvotaan satunnainen x-akselin nopeus miinus kahdeksan ja kahdeksan väliltä. Partikkelin y-akselille annetaan nopeus *i*-muuttujan mukaan, jolloin partikkelit lentävät vaihtelevilla nopeuksilla yläsuuntaan. Partikkeliluokassa on myös valmiiksi määritelty räjähdykseen sopivia värejä, joista sattumanvaraisesti valitaan yksi partikkelia luotaessa. Partikkeleita luotaessa argumenteiksi annetaan myös koordinaatit, jotka vastaavat räjähdyspaikkaa sekä elinikä, joka pommin tapauksessa on puoli sekuntia. (kuva 18)

Lopuksi partikkelit luodaan particles-taulukkoon, jonka läpi iteroidaan pelisilmukassa, samalla pudottaen partikkeleiden elinikää ja eliniän pudotessa nollaan, poistaen partikkelit maailmasta.

Lopputuloksena partikkelit lentävät räjähdysmäisesti pommin ympärillä (kuva 20) kutsuttaessa pommin destroy-funktiota (kuva 19).

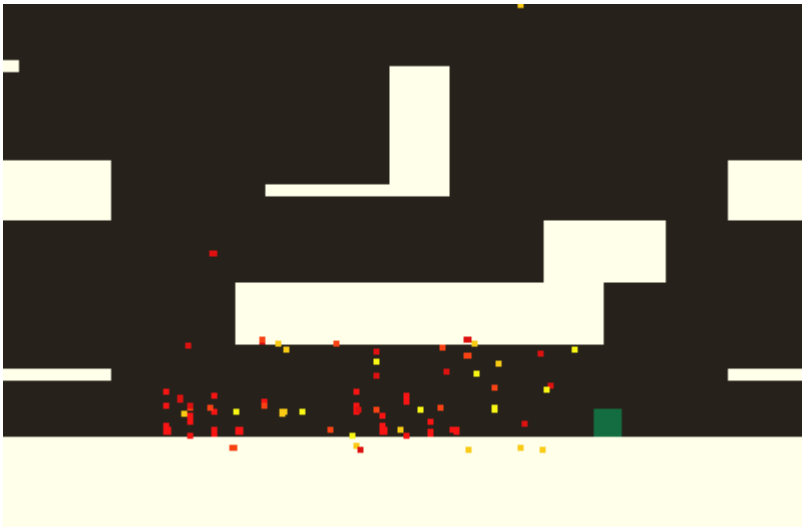
```
function onExplosion() {
    socket.on('exploded', function(id) {
        if(bombs[id] != null) {
            var x = bombs[id].body.position[0];
            var y = bombs[id].body.position[1];
            for(var i = 0; i < 50; i++) {
                var randX = Math.floor((Math.random() * (8 - (-8) + 1) ) + (-8));
                var randColor = Math.floor((Math.random() * (4 - (1) + 1) ) + (1));
                var vel = {x: randX, y: i/2};
                particles.push(new Particles(x, y, vel, randColor, 0.5));
            }
            bombs[id].destroy(id);
        }

    });
}
```

Kuva 18. Selain vastaanottaa palvelimelta viestin pommin räjähtämisestä.

```
Bomb.prototype.destroy = function(id) {
    world.removeBody(this.body);
    container.removeChild(this.graphics);
    bombs.splice(id, 1);
};
```

Kuva 19. Pommin fyysinen keho sekä grafiikkaominaisuudet poistetaan.



Kuva 20. Pommi ja pelaaja räjähtävät samaan aikaan.

Kun pommin räjähdysalueelta löytyy pelaaja, selain vastaanottaa palvelimelta viestin pelaajan tuhoutumisesta. Samalla tavalla kuin pommin räjähtäessä, myös pelaajan kuollessa luodaan partikkeleita. Pelaajan räjähtäessä partikkeleille annetaan pienempi x-, ja y-akselin nopeus kuin pommin räjähdyksessä,

partikkeleita luodaan vähemmän ja partikkeleiden eliniäksi merkintään 10 sekuntia. Lisäksi jokaisen partikkelin väri on punainen. (kuva 20)

Pelaajan tuhouduttua näytetään ruudulla, kuka sen tuhosi. Koska pelaajille ei vielä voi antaa nimiä, kirjoitetaan pelaajan nimen sijasta vain ”player”. Jos pelaaja kuoli omaan pommiinsa, kirjoitetaan ”player committed suicide”. Jos taas tuhoajana oli joku muu, kirjoitetaan ”player killed player” (kuva 21). Pelaajan ”nimi” tulee aina pelaajan omalla värillä, jotta olisi helpompi erottaa kuka tuhosi kenet. (kuva 22)

```
function onDestroyed() {
    socket.on('destroyed', function(data) {
        if(players[data.id] != null && !players[data.id].isDead) {
            var x = players[data.id].body.position[0];
            var y = players[data.id].body.position[1];
            for(var i = 0; i < 15; i++) {
                var randX = Math.floor((Math.random() * (2 - (-2) + 1) ) + (-2));
                var vel = {x: randX, y: i/5};
                particles.push(new Particles(x, y, vel, 0, 10));
            }
            players[data.id].destroy();
            if(data.id === data.destroyerId)
                drawText(players[data.id].color, players[data.destroyerId].color, "PLAYER", "COMMITTED SUICIDE", "");
            else
                drawText(players[data.destroyerId].color, players[data.id].color, "PLAYER", "KILLED", "PLAYER");
        }
    });
}
```

Kuva 21. Selain käsittelee palvelimelta tulleen viestin pelaajan tuhoutumisesta.

Kuva 22. Teksti joka ilmaantuu pelaajan tuhotessa toisen pelaajan ja itsensä.

Jokaiselle pelaajalle on myös tarpeellista ilmoittaa, jos joku on kuollut ja haluaa herätä uudestaan henkiin. Tällöin kutsutaan kyseisen pelaajan respawn-funktiota, joka merkitsee pelaajan isDead-muuttujan epätodeksi ja lisää fyysiset ja graafiset ominaisuudet takaisin maailmaan. (kuva 23)

```
Player.prototype.respawn = function() {
    this.isDead = false;
    world.addBody(this.body);
    container.addChild(this.graphics);
};
```

Kuva 23. Palvelimelta tullut viesti kutsuu pelaajan respawn-funktiota.

Peliobjektien piirto

Jokainen peliobjekti luodaan pelin alussa kutakuinkin samalla tavalla kuin palvelimella. Erona on se, että objektille annetaan myös grafiikkaominaisuudet, jotta sen voi esittää visuaalisesti. Tämä onnistuu kätevästi käyttämällä Pixi.js-kirjastoa.

Objektille valitaan ensin täyteväri, jonka jälkeen objektia kuvaava muoto piirretään objektin koordinaatteihin. Tämän jälkeen jokainen objekti lisätään ns. säiliöön joka myöhemmin piirretään kokonaisuudessaan peliluupissa. (Kuva 24)

```
this.graphics = new PIXI.Graphics();
this.graphics.beginFill("0x" + col);
this.graphics.drawRect(-rectangle.width/2, -rectangle.height/2, rectangle.width, rectangle.height);
container.addChild(this.graphics);
```

Kuva 24. Pelaajalle luodaan grafiikat.

Jotta luodut grafiikat liikkuisivat objektien mukana, täytyy niiden paikkaa muuttaa jokaisella piirtokerralla objektin fyysisen kehon mukana. Tämä saavutetaan pitämällä objekteja listassa, jonka läpi kuljetaan jokaisella peliluupin iteraatiolla samalla päivittäen objektien grafiikkaominaisuuksia. (kuva 25)

```
for(var i in bombs) {
    if(bombs[i] != null) {
        bombs[i].graphics.position.x = bombs[i].body.position[0];
        bombs[i].graphics.position.y = bombs[i].body.position[1];
        bombs[i].graphics.rotation = bombs[i].body.angle;
    }
}
```

Kuva 25. Pommien grafiikoiden paikkaa päivitetään peliluupissa.

Tekstin piirto

Halusin toteuttaa peliin tekstiosion, johon ilmaantuu aina tietyn tapahtuman seurauksena tietoa kyseisestä tapahtumasta ja joka tietyn ajan kuluessa häviää pois ruudulta.

Pelaajan liittyessä peliin, poistuessa pelistä ja tuhotessa jonkun kutsutaan drawText-funktiota, joka ottaa argumenteiksi värin numero yksi, värin numero kaksi sekä kolme erillistä tekstikenttää.

Funktio luo tekstejä allekkain aina edellisen tekstin paikan mukaan. Jokaiselle tekstille annetaan eliniäksi 5 sekuntia ja se lisätään textLogs-listaan. (kuva 26)

Listan läpi iteroidaan moveText-funktiossa, jossa tekstien paikkaa lasketaan alaspäin kunnes viimein tekstin eliniän tippuessa nollaan, se poistetaan. (kuva 27)

```
function drawText(firstColor, thirdColor, firstText, secondText, thirdText) {

    if(textLogs.length != 0)
        lastTextHeight = textLogs[textLogs.length-1].first.position.y;
    else
        lastTextHeight = y;

    var first;
    var second;
    var third;

    var text = new PIXI.Text("", {font:"12px Arial", fill:"#000000"});
    text.first = new PIXI.Text(firstText, {font:"20px Arial", fill:"#" + firstColor, stroke: "#elelel", strokeThickness: 3});
    text.second = new PIXI.Text(secondText, {font:"20px Arial", fill:"#000000", stroke:"#elelel", strokeThickness: 5});
    text.third = new PIXI.Text(thirdText, {font:"20px Arial", fill:"#" + thirdColor, stroke:"#elelel", strokeThickness: 3});

    text.ttl = 5;
    var offSet = 25;
    var posY;

    text.position.x = x;

    if(lastTextHeight >= y + offSet)
        posY = y;
    else
        posY = lastTextHeight - offSet;

    text.first.position.y = posY;
    text.second.position.y = posY;
    text.third.position.y = posY;

    text.first.position.x = x;
    text.second.position.x = text.first.position.x + (offSet * 15);
    text.third.position.x = text.second.position.x + (offSet * 5);

    stage.addChild(text.first);
    stage.addChild(text.second);
    stage.addChild(text.third);
    textLogs.push(text);
}
```

Kuva 26. Tekstinpiirtofunktio

```
function moveText(delta) {

    var scrollSpeed = 35;
    for(var i in textLogs) {
        var t = textLogs[i];
        if(t.ttl <= 0) {
            stage.removeChild(t.first);
            stage.removeChild(t.second);
            stage.removeChild(t.third);
            t.first.destroy();
            t.second.destroy();
            t.third.destroy();
            textLogs.splice(i, 1);
        }else
            t.ttl -= delta;

        t.first.position.y += delta * scrollSpeed;
        t.second.position.y += delta * scrollSpeed;
        t.third.position.y += delta * scrollSpeed;
    }
}
```

Kuva 27. Funktio joka liikuttaa tekstiä.

Kesken jäänyt ominaisuus oli piirtää jokaisen pelaajan tapot ja kuolemat listaan ruudun alareunaan. Piirto onnistui kyllä mutta ruudunpäivitysnopeus putosi niin alas, että pelattavuus kärsi vakavasti. Pienellä hiomisella lista on täysin toteutettavissa.

4.5 Pelaaja lähtee palvelimelta

4.5.1 Palvelimen toiminta

Pelaajan sulkiessa pelin Socket.io vastaanottaa disconnect-tapahtuman, jolloin selaimille lähetetään lopettaneen pelaajan tunniste, jotta selain osaa poistaa kyseisen pelaajan hahmon maailmasta.

Tämän jälkeen palvelin poistaa maailmasta oman kopionsa pelaajan hahmosta ja poistaa pelaajan players-listasta. (kuva 28)

```
socket.on('disconnect', function() {
    console.log(ts + player.id + " disconnected");
    io.emit('dc', player.id);

    if(players[player.id] != null) {
        world.removeBody(players[player.id].body);
        delete players[player.id];
    }
});
```

Kuva 28. Käyttäjän poistuminen palvelimen näkökulmasta.

4.5.2 Selaimen toiminta

Selaimen lähtiessä palvelimelta, palvelin vastaanottaa disconnect-tapahtuman, jolloin palvelin lähettää muille pelaajille viestin, että käyttäjä on lähtenyt. Saamallaan tunnisteella selain poistaa kyseisen pelaajan players-taulukosta ja kutsuu pelaajan destroy-funktiota, joka poistaa hahmon fyysiset sekä graafiset ominaisuudet maailmasta. (kuva 29)

```
function userDisconnected() {
    socket.on('dc', function(pid) {
        if(players[pid] != null) {
            drawText(players[pid].color, "#000000", pid, "DISCONNECTED","");
            players.splice(pid, 1);
            players[pid].destroy();
        }
    });
}
```

Kuva 29. Käyttäjän poistuminen selaimen näkökulmasta.

5 PÄÄTELMÄT, PARANNUSEHDOTUKSET JA JATKOKEHITYS

5.1 Skaalautuvuus

Selaimeen tehdyt pelit havittelevat mahdollisimman suurta yleisöä valitsemalla alustakseen vaivattoman, nopeasti ja usealla alustalla saatavilla olevan selaimen. Tästä syystä verkkopelin on oltava mukautumiskykyinen ja skaalautua suurelle yleisölle pienessä ajassa.

Autoritäärinen malli selainpelissä törmää tähän ongelmaan nopeasti, sillä palvelinkapasiteetin tarve nousee pelaajamäärän mukana. Normaalisti tämä ei ole peleillä ongelma, sillä usein monet pelaajat ylläpitävät itse omia palvelimi-

aan suosituissa peleissä. Selainpohjaisella pelillä tämä ei kuitenkaan ole niin suoraviivaista.

Ehdotankin ratkaisuksi käyttämään palvelinta ainoastaan pelisession muodostamiseen ja itse viestintään Peer2Peer-mallia, jossa käyttäjät lähettävät paketteja toisilleen ja leikkaavat palvelimen välistä kokonaan. Tämä toimii hyvin pienillä pelaajamäärillä ja leikkaa palvelinkapasiteetin tarvetta mittavasti.

Ongelmaksi tässä mallissa koituu mahdolliset huijarit, joiden syötteitä palvelin ei pääse verifioimaan ja verkkotopologiasta riippuen mahdolliset latenssiongelmat yhden pelaajan ongelmallisen verkkoyhteyden aiheuttamana.

5.2 WebRTC vs WebSocket

Kuten aiemmin mainittua, WebSocketit tukevat ainoastaan TCP-tietoliikenneprotokollaa, jonka luotettavat paketit aiheuttavat huomattavaa viivettä nopeatempoisissa peleissä. Myöskään Peer2Peer-yhteydet eivät ole mahdollisia WebSocketeilla.

Nopeatempoisille peleille kenties parempi vaihtoehto olisikin käyttää uudempaa WebRTC:tä joka tarjoaa UDP-implementaation ja näin hieman nautinnollisemman pelattavuuden, kun luotettavasti saapuvien pakettien aiheuttama ajoittainen nykiminen eliminoidaan. WebRTC tukee myös Peer2Peer-yhteyksiä, joten skaalautuvuusongelmat ratkeaa samalla. (Webrtc.org 2015.)

Toimiva vaihtoehto käyttämälleni WebSocket-pohjaiselle projektilleni on käyttää Peer.js kirjastoa, joka tarjoaa WebRTC-implementaation sekä Node.js pohjaisen palvelimen pelaajien yhdistämiseen. (Peerjs.com 2015.)

5.3 Jatkokehitys

Peli-ideaa on helppo jalostaa eteenpäin lisäämällä vaikkapa kierrosaika, jonka loputtua katsotaan, kuka pelaaja sai eniten tappoja.

Ilmiselvä lisäämisen arvoinen ominaisuus on antaa pelaajille mahdollisuus valita hahmonsa nimi sen sijaan, että pelaajat joutuu erottelemaan pelkästään värin perusteella.

Jotta peli ei menisi liian hektiseksi, olisi pelaajat myös mahdollista jaotella muutamien pelaajien ryhmiin ja antaa heille oma instanssi pelimaailmasta. Itse kuitenkin pidän nykyisestä tyylistä, jossa palvelimella on vain yksi maailma johon kaikki liittyvät.

Teknisellä tasolla lähetettävien pakettien kokoa voi parantaa poistamalla esim. tapot ja kuolemat monta kertaa sekunnissa lähetettävien tilapäivitysten paketeista.

5.4 Loppupäätelmät

Projekti onnistui mielestäni kohtalaisen hyvin, vaikka alussa arkkitehtuurin ja itse pelin suuntaa haettiin pidemmän aikaa. Opin toteutuksen aikana paljon selaimen verkkoteknologioista, verkkopelien arkkitehtuureista sekä itse JavaScriptin käytöstä.

Autoritäärinen palvelinlogiikka toimii mielestäni parhaimmillaan alustoilla, joilla käyttäjät voivat isännöidä omia palvelimiaan ja näin ollen keventää kehittäjien ylläpitotaakkaa. Toki jokainen ratkaisu pitää harkita tuotteen mukaan.

Jatkokehityksenä haluankin muuttaa pelin palvelinarkkitehtuurin P2P-malliksi: osittain oppimisen ja osittain skaalautuvuuden vuoksi. Lisäksi haluan luoda muutaman pelimekaniikan lisää ja parantaa pelin kilpailullisia kannustimia luomalla kenties jonkunlaisen johtoportaan.

LÄHTEET

Expressjs.com, 2015. *Express - Node.js web application framework*. Verkkosivu saatavilla: http://expressjs.com/ [2.6.2015].

Fiedler, G. 2010. *Gaffer on Games | What every programmer needs to know about game networking*. Gafferongames.com. Verkkosivu saatavilla: http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/ [2.6.2015].

Gambetta, G. 2015A. *Gabriel Gambetta - Fast-Paced Multiplayer (Part III): Entity Interpolation*. Gabrielgambetta.com. Verkkosivu saatavilla: http://www.gabrielgambetta.com/fpm3.html [22.4.2015].

Gambetta, G. 2015B. *Gabriel Gambetta - Fast-Paced Multiplayer (Part II): Client-Side Prediction and Server Reconciliation*. Gabrielgambetta.com. Verkkosivu saatavilla: http://www.gabrielgambetta.com/fpm2.html [22.4.2015].

Gambetta, G. 2015C. *Gabriel Gambetta - Fast-Paced Multiplayer: Sample Code and Live Demo*. Gabrielgambetta.com. Verkkosivu saatavilla: http://www.gabrielgambetta.com/fpm_live.html [22.4.2015].

Hedman, S. 2015. *p2.js*. Verkkosivu saatavilla: https://github.com/schteppe/p2.js [2.6.2015].

Kelleher, F. 2015. *Understanding Socket.IO*. Verkkosivu saatavilla: https://nodesource.com/blog/understanding-socketio [2.6.2015].

Nodejs.org, 2015. *Node.js*. Verkkosivu saatavilla: https://nodejs.org/ [2.6.2015].

NodeSource, 2015. *Node.js v0.12, io.js, and the NodeSource Linux Repositories*. Verkkosivu saatavilla: https://nodesource.com/blog/nodejs-v012-iojs-and-the-nodesource-linux-repositories [2.6.2015].

Peerjs.com, 2015. *PeerJS - Simple peer-to-peer with WebRTC*. Verkkosivu saatavilla: http://peerjs.com/ [2.6.2015].

PixiJS, 2013. *Pixi.js - 2D webGL renderer with canvas fallback*. Verkkosivu saatavilla: http://www.pixijs.com/ [2.6.2015].

Souza, C. 2014. *Learn and Understand Node.js*. Verkkosivu saatavilla: https://www.codeschool.com/blog/2014/10/30/understanding-node-js/ [2.6.2015].

Webrtc.org, 2015. *FAQ - WebRTC*. Verkkosivu saatavilla: http://www.webrtc.org/faq#TOC-What-is-WebRTC- [2.6.2015].

Liite 1

PROJEKTIN LÄHDEKOODI

https://github.com/jstu/bombgame

PELIN KOTISIVU

http://sipuli.koponen.io/