Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Kari Pajunen

Ajax ja sen käyttö peliohjelmoinnissa

Insinöörityö 12.4.2010

Ohjaava opettaja: yliopettaja Kirsti Äystö

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Kari Pajunen Ajax ja sen käyttö peliohjelmoinnissa 80 sivua 12.4.2010
Koulutusohjelma	tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaava opettaja	yliopettaja Kirsti Äystö

Insinöörityön tavoitteena oli tutustua vuorovaikutteisten web-sovellusten toteuttamisen mahdollistavaan Ajax-tekniikkaan ja siihen liittyviin teknologioihin. Työssä kerrottiin Ajaxin historiasta sekä esitettiin Ajaxin hyvät ja huonot puolet yleisellä tasolla. Lisäksi selvitettiin, minkä eri teknologioiden käyttöä Ajax edellyttää ja minkälaisiin sovelluksiin Ajaxia voi käyttää. Eniten käytössä olevat teknologiat esiteltiin ja niiden käyttämistä havainnollistettiin koodiesimerkkien avulla. Ajaxin sovelluskohteita lueteltiin, ja selvitettiin, mitkä tämän hetken suosituimmista web-palveluista Ajaxia käyttävät.

Toisena päätavoitteena oli kehittää toimiva pelisovellus Ajaxia ja dynaamista sivunkäsittelyä soveltaen. Tarkoituksena oli siis suunnitella ja ohjelmoida alusta loppuun interaktiivinen pelisovellus internetissä julkaistavaksi. Tällä haluttiin selvittää Ajaxin soveltuvuutta peliohjelmointiin. Lisäksi haluttiin tietää, kuinka hyvin tällä hetkellä käytössä oleva HTML-standardi suoriutuu peliohjelmoinnin luomista haasteista. Tämän ohella haluttiin myös tutkia, miten olio-ohjelmointi JavaScript-kielellä onnistuu.

Työn tuloksena havaittiin, että Ajax on erittäin hyvä tekniikka työpöytäsovellusmaisten web-sivujen toteuttamiseen. Työn jälkimmäisessä osassa opittiin soveltamaan Ajaxia yksinkertaisten pelien ohjelmoinnissa. HTML4 näytti soveltuvan tietyin rajoituksin myös peliohjelmointiin, mutta tulossa olevan HTML5:n arvioitiin olevan järkevämpi vaihtoehto tulevaisuuden Ajax-pelisovelluksien perustaksi. Pelin julkaisuvaihetta ei vielä saavutettu, koska lopullinen viimeistely ja testaus jäi vielä kesken. Lopullinen sovelluksen julkaisu tullaan tekemään lähitulevaisuudessa.

Hakusanat	Ajax, peliohjelmointi, web-ohjelmointi, HTML5

Helsinki Metropolia University of Applied Sciences Abstract

Author Title Number of Pages Date	Kari Pajunen Ajax and its use in game programming 80 12 April 2010
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor	Kirsti Äystö, Principal Lecturer

The goal of this thesis was to explore a technique called Ajax, used for making more interactive web applications, as well as also study the technologies behind it. The history of Ajax and the pros and cons of the technique were presented in general. The most used Ajax technologies were introduced and demonstrated with code examples. A few possible application areas were mentioned and it was also reviewed which of the most popular web services today are powered with Ajax.

The second main goal was to develop a working game application using Ajax and dynamic display morphing. The idea was to design and program an interactive game to be published on the internet. This project helped to find out how well Ajax can be applied to game programming. A further objective was to evaluate the possibility to use the current HTML standard in game programming. In addition to that, there was a desire to study how well JavaScript lends itself to programming in an object-oriented fashion.

The project suggested that Ajax is a very good technique to create desktop-like web applications. The latter part of this thesis helped in learning to use Ajax in simple game programming. HTML4 worked well in game programming with certain limitations, but the upcoming HTML5 seemed to be a more reasonable choice for future web games. The publication of the game was yet to be accomplished because the finalization and testing had not been completed. However, the final release of the application is going to take place in the near future.

Keywords	Ajax, game programming, web programming, HTML5

Sisältö

Tiivistelmä

Abstract

1 Johdanto 6

2 Ajaxin historia 7

3 Ajaxin edut ja haitat 9

3.1 Etuja 9
3.2 Haittoja 12
3.3 Vaihtoehtona Richer Plugin 14

4 Ajax-sovellukset 15

4.1 Hakukentät 15
4.2 Validointi 16
4.3 Karttasovellukset 16
4.4 Muut sovellukset 17

5 Ajax-teknologiat 17

5.1 Ajax 17
5.2 JavaScript 18
5.2.1 LiveScript/JavaScript/JScript/ECMAScript 18
5.2.2 Olio-ohjelmointi 18
5.3 XHTML ja CSS 20
5.4 DOM 21
5.5 Pyyntömenetelmät 25
5.5.1 XMLHttpRequest 25
5.5.2 IFrame 28
5.6 Vastausformaatit 29
5.6.1 XML 29
5.6.2 JSON 31
5.6.3 Plain Text 32
5.7 REST 32

6 Palvelinohjelmointi Ajaxin kanssa 33

6.1 Palvelinteknologiat 33
6.2 Cross-domain-pyynnöt 34

7 Ajax-työkalut 34

8 Ajax-sovellusesimerkkinä peli 36

8.1 Peli-idea 36
8.2 Palvelin 37
8.2.1 Istunnot 37
8.2.2 Tietokanta 37
8.2.3 Vastaukset 38
8.3 Asiakas 39
8.3.1 Palvelimen kutsuminen 39
8.3.2 Vastausten käsittely 40
8.4 Pelin ohjelmointi 41
8.4.1 Vaihtoehtona HTML 5 41
8.4.2 Pelin aloittaminen 42
8.4.3 Liikkuminen 43
8.4.4 Lähtökulma ja voima 43
8.4.5 Tapahtumien päivitys 44
8.4.6 Kranaatti 44
8.4.7 Törmäykset 45
8.4.8 Räjähdykset 47
8.4.9 Örkit 48

9 Yhteenveto 49

Lähteet 51

Liitteet 54

Liite 1: Istuntomuuttujat 54
Liite 2: Tietokanta 55
Liite 3: MySQL-funktiot 57
Liite 4: PHP-skripti 58
Liite 5: AjaxCaller 60
Liite 6: Callback-funktiot 62
Liite 7: Game-luokka 64
Liite 8: Pelin aloittaminen 66
Liite 9: Mousedown ja mouseup 67
Liite 10: Liikkumistiedon hakeminen 68
Liite 11: Liikuminen 70
Liite 12: Päivityssykli 71
Liite 13: Grenade-luokka 75
Liite 14: Törmäykset 76
Liite 15: Räjähdys 79
Liite 16: Orc-luokka 80

1 Johdanto

Viime vuosina web-suunnittelussa on entistä enemmän keskitytty nopeuteen, käytettävyyteen ja interaktiivisuuteen. Ilmiö luo haasteita web-sovellusten kehitykselle, koska internet on hidas ja staattinen väline johtuen HTTP-protokollan luonteesta. Kun ajatellaan mihin tarkoitukseen internet aikoinaan suunniteltiin ja käytettiinkin pääasiallisesti vielä 10–15 vuotta sitten, tiedon välittämiseen, niin voidaan vain ihmetellä miten käytössä voi olla vielä sama väline. Vaikka tiedon saanti on edelleen tärkeänä osana internetiä, asiat kuten viihdekäyttö, yhteyksien pitäminen ja luominen, markkinointi sekä tavaroiden ja palveluiden ostaminen, ovat vähintään yhtä tärkeitä. Viimeisimpinä "hypetyksessä" ovat olleet erilaiset sosiaaliset mediat (Facebook, Twitter) sekä video- ja musiikkipalvelut (Youtube, Spotify).

Tähän asti ollaan totuttu siihen, että web-sivut ovat staattisia elementtejä, ja aina kun jotain tehdään, joudutaan odottelemaan seuraavan sivun latausta. Toisaalta jokainen on käyttänyt myös perinteisiä työpöytäsovelluksia ja omaksunut niiden helpommin käytettävän ja nopeasti reagoivan luonteen. Mutta minkä takia Facebook on alkanut muistuttaa enemmänkin työpöytäsovellusta? Miksi Googlen kartalla liikkuminen on niin sulavaa? Miten sähköpostit voivat saapua reaaliajassa Gmailissa? Miten Google osaa arvata, mitä yritän hakea? Vastaus kaikkiin näihin kysymyksiin on Ajax. Tämän viime vuosina suuren suosion saavuttaneen tekniikan voisikin sanoa olevan yhdistävä tekijä työpöytäsovellusten käytettävyyden ja nopeuden sekä perinteisten web-sovellusten kevyyden ja vuorovaikutteisuuden välillä. Jotkut jopa väittävät, että selaimessa suoritettavat sovellukset tulevat korvaamaan työpöytäsovellukset hyvinkin pian [1].

Vaikka Ajax ei olekaan uunituoretta tekniikkaa, vasta viime vuosina on opittu hyödyntämään sen tuomia mahdollisuuksia. Koska Ajax on tällä hetkellä erittäin ajankohtainen ja jatkuvasti uusia ulottuvuuksia saava tekniikka, soveltui se mielestäni hyvin insinöörityön aiheeksi. Tämän työn tarkoituksena on luoda yleiskatsaus Ajax-

tekniikkaan, vertailla tekniikan hyviä ja huonoja puolia sekä opastaa muita web-ohjelmointiin jo tutustuneita Ajax-sovellusten maailmaan. Työssä selvitetään, mitä teknologioita Ajaxin takaa löytyy ja miten niitä voidaan hyödyntää. Lisäksi pohditaan, mitkä internetin palvelut jo Ajaxia käyttävät ja millaisiin sovelluksiin ylipäätänsä Ajax on omiaan.

Työn jälkimmäisessä osassa sovelletaan näitä tekniikoita ja kehitetään alusta loppuun toimiva pelisovellus Ajaxia ja dynaamista sivun muokkausta apuna käyttäen. Tarkoituksena on selvittää Ajaxin käyttökelpoisuutta peliohjelmoinnin apuna. Peliohjelmoinnissa yleensä tärkeää on myös grafiikan käyttö. Kolmannen osapuolen teknologiat ovat mahdollistaneet grafiikan luomisen helposti, mutta tässä työssä tarkoituksena on selvittää, kuinka dynaamisesti muuttuvaa grafiikkaa voidaan käyttää täysin suositusten mukaisten HTML-sivujen ohjelmoinnissa. Olio-ohjelmointi on tärkeää laajempien projektien tekemisessä, joten sitä sovelletaan myös tässä työssä. Lisäksi pohditaan Ajaxin tulevaisuutta ja katsotaan, mitä uutta ja parempaa on luvassa.

2 Ajaxin historia

Ennen vuosituhannen vaihdetta oli jo luksusta, että jostain kaupasta pystyi tilaamaan tuotteita internetin välityksellä. Nopeudella ja käytettävyydellä ei ollut mitään merkitystä itse palvelun ollessa tärkeintä. Nykyään verkkokauppa on palveluna jo itsestäänselvyys. Verkkoon on mentävä, jos aikoo tavoitella vähäänkään suurempia markkinoita. Kilpailu on kovaa, ja erottuaakseen joukosta palvelun on oltava helppokäyttöinen, nopea ja visuaalisesti hieno.

Toki mahdollisuudet toteuttaa edellä mainitun kaltainen palvelu ovat nousseet radikaalisti viime vuosien aikana. Valmiita kauppasovelluksia löytyy vaikka millä mitalla, ja vaikka kaikki tehtäisiinkin alusta asti, sovelluksen kehittäminen on paljon nopeampaa ja helpompaa. Valmiiden työkalujen, kehysten (framework) ja muiden kirjastojen kirjo on valtava. Asiakaspuolella muun muassa selainten laajennusosat, kuten Adoben Flash, ovat vakinnuttaneet paikkansa osana internetiä. On myös

muistettava, että HTML- ja CSS-standardit sekä JavaScript ovat myös kehittyneet ensimmäisistä versioistaan, puhumattakaan siitä että pelkästään tehokkaammat tietokoneet ovat mahdollistattaneet entistä rikkaampien ja monipuolisempien sovellusten rakentamisen. Myös palvelinpuolen teknologiat ja ohjelmointikielet, kuten PHP, Java EE tai Microsoftin ASP.NET, ovat olleet osana "internetin vallankumousta".

Viimeisin villitys asiakaspuolen sovellusten kehittämisessä on nimeltään Ajax. Vaikka Ajax on monissa web-sovelluksissa vasta viime vuosina otettu laajalti käyttöön, juontaa sen juuret jo 1990-luvun puolelle, Microsoftin leiriin. Alex Hopmann mainitsee blogissansa työskennelleensä työryhmässä kehittämässä Outlookin web-versiota. Hän kertoo kirjoittaneensa ensimmäisen version Ajaxin ydinkomponentista, joka nyttemmin tunnetaan nimellä XMLHttpRequest. Outlookin web-version ajatuksena oli mahdollistaa oman sähköpostiosoitteensa käyttäminen miltä tahansa selaimelta käsin. Tämä ei ollut mahdollista muuten kuin saamalla kyseinen komponentti kiinteäksi osaksi Internet Exlorer -selainta. Hopmann oli ollut kehittämässä myös XML:ää, ja hänellä oli hyvät suhteet Microsoftin XML-työryhmän kanssa. Niinpä hän keksi ehdottaa komponenttia MSXML-kirjastoon lisättäväksi. Hän tunnustaa antaneensa komponentille nimen XMLHTTP juuri siitä syystä, että saisi myytyä ideansa työryhmälle. [2.]

XMLHTTP oli ensimmäisenä käytössä Internet Explorerin versiossa 5 [3]. Mozilla kehitti tämän pohjalta natiivin version XMLHttpRequestista jo 2000-luvun alussa [4], mutta W3C:n virallinen suositus ilmestyi vasta 2006 [5]. Nykyään ominaisuus on implementoitu käytännössä kaikkiin selaimiin. Vaikka XMLHttpRequest on ehkä Ajaxin tärkein komponentti, se ei ole kuitenkaan sama kuin Ajax. Ajax on enemmänkin ajattelutapa rakentaa dynaamisia web-sivuja, ja tämän idean vieminen eteenpäin vei aikaa.

Kesti kauan ennen kuin sovelluskehittäjät omaksuivat uuden teknologian. Vaikka teknologia oli jo olemassa, Ajax alkoi nousta vasta vuosien päästä. Hopmann arvioi

yhden syyn olleen silloisten internet-yhteyksien hitaus ja erityisesti tietokoneiden tehottomuus. Koska enemmän työtä annetaan selaimen hoidettavaksi, vaatii se toki myös enemmän prosessointitehoa. [2.]

Vähäisimpänä syynä ei varmasti ollut myöskään se, että Ajaxia ei vielä osattu käyttää. Ei tiedetty, mitä sillä voisi tehdä ja missä sitä voisi hyödyntää. Ajax myös muutti perinteisen web-sovelluskehityksen prosessia, ja senkään takia kaikki eivät heti sille tielle uskaltaneet lähteä [6]. Mutta kun aika kului, kaikki muuttui. Ehkä suurimpana suunnannäyttäjänä oli internet-jätti Google, jonka Gmail ja Maps olivat ensimmäisiä Ajax-ideologiaa noudattavia sovelluksia [7].

Vaikka Ajaxia oli käytetty jo vuosia, sille ei oltu keksitty vielä nimeä. Kunnes helmikuussa 2005 Adaptive Path -sivuston perustaja Jesse James Garrett antoi tekniikalle nimen artikkelissaan "Ajax: A New Approach to Web Applications" [8]. Termi Ajax ponnahti Garrettin päähän suihkussa, kun hän mietti kuinka selittäisi asiakkailleen Googlen uusimpien sovellusten joustavuutta [9]. Garrettin artikkelin jälkeen Ajax on levinnyt kuin rutto, jos näin erinomaisesta keksinnöstä niin kehtaa sanoa.

3 Ajaxin edut ja haitat

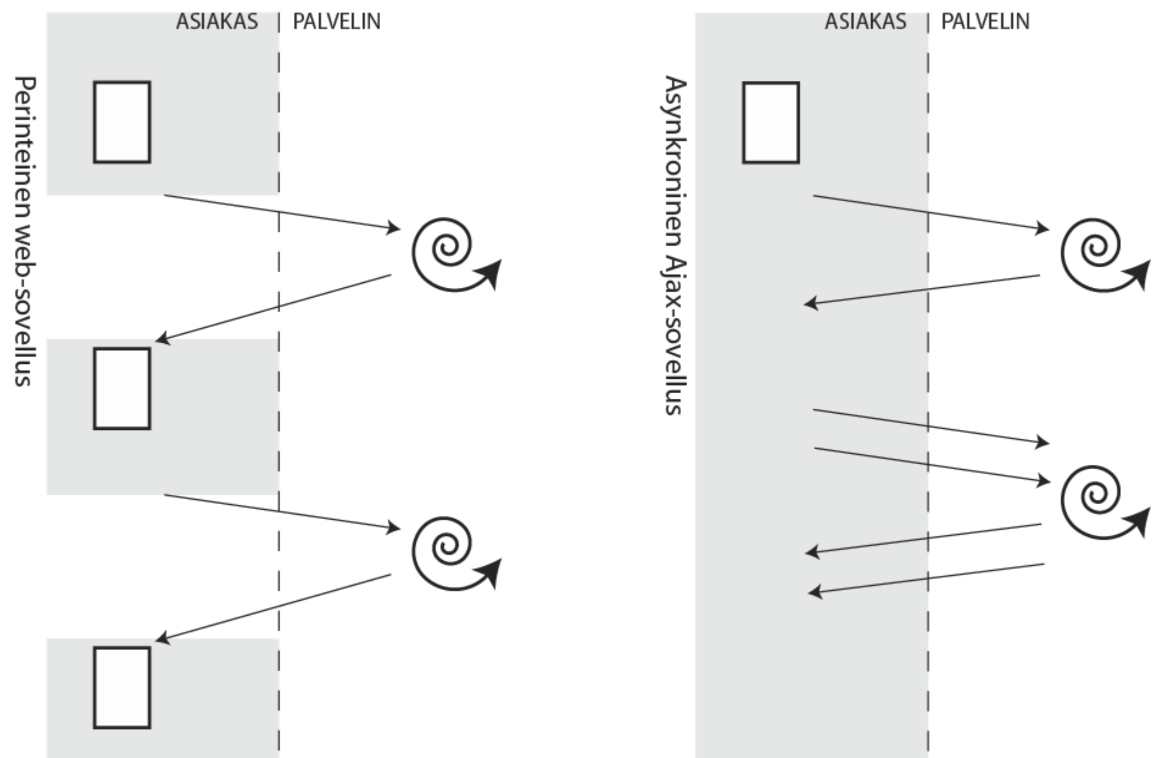
3.1 Etuja

Mitä uutta käytännössä Ajax tuo perinteisiin web-sovelluksiin verrattuna? Ensinnäkin Ajax sanoo hyvästi jatkuville sivun uudelleenlatauksille. Jos esimerkiksi kyseessä on urheilutuloksia reaaliajassa päivittävä sivusto, perinteinen web-sovellus joutuisi lataamaan koko sivun uudelleen aina kiintein väliajoin hakeakseen päivitetyt tulokset palvelimelta. Tämä näkyisi käyttäjälle sivun vilkkumisena ja uuden sivun latauksen odotteluna. Mutta miksi ladata koko sivua uudestaan, jos vain pieni osa siitä muuttuu? Tätä varten on Ajax. Ajaxin avulla vain päivitettävä osa tiedoista voidaan hakea palvelimelta taustalla ja muokata HTML-sivua näyttämään muutokset.

Koska koko sivua ei noudeta, vähentää se merkittävästi tarvittavaa tiedonsiirtoa asiakkaan ja palvelimen välillä. Tiedonsiirtoa voi vielä entisestään vähentää kysymällä palvelimelta, onko mitään tapahtunut sitten viimeisen päivityksen. Jos on, palvelin voi lähettää vain tarvittavat päivitykset. Vastaava kysely voidaan hyvin tehdä hyvinkin tihein aikavälein, koska tarvittava tiedonsiirtomäärä on erittäin pieni.

Nopeuden ja tehokkuuden lisäksi Ajaxille ominaista on sen asynkroninen luonne. Tietoa haetaan ja päivitetään silloin kun tarvitaan reagoiden käyttäjän tekemiin valintoihin. Tämä tekee web-sivuista entistä käyttäjäystävällisempiä ja vuorovaikutteisempia.

Perinteisen ja asynkronisen Ajax-sovelluksen välistä eroa on selvennetty kuvassa 1. Harmaa alue kuvassa edustaa aikaa, jolloin sivun vuorovaikutteinen käyttäminen on mahdollista. Klassisen web-sovellukseen käyttö on katkonaista ja hidasta, koska uuden sivun latautumista joudutaan aina odottamaan käyttäjän joutuessa toimettomaksi. Ajax-sovelluksessa käyttö ei katkea lainkaan. Kun uutta tietoa haetaan, käyttäjä voi jatkaa sivun käyttämistä keskeyttämättä ajatusprosessiaan. Reagointi käyttäjän aktiviteettiin on muutenkin huomattavasti nopeampaa.



Kuva 1. Klassisen ja asynkronisen Ajax-sovelluksen erot.

Ajaxia voi käyttää hyvin myös monimutkaisten tehtävien ratkaisun apuna. Jos selaimella tulee suorittaa hieman monimutkaisempi tehtävä, käyttäjän tekemä laskutoimitus tai piirtää vaikkapa kuvaaja sellaisesta, tehtävä voi olla liian vaikea tai hidas selaimelle. Niinpä tehtävän voi siirtää ratkaistavaksi palvelimelle, jossa resurssit ovat paremmat. Ainoastaan vastaus lähetetään takaisin käyttäjälle hänen tietämättään edes koko tiedon vaihdosta.

Erilaisia tehtäviä voi siis siirtää palvelimen hoidettavaksi. Ajaxin ansiosta asiakkaan ja palvelimen välillä on oma kommunikaatiokanava. Päätettäväksi jää vain se, kuinka suuri osa logiikasta haluataan toteuttaa milläkin puolella (niin sanottu Thin Client–Fat Client -ajattelutapa [10, luku 13.5]).

Ajaxin jopa ehkä tärkein etu on se, että Ajax on jo valmiina osana selainten tekniikkaa. Toisin kuin esimerkiksi Flash-sovelluksissa, erillistä lisäkomponenttia ei siis tarvita.

Tekniikka on turvallista käyttää, koska se on yleisesti hyväksytty. Ajaxin voisi sanoa olevan jopa työpöytävelluksiakin turvallisempi [9]. Tämä johtuu siitä, että web-sovelluksia ei ladata koneelle, vaan ne suoritetaan palvelimella web-käyttöliittymän toimiessa vain rajapintana asiakkaan ja palvelimen välllä. Ohjelman logiikka voi olla piilossa palvelimella. Ainoa huolenaihe on se, että tämä – joskus arkaluontoinenkin – tieto matkustaa internetissä alttiina salakuuntelulle ja muille uhille. Tärkeää on siis pitää huolta myös tietoliikenteen turvallisuudesta. Se on kuitenkin tämän työn aihepiirin ulkopuolella. Ajaxin tietoturvaan pätevät joka tapauksessa tutut säännöt, eli esimerkiksi SSL/TLS-protokollan käyttäminen on täysin mahdollista.

3.2 Haittoja

Voiko Ajaxista olla jotain haittaa? Yksi huomionarvoinen asia ei sinänsä ole mikään vika, vaan liittyy lähinnä käyttäjien tottumuksiin. Vanha tottumus on, että mitään ei tapahdu, jos sivu ei lataudu uudelleen. Todellisuudessa sivulle voi päivittyä koko ajan uutta tietoa käyttäjän huomaamatta sitä. Siksi usein käytetään keinotekoisia latausanimaatioita ja -palkkeja ilmaisemaan, että "jotain tapahtuu".

Koska Ajaxia käytettäessä pysytään aina samalla sivulla, aiheuttaa se yleensä selaimen back-napin toimimattomuuden. Se ei välttämättä ole suuri ongelma, mutta on myös vastoin käyttäjien tottumuksia. Back-napin oletetaan toimivan, ja sillä päästävän aina edelliselle sivulle. Tämä ongelma on kuitenkin ohjelmoijan ratkaistavissa, jos se koetaan tarpeelliseksi.

Jotain vielä pahempaakin on luvassa, ainakin jos on uskominen ohjelmointianalyytikko Earle Castledineen. Hän kritisoi Ajaxia artikkelissaan "Using the XMLHttpRequest Object and AJAX to Spy On You" [11]. Hän kertoo hieman sarkastisella kirjoitustyylillä Ajaxin tuomista mahdollisuuksista vakoiluun käyttäjän täysin tietämättä siitä.

Castledine sanoo ongelman muodostuvat lähinnä siitä, miten ihmiset ovat tottuneet perinteiseen internetin käyttöön. Internetiä pitkään käyttänyt "mattimeikäläinenkin"

tietää, miten internet toimii, ei teknologiselta kannalta, vaan käytännön kannalta. Jokainen esimerkiksi tietää, ettei lomakkeen lähetyspainiketta kannata painaa kuin kerran. Suurin osa ihmisistä myös olettaa, ettei mitään tapahdu silloin kun mitään ei näy tapahtuvan ruudulla. Tämä on kuitenkin Ajaxin aikakaudella harhaluuloa. [11.]

Castledine käyttää esimerkkinä tapausta, jossa henkilö on rikkonut uuden iPod-soittimensa pudottamalla sen alas portaista. Hän on lähettämässä Applelle reklamaatiota asiasta. "Ostin juuri uuden iPodin. Se meni rikki kun pudotin sen...", hän kirjoittaa, kunnes päättää korjata jälkimmäistä lausettaan sopivammaksi. Tämä on kuitenkin liian myöhäistä. Aikasempi viesti onkin lähtenyt jo taustalla eteenpäin. [11.]

Michael Mahemoff, Ajax Design Patterns -kirjan kirjoittaja ja samannimisen internet-sivuston ylläpitäjä, kommentoi blogissaan edellä mainittua tapausta. Hän muistuttaa, vaikka tämä teoriassa olisikin täysin mahdollista ja Apple saisi tietoonsa alkuperäisen viestin, käyttäjä ei voi olla mitenkään vastuussa kirjoittamastaan, jollei hän ole itse tiedostanut viestin lähettämistä [12].

Mahemoff yrittää vierittää syytä pois Ajaxin niskoilta toteamalla, että vastaavanlainen "salakuuntelu" on yhtä mahdollista myös perinteisin keinoin, ilman Ajaxia. Mikään ei estä tallentamasta lomakkeen tilaa muutaman sekunnin välein taulukkoon, jonka sisältö lähetetään palvelimelle vasta lomakkeen lähetyksen yhteydessä. [12.]

Kaikkihan on kuitenkin kiinni sivuston ylläpitäjän rehellisyydestä. Miksei hän voisi tallentaa jokaisen kirjautumisen yhteydessä käyttäjän nimeä ja salasanaa omiin tarkoituksiinsa? Kuinka moni loppujen lopuksi käyttää samaa käyttäjänimeä ja salasanaa useaan eri palveluun? Epärehellinen webmaster voisi testata saamaansa listaa tunnuksista muihin suosittuihin sivuihin ja näin ollen päästä käsiksi henkilökohtaisiin tietoihin. Vastuu on myös käyttäjillä, kaikkiin sivuihin ei siis kannata luottaa, vaan järjen käyttö on aina sallittua internet-olosuhteissa.

3.3 Vaihtoehtona Richer Plugin

Vaikka Ajax tekee sovelluksista tavallisia web-sovelluksia monipuolisempia, voidaan joskus tarvita vieläkin enemmän. Richer Pluginilla tarkoitetaan jotain normaalin web-teknologian ulkopuolelle jäävää teknologiaa. Näitä teknologioita ovat muun muassa valmiit selainten lisäosat, kuten Java tai Flash, sekä sivustokohtaiset lisäosat. Näiden avulla voidaan web-sovellusten käyttöön saada lisää ominaisuuksia, kuten [10, l. 8.1]

- muuttaa selaimen käyttäytymistä (esimerkiksi lisäpalkit)
- kirjoittaa tiedostoihin käyttäjän koneelle
- käyttää ääntä ja parempaa grafiikkaa
- hyödyntää käyttäjän mikrofonia, webbikameraa yms. tai
- pitää nopeampaa yhteyttä palvelimeen (käytössä ei vain HTTP-protokolla).

Vaikka edellä mainitut ominaisuudet voivat kuulostaa houkuttavilta, aiheuttavat ne kuitenkin myös paljon ongelmia. Jotta käyttäjä voisi niistä hyötyä, täytyy hänellä olla kyseinen lisäosa asennettuna. Ja jotta hän suostuisi sellaisen asentamaan, hänellä pitää olla täydellinen luottamus sivuun. Lisäksi ei ole järkevää rajoittaa käyttäjiä tietyn selaimen käyttäjiin, vaan lisäosista pitäisi olla tehtynä oma versionsa jokaiseen selaimeen.

Helpointa on käyttää jo suosittua lisäosaa, kuten Flash, joka on jo tehty kaikille valtavirtaselaimille ja on useimpiin asennettuna. Flashin levinneisyydestä kertoo hyvin se, että Adoben mukaan Flash on asennettuna jopa 98 prosenttiin internetiin liitetyistä koneista [13]. Toimivuutta ei voida kuitenkaan taata jokaisessa selaimessa, esimerkiksi Applen iPhone ei tue Flashiä. Asiantuntijat jopa väittävät, että Flashin dominoinnin päivät alkavat olla luetut [13]. Syinä tähän ovat muun muassa mainittu Applen vastahakoisuus Flashia kohtaan, ja erityisesti tuleva HTML5-standardi, joka parantaa mahdollisuutta tehdä entistä interaktiivisempia sovelluksia, ja vielä standardinmukaisesti ilman kolmannen osapuolen lisäosia [13]. Miksei siis käytettäisi tekniikkaa, joka toimii varmasti kaikilla? Miksei käytettäisi Ajaxia?

4 Ajax-sovellukset

4.1 Hakukentät

Jos on vähäänkään käyttänyt internetiä viimeisten vuosien aikana, ei Ajax-sovelluksiin ole voinut olla törmäämättä. Kaikki ovat varmasti käyttäneet Googlen hakua, Google Mapsia, Gmailia, Youtubea, Facebooka, Flickriä tai Twitteriä? Internetin suosituimpien palvelujen ohella myös tuhannet muut sivustot hyödyntävät Ajaxia.

Varmasti kaikkein yleisin tapa käyttää Ajaxia ovat hakukentät. Käyttäjän kirjoittaessa hakua hakukenttään, sivu voi käydä vaikka jokaisen näppäimen painalluksen jälkeen hakemassa palvelimelta hakuehdotuksia. Tämän toimintamallin ovat omaksuneet lukemattomat sivustot hakutoimintoihinsa, tunnetuimpana näistä Google. Google ei tosin käy aivan joka näppäinpainalluksen välissä palvelimella, vaan käyttää parempaa menetelmää, niin sanottua Submission Throttling -suunnittelumallia [10, luku 10.3]. Näppäinpainallukset menevät jonoon välimuistiin, ja aina määrätyn ajan välein jonon sisältö lähetetään palvelimelle [14]. Ripeän kirjoittajan kirjoittaessa useamman kirjaimen intervallien välissä, vain yksi pyyntö lähtee palvelimelle [14]. Lisää optimointia on saavutettu historiatiedon tallentamisella. Askelpalautinta käytettäessä vanhat hakuvaihtoehdot palautetaan välimuistista, joten turhia kutsuja palvelimelle ei tästäkään synny [14].

Interaktiiviset hakukentät voidaan viedä vielä pidemmälle. Mitä jos hakuvaihtoehtoja ei vain ehdotettaisi, vaan haun tulokset näytettäisiin välittömästi? Tämänkaltaisen toiminnallisuuden tekemisessä kannattaa kuitenkin käyttää harkintaa. Yksinään käytettynä se aiheuttaa yleensä enemmän sekavuutta kuin löytämisen helppoutta, mutta oikein käytettynä se voi olla korvaamaton. Esimerkiksi Hintaseuranta.fi käyttää hakukentässään Googlen tapaista ehdottavaa hakua, mutta hakutuloksia onkin mahdollista haun jälkeen rajata edelleen raahaamalla hintapalkista ylä- ja alarajan tuotteen hinnalle. Hakutulokset päivittyvät dynaamisesti Ajaxin avulla. Samankaltaista ideaa käyttää myös esimerkiksi Etuovi.com. Hakua voi rajata muun muassa hinnan,

asunnon pinta-alan ja rakennusvuoden perusteella, ja alueelta löytyvien kohteiden lukumäärä päivittyy aina dynaamisesti sivulle.

4.2 Validointi

Lähes kaikki sivut, jotka vaativat kirjautumista, käyttävät rekisteröitymislomakkeessaan reaaliaikaista validointia eli syötettyjen tietojen tarkastamista. Osa tiedoista toki pystytään helposti validoimaan myös asiakaspuolella, kuten esimerkiksi annetun salasanan vahvuus. Kuitenkin esimerkiksi tieto siitä, onko käyttäjänimi jo käytössä, on mahdotonta sanoa, ellei listaa kaikista käytössä olevista käyttäjistä ladata joka rekisteröitymislomakkeen yhteydessä. Se on tietenkin järjetöntä. Käyttäjän siirtyessä seuraavaan kenttään voidaan syötetty käyttäjänimi lähettää taustalla palvelimelle, joka käy etsimässä tiedon tietokannasta. Vastaukseen voi riittää pelkästään yhden bitin kokoinen totuusarvo. Lopullinen validointi kannattaa kuitenkin aina tehdä lomakkeen lähettämisen yhteydessä, jotta tiedot tulevat varmistetuiksi myös esimerkiksi selaimilla, joilla JavaScript ei ole käytössä.

4.3 Karttasovellukset

Googlen, Microsoftin, Yahoon, Eniron ja Fonectan karttasovellukset käyttävät kaikki apunaan Ajaxia. Karttojahan pystyy selaamaan täysin vapaasti ilman sivun uudelleenlatauksia. Karttasovellukset toimivat niin, että kartta on jaettu tietyn kokoisiin palasiin (yleensä 256 x 256 pikseliä) ja palaset on sijoitettu saumattomasti muodostamaan yhtenäisen kartan. Käyttäjän liikuttaessa näkymäänsä tieto välittyy palvelimelle, joka lähettää uusia palasia täyttämään koko näkymän.

Toteutus on loppujen lopuksi hämmästyttävän yksinkertainen. Analysoimalla Google Mapsin koodia nähdään, että kartalle piirtyvät reittiohjeet ovatkin SVG-grafiikkaa, jonka linjatiedot saadaan palvelimelta. Vielä erikoisemmalta kuulostaa se, että kun SVG-tuen ottaa pois päältä selaimen asetuksista, reitit välittyvätkin läpinäkyvinä PNG-kuvina, jotka generoituvat palvelimella lähes reaaliajassa.

4.4 Muut sovellukset

Mahdollisuus keskusteluun palvelimen kanssa, yhdistettynä dynaamisen sivun käsittelemiseen, antavat edellytykset vaikka mihin. Miksei kauppasovelluksessa voisi esimerkiksi tuotteen klikkaamisen sijaan vain raahata sen ostoskoriin, jonka tila päivittyisi aina palvelimelle asti? Tuote voisi näin ollen olla varattuna ja vähentyä varastosaldosta siksi aikaa, kunnes asiakas tekee päätöksen ostamisesta. Tai miksei käyttäjää voisi antaa itse määrittää uutissivun sisältöä sisältämään vain häntä kiinnostavia aiheita? Uusia kategorioita voisi siirtää sivupalkista haluttuun paikkaan etusivulla, johon tuoreimmat uutiset päivittyisivät automaattisesti. Tai miksei valintalaatikosta valitsemisen sijaan voisi sääsivustolla vain siirtää hiiren osoittimen haluamansa kaupungin kohdalle? Alueen sen hetkinen säätila ilmestyisi välittömästi kartalle. Monia näistä ideoista on jo toteutettu ja uusia keksitään koko ajan. Vain mielikuvitus on rajana.

5 Ajax-teknologiat

5.1 Ajax

Tähän mennessä ollaan puhuttu vain Ajaxista, mutta mitä tämä pesuaineesta ja hollantilaisesta jalkapallojoukkueesta muistuttava termi oikein sisältää? Ajax on alun perin tullut sanoista Asynchronous Javascript and XML. Tosin ei XML:n tai edes Javascriptinkaan käyttö ole pakollista (joskin erittäin suotavaa), ja itse asiassa tiedonsiirronkaan ei tarvitse olla asynkronista. Siksi termiä voidaankin kutsua pseudo-akronyymiksi eli lyhenteeksi, joka ei tarkoita oikeastaan mitään. Ajax ei ole oma teknologia tai standardi, vaan kokoelma erilaisia teknologioita ja tekniikoita nidottuna yhteen [8].

5.2 JavaScript

5.2.1 LiveScript/JavaScript/JScript/ECMAScript

Ensimmäinen selain, joka tuki mitään skriptauskieltä oli Netscape 2. Kielen nimi oli LiveScript. Pian nimeksi vaihdettiin JavaScript toivoen Javan suosion siivittävän omaa suosiotansa. Javallahan ei todellisuudessa ollut mitään tekemistä JavaScriptin kanssa, ja samankaltaiset nimet ovatkin pitkään aiheuttaneet sekaannusta ainakin aloittelevien ohjelmoijien keskuudessa. Microsoft vastasi kehittämällä Internet Explorer 3:een JavaScriptiä erittäin läheisesti muistuttavan JScript-kielen. [15.]

Kun Internet Explorer alkoi syödä Netscapen markkinaosuutta, piti pysyvä ratkaisu kehittää. Netscape antoi ohjakset ECMA:lle (European Computer Manufacturers Association), joka kehitti yhteisen kielen, ECMASciptin. Suurin osa eroavaisuuksista selaimien käsittelyssä hävisi. Eroavaisuuksia Microsoftin ja muiden selaimien välillä on edelleen, mutta ne pystytään yleensä hoitamaaan pienellä vaivalla. [15.]

Ennen Ajaxin nousua JavaScriptin suosio oli laskemassa. Sen todellisia mahdollisuuksia ei oikein osattu hyödyntää, ja monien mielestä se ei tuntunut kelpaavan kuin joidenkin pienten askareiden hoitamiseen. Sen sijaan palvelinpuolen kielet, kuten PHP, niittivät suosiota entistä enemmän. Kaikki kuitenkin muuttui Ajaxin rantauduttua. JavaScript sai siitä uutta potkua. JavaScriptin osa Ajaxissa on ikään kuin sitoa kaikki tekniikat yhteen. Se on edellytys dynaamisen Ajax-sovelluksen luomiseen.

5.2.2 Olio-ohjelmointi

Vaikka JavaScript ei ole täysin oliopohjainen kieli, on silti mahdollista noudattaa olio-ohjelmoinnin ajattelutapaa. Oliot koostuvat ominaisuuksista, ja toisin kuin esimerkiksi C++:ssa, näitä ominaisuuksia ei tarvitse määritellä etukäteen, vaan niitä voi luoda lisää jo luotuun olioon [16]. Luokkia voi määritellä yksinkertaisesti määrittelemällä uuden funktion. Funktiosta tulee luokan konstruktori siinä vaiheessa, kun uusi olio luodaan

siitä new-operaatiolla [16]. This-muuttuja alustetaan automaattisesti osoittamaan juuri luotua oliota [16], ja this-etuliitettä tuleekin käyttää aina luokan omiin muuttujiin ja metodeihin viitattaessa.

```
function ClassA()
{
   this.varA = "muuttuja";
}
var objA = new ClassA();
```

Jokainen olio voi periä ominaisuuksia toiselta oliolta, niin sanotun prototypen kautta. Prototype on myös oivallinen apu metodien ja periytymisen toteuttamiseen JavaScriptillä. Luokalle voi lisätä uuden funktion asettamalla sen funktion prototyypiksi [16].

```
function ClassB()
{
   this.value = 123;
}
ClassB.prototype.setValue = function(x)
{
   this.value = x;
}
var objB = new ClassB();
objB.setValue(456);
```

Koska JavaScriptissä ei voi määrittää luokalle todellisia yksityisiä metodeja tai ominaisuuksia, ne jäävät ohjelmoijan huomioitavaksi. Usein käytetään alaviiva-etuliitettä yksityisten ominaisuuksien määrittelemisessä. Tämä ei tietenkään estä käyttämästä kyseistä ominaisuutta muista luokista käsin, mutta voi helpottaa ohjelmoijan ajatustyötä. Tätä ajatusta on käytetty myös tämän työn esimerkkisovelluksen toteuttamisessa. Rajoituksia ei ole myöskään esimerkiksi isojen alkukirjainten käytössä, mutta yleensä on järkevää noudattaa muistakin ohjelmointikielistä tuttuja yleisiä nimeämiskäytäntöjä ja kirjoittaa esimerkiksi luokkien nimet isolla alkukirjaimella.

Intervallien käyttö

Intervallien ja timeoutien asettaminen JavaScriptissä on periaateessa helppoa, mutta jos käytetään luokka-rakennetta, joudutaan intervallien käsittelyssä käyttämään hieman normaalista poikkeavaa tapaa. Jos setInterval()-funktiolla halutaan kutsua olion metodia, this-määrittely ei toimi, vaan se viittaakin virheellisesti oman olionsa (window) vastaavaan metodiin [17].

Firefoxissa setInterval()-funktiosta on käytössä paranneltu versio, joka hyväksyy kolmannen parametrin. Siinä voidaan välittää viittaus haluttuun olioon ja kutsua metodia sitten tämän viittauksen kautta [17]. Internet Explorer ei kuitenkaan kolmatta parametria hyväksy. Viittaus olioon voidaan joka tapauksessa tallentaa globaaliin muuttujaan, jonka kautta intervallein suoritettavaa metodia kutsutaan. Sama toimii myös Firefoxissa, joten sitä voitanee käyttää yhteisesti. Tätä tapaa, olio-ohjelmoinnin ohella, käytettiin myös tämän työn pelisovelluksen toteutuksessa.

```
scope = this;
_timer = setInterval('scope._update()', 1000);
```

5.3 XHTML ja CSS

HTML on internetin peruspilari. Mitään sivua ei oikeastaan voi rakentaan ilman HTML:n käyttöä. Niinpä siis HTML, tai usein muodollisempi XHTML, on osana Ajaxin käyttöä. CSS-tyylit puolestaan määrittelevät tarkemmin sivun rakenteen ja visuaalisen ilmeen. Onneksi nykyään kaikki selaimet osaavat käsitellä CSS-tyylejä lähes samalla tavalla, mikä on edesauttanut siirtymistä taulukkorakenteisista web-sivuista CSS-tyyleillä määriteltyihin.

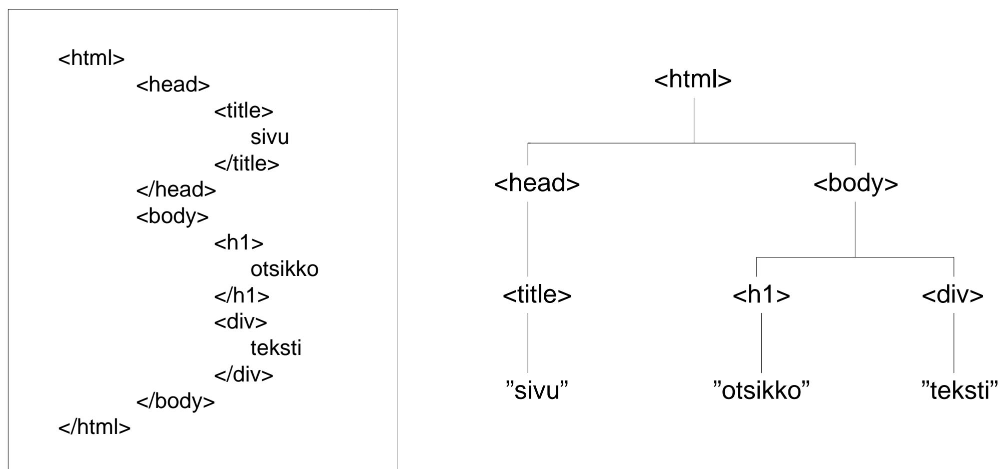
CSS:n avulla voidaan määritellä elementien sijainteja ja tyylejä, kuten värejä, taustoja, reunuksia, läpinäkyvyyttä ja fontteja. Myös niiden muuttaminen JavaScript-koodin

kautta on mahdollista, ja onkin erittäin suuressa osassa Ajaxin käytössä (Display Manipulation -suunnittelumalli [10, luku 5]).

5.4 DOM

Perinteisille web-sivuille ominaista on niiden staattisuus, eli ne pysyvät samana koko ajan, kunnes seuraava sivu ladataan. Koska Ajaxin ajatuksena on pysyä koko ajan samalla sivulla muuttaen sen sisältöä aina tarvittaessa, tarvitaan työkalu sivun rakenteen muokkaamiseen.

Document Object Model, tai tuttavallisemmin DOM, esittää dokumentin puurakenteen muodossa. Kaikkien XML-muodossa olevien dokumenttien rakenne voidaan esittää DOMilla. Jokaista solmukohtaa tai haaraa kutsutaan solmuksi (node). Jokaisesta solmusta jakautuvat solmut ovat kyseisen solmun lapsia (child) ja toisilleen sisaria (sibling). Ylemmällä tasolla olevaa solmua sanotaan loogisesti vanhemmaksi (parent). Koska XHTML on XML:ää, voidaan web-sivujenkin rakenne esittää puumuodossa DOMin avulla, kuten kuvasta 2 nähdään.



Kuva 2. HTML-sivu ja sen DOM-malli.

DOMiin pääsee käsiksi monilla skriptaus- ja ohjelmointikielillä. Mutta koska tässä työssä käsitellään Ajaxia, on esimerkit toteutettu JavaScriptillä. Solmuilla on paljon ominaisuuksia ja metodeja, joista yleisimpiä lueteltu taulukossa 1.

Taulukko 1. Node-luokan ominaisuuksia ja metodeja [18].

childNodes	Palauttaa listan lapsisolmuista.
parentNode	Palauttaa solmun ylemmällä tasolla olevan solmun.
firstChild	Palauttaa solmun ensimmäisen lapsen.
lastChild	Palauttaa solmun viimeisen lapsen.
nextSibling	Palauttaa solmun heti seuraavan sisarsolmun.
previousSibling	Palauttaa solmun heti edellisen sisarsolmun.
nodeName	Palauttaa solmun nimen, riippuen sen tyypistä.
nodeType	Palauttaa solmun tyypin.
nodeValue	Asettaa tai palauttaa solmun arvon, riippuen sen tyypistä.
appendChild()	Lisää uuden solmun viimeiseksi lapseksi.
hasChildNodes()	Palauttaa "true", jos solmulla on lapsia, muuten palauttaa "false".
insertBefore()	Lisää uuden lapsisolmun edelliseksi sisareksi.
removeChild()	Poistaa lapsisolmun.

Yksi tapa päästä käsiksi haluttuun solmuun on ottaa kiinni jostain solmusta, ja liikkua sitten puurakenteessa. Solmun ominaisuuksiin ja metodeihin viittaamalla pääsee helposti käsiksi sen sisariin, lapsiin ja vanhempaan. Kuvan 2 mukaisesta rakenteesta div:n sisältämä teksti voitaisiin saada selville esimerkiksi seuraavalla tavalla:

```
alert(document.firstChild.childNodes[1].childNodes[1].firstChild.nodeValue);
```

Koko dokumenttiin päästään käsiksi viittaamalla document-olioon. Dokumentin ensimmäinen lapsi on tässä tapauksessa html. Html:n toinen lapsi on puolestaan body ja bodyn div. Solmun sisältämä teksti on aina myös oma solmunsa, niin sanottu tekstisolmu, jonka tekstisisältö saadaan selville nodeValue-ominaisuudella.

Edellinen koodirivi Internet Explorerilla tulostaa sanan "teksti", niin kuin oli odotettavissakin. Kuitenkin sama koodinpätkä Firefoxilla antaakin vastaukseksi "otsikko". Mistä tämä johtuu? Jotkut selaimet käsittelevät kaikki välilyönnit,

rivinvaihdot ja muut tyhjät merkit (whitespace), omina solmuinaan. Bodylla onkin kolmen lapsen sijasta viisi, kun h1- ja div-elementtien edellä ja jälkeen oli käytetty rivinvaihtoja. Kaikki turhat merkit on toki mahdollista jättää pois, ja näin ollen koodi toiminee samalla tavalla eri selaimissa. Toisaalta ottamalla huomioon nämä selainten väliset eroavuudet ohjelmoinnissa, on tämäkin ongelma ratkaistavissa.

Onneksi kuitenkaan usein tällaista tietorakenteen läpi surffailua ei tarvita. Muihin solmuihin päästään käsiksi myös document-olion kautta. Document-olio edustaa koko dokumenttia. Sillä on joitain samoja ominaisuuksia kuin solmuilla, kuten childNodes, firstChild ja lastChild. Näiden lisäksi sillä on muita hyödyllisiä metodeita, joista tärkeimmät on esitetty taulukossa 2.

Taulukko 2. Document-luokan metodeja [18].

createAttribute()	Luo attribuuttinoden annetulla nimellä ja palauttaa sen.
createElement()	Luo elementtinoden.
createTextNode()	Luo tekstinoden.
getElementById()	Palauttaa elementtinoden, jossa annettu ID. Palauttaa "null" jos sellaista ei löydy.
getElementsByTagName()	Palauttaa NodeListan kaikista nodeista, joilla sama tagin nimi.

Osa näistä metodeista palauttaa nodesta periytyvän elementtisolmun. Sillä on solmun ominaisuuksien lisäksi myös samoja ominaisuuksia kuin dokumentilla. Koko dokumentin lisäksi voidaan myös siis hakea pelkästään elementtisolmun alla olevista solmuista.

```
var place = document.getElementById("place");     // löytää <div id="place"></div>
var divs = place.getElementsByTagName("div");     // noden alla olevat div:t
```

GetElementById() on ehkä kaikista käytetyin metodi, ja siksi usein käytetäänkin apufunktiota viittausten helpottamiseksi. JavaScript sallii lyhyen ja ytimekkään "$"-nimen käyttämisen. Näin ollen paitsi viittaaminen siihen on nopeaa, myös koodin koko voi pienentyä merkittävästi, jos viittausta on käytetty jopa satoja kertoja.

```
function $(id) { return document.getElementById(id); }
```

HTML-elementeillä on usein käytössä erilaisia attribuutteja. Niiden käsittely XML DOMin kautta on kuitenkin hankalaa, vaikkakin mahdollista. Helpompi tapa on kuitenkin käyttää apuna HTML DOMia. Se suo nopeampia tapoja solmun atribuuttien muokkaamiseen. Lisäksi CSS-tyylejä voi muokata suoraan style-atribuutin kautta.

```
$("place").id = "newplace";                     // vaihtaa id-attribuutin arvon
$("newplace").className = "places";            // asettaa class-atribuutin arvon
$("newplace").style.backgroundColor = "blue"; // vaihtaa taustavärin
$("newplace").innerHTML = "Hei Maailma";      // alku- & lopputagin väliin tuleva teksti
```

Joidenkin attribuuttien nimiä on muutettu sekaannuksien välttämiseksi, ja esimerkiksi className tarkoittaa elementin class-attribuuttia [19]. CSS-tyyleihin viitattaessa JavaScriptin puolelta on myös muistettava, että viiva-merkki on varattu vähennys-operaatioon. Näin ollen jälkimmäinen sana on kirjoitettava isolla alkukirjaimella. Esimerkiksi "background-color" muuntautuu muotoon "backgroundColor".

Atribuutit asettuvat siis näin helposti ilman, että niitä joudutaan erikseen luomaan createAttribute()-metodilla. XML DOMia käyttäen viimeisellä rivillä tulisi lukea "place.firstChild.nodeValue". Kyseinen rivi ei kuitenkaan toimi, jos nodelle ei ole vielä erikseen luotu tekstinodea lapseksi. Ratkaisuna olisi luoda ensin tämä tekstinode createTextNode()-metodilla. Helpompana vaihtoehtona on innerHTML-ominaisuus. InnerHTML sisältää koko elementin alku- ja lopputagin väliin jäävän tekstin. Muokkauksen jälkeen HTML-elementti näyttäisi seuraavalta:

```
<div id="place" class="divs" style="background-color: blue">Hei Maailma</div>
```

Joskus kuitenkaan pelkästään olemassa olevien elementtien muokkaus ei riitä. Elementtejä voi onneksi luoda helposti lisää. Luomisen jälkeen elementti pitää vielä sijoittaa sivun rakenteeseen. Toisaalta, sama lopputulos saadaan suoraan kirjoittamalla tagin sisältö merkkijonona ja lisäämällä se halutun solmun sisällön jatkoksi innerHTML:n avulla.

```
var body = document.getElementsByTagName("body")[0];
// vaihtoehto 1:
body.appendChild(document.createElement('div'));
// vaihtoehto 2:
body.innerHTML += "<div></div>";
```

DOMilla voisi ajatella olevan kaksi tärkeää tarkoitusta Ajaxin käytössä. Ensinnäkin palvelimelta saadaan tietoa usein XML-muodossa, DOM antaa keinot etsiä siitä tarvittavat osat. Toiseksi DOMin käsittelyn avulla voidaan muokata HTML-sivun rakennetta, jotta Ajaxilla haetut tiedot näkyvät loppukäyttäjälle.

5.5 Pyyntömenetelmät

5.5.1 XMLHttpRequest

Edellä mainitut teknologiat eivät kuitenkaan vielä sinänsä tuo paljoakaan uutta. Ajaxin ideaanhan kuuluu jatkuva palvelimen kanssa keskusteleminen. Tätä tehtävää voidaan laittaa hoitamaan XMLHttpRequest-olio tai lyhyemmin XHR-olio. XHR:ää voisikin sanoa Ajaxin ydinkomponentiksi. Myös XMLHttpRequest-nimeä voi pitää harhaanjohtavana. Kaikkia tekstipohjaisia tiedostomuotoja tuetaan, ei ainoastaan XML:ää, ja HTTP-protokollan lisäksi myös salatun HTTPS:n käyttö on mahdollista [19]. Lisäksi olion tehtävänä ei ole ainoastaan tietojen pyytäminen, vaan se hoitaa myös muun muassa tiedon vastaanoton [20].

XHR-olion luominen JavaScriptilla on periaatteessa hyvin helppo tehtävä. Niin sanottu cross-browser-tapa eli tapa, joka toimii lähes kaikissa selaimissa ja uusimmissa selainversioissa, on luoda natiivi versio XMLHttpRequest-oliosta. Internet Explorerissa tämä suosituksen mukainen tapa toteutettiin kuitenkin vasta versioon 7 [21]. Jos tuki myös tätä vanhemmille selaimille halutaan, täytyy versioita 5 ja 6 varten XHR-olio luoda käyttäen ActiveX-komponenttia. Niitä vanhemmissa Internet Explorer -versioissa XHR ei ole käytössä [21]. Jotta olio saadaan luotua selaimesta riippumatta, kannattaa natiivin XMLHttpRequest-olion olemassaoloa ensin testata:

```
if (window.XMLHttpRequest)
   var xhr = new XMLHttpRequest();
else   // IE
   var xhr = new ActiveXObject('MSXML2.XMLHTTP.3.0');
```

Taulukko 3. XHLHttpRequestin metodeja ja ominaisuuksia [20].

open()	Avaa yhteyden annetuin parametrein.
send()	Lähettää kutsun palvelimelle.
setRequestHeader()	Asettaa otsikkotietoja.
onreadystatechange	Sisältää funktion osoitteen, jota kutsutaan valmiustilan muutuessa.
responseText	Sisältää vastaustiedon tekstimuotoisena.
responseXML	Sisältää vastaustiedon XML-muotoisena.
readyState	Sisältää sen hetkisen valmiustilan.
status	Sisältää vastauksen HTTP-tilakoodin.

XMLHttpRequestin tärkeimmät metodit ja ominaisuudet on esitetty taulukossa 3. Kun olio on luotu, voidaan yhteys avata. Pyynnön lähettämiseen on käytettävissä kaikki HTTP-metodit; GET, POST, HEAD, PUT, DELETE ja OPTIONS [19]. Näistä useimmiten käytetään joko GET- tai POST-metodia. Lisäksi open()-metodille annetaan URL-osoite, johon pyyntö lähetetään. Kolmantena parametrina annetaan totuusarvona tieto siitä suoritetaanko pyyntö asyknronisena vai synkronisena. Jos URL vaatii käyttäjänimeä tai salasanaa, sijoitetaan ne viimeisiksi parametreiksi.

```
xhr.open("GET", "test.php", true);
xhr.open("GET", "test.php", true, "username", "password");
```

Ominaisuuden onreadystatechange arvoksi voidaan nyt asettaa niin sanottu callback-osoite eli funktio, jota kutsutaan aina valmiustilan muuttuessa. Valmiustila tarkoittaa vaihetta, jossa tiedon pyynnön prosessi sillä hetkellä on. Kaikki mahdolliset valmiustilat on lueteltu taulukossa 4. Tilan muuttuessa suoritettavan funktion voi halutessaan määritellä suoraan niin sanottuna inline-funktiona:

```
xhr.onreadystatechange = function()
{
   if (xhr.readyState==4 && xhr.status==200)
```

```
        alert(xhr.responseText);
}
```

Taulukko 4. XMLHttpRequestin readyState-ominaisuuden tilat [20].

0	UNSENT	Olio on luotu.
1	OPENED	Open()-metodi on onnistuneesti suoritettu.
2	HEADERS_RECEIVED	HTTP-otsikkotiedot vastaanotettu.
3	LOADING	Vastauksen sisältöä vastaanotetaan.
4	DONE	Tiedon vaihto on valmis tai jotain meni vikaan.

Callback-funktiossa yleensä tarkastetaan readyState- ja status-ominaisuuksien arvot. Funktio suoritetaan jokaisen tilamuutoksen jälkeen, mutta yleensä ainoa tila, joka kiinnostaa, on tila DONE. Status-ominaisuus sisältää arvon 0 silloin, kun valmiustila on UNSENT, OPENED tai DONE-tilassa on tapahtunut verkkovirhe tai muu keskeytys [20]. Muussa tapauksessa ominaisuus sisältää HTTP-tilakoodin [20]. Jos virheiden käsittelyä ei haluta tehdä, voidaan vain tarkistaa, onko vastaus OK eli tilakoodi arvoltaan 200.

Lopuksi pyyntö lähetetään palvelimelle send()-metodilla. Jos käytetty HTTP-metodi vaatii rungon, voi sen antaa parametrina. GET-metodi ei sitä vaadi, joten parametrina voi antaa tyhjän "null"-olion:

```
xhr.send(null);
```

Vaikka lähes kaikki selaimet tukevat XMLHttpRequestia, se ei ole vielä virallinen W3C-organisaation standardi, vaan vasta suositus (draft) [20], jonka kehitys jatkuu koko ajan. Tulevaisuudessa käyttöön otettavassa XMLHttpRequest Level 2 -suosituksessa on muun muassa mahdollisuus jäljittää tarkemmin tiedon lataus- ja lähetysprosessia kuuntelijoiden avulla sekä mahdollisuus pakottaa vastauksen MIME-tyyppi halutuksi [22]. Lisäksi suosituksessa vielä tarkemmin määrittelemättä on jonkinnäköinen tuki myös bittijonojen siirtoon [22], joten myös esimerkiksi kuvien siirto tullee olemaan mahdollista tulevaisuudessa.

5.5.2 IFrame

IFramea voidaan pitää vaihtoehtona XHR:lle. Tietojen vaihtaminen taustalla palvelimen kanssa onnistuu siis myös iFramen avulla. Ennen XHR:n yleistymistä selaimissa iFrame oli ainoa vaihtoehto Ajax-ideologiaa noudattavan web-sivun tekemiseen. Hyvänä esimerkkinä iFrameja hyödyntävästä sovelluksesta voidaan pitää jo useasti mainittua Google Mapsiä.

Perusidea iFramejen käytössä on seuraavanlainen. Luodaan aluksi iFrame sivulle ja annetaan sille onload-ominaisuudeksi funktio, jota kutsutaan aina, kun uutta tietoa on ladattu. Koska iFrame toimii vain apukehyksenä, eikä sitä haluta näkyviin, se voidaan asettaa tyylimuotoilulla näkymättömäksi. [10, luku 6.3.]

```
<iframe id="iFrame" onload="loadContent()" style="visibility: hidden; width: 0px;
height: 0px"></iframe>
```

Uutta sisältöä voidaan ladata iFrameen asettamalla iFramen osoitteeksi halutun resurssin osoite [10, luku 6.3]. Näin GET-kutsu lähtee palvelimelle. Sisällön latauduttua kutsutaan ennalta määriteltyä funktiota, jossa vastausta voidaan käsitellä.

```
document.getElementById("iFrame").src = "resource.php?param=1";

function loadContent()
{
   document.getElementById("result").innerHTML =
      document.getElementById("iFrame").contentWindow.document.body.innerHTML;
}
```

IFramen käyttöön liittyy kuitenkin useita ongelmia, joihin ei tässä sen enempää paneuduta. Yleensä suositeltavaa on käyttää varta vasten tähän tarkoitukseen tehtyä XMLHttpRequest-oliota.

Tässä vaiheessa voisi herätä kysymys, minkä takia esimerkiksi Google Mapsin karttakuvien latauksessa käytetään iFrame-tekniikkaa XHR:n sijasta. Vaikka iFrame

onkin kömpelömpi käyttää, on siinä jotain etuakin XHR:ään verrattuna. XHR kun ei vielä tue datajonojen siirtoa, eikä esimerkiksi kuvien siirtäminen suoraan XHR-olion kautta ole mahdollista. Kuvan osoitteen välittäminen toki on mahdollista, ja kuva voidaankin luoda koodissa tämän tiedon perusteella. Kuitenkin iFrame-vastaukset latautuvat kuin normaalit HTML-sivut, joten kuvienkin sisällyttäminen niihin onnistuu. Lisäksi iFrame kutsut muuttavat selaimen historiatietoja, joten back-napin käyttökin on mahdollista [23]. Tämä on erittäin hyödyllistä, kun esimerkiksi kartalla halutaankin palata edelliseen reittisuunnitelmaan. Yksi syy iFrame-tekniikan käyttöön saattoi yksinkertaisesti olla se, että kun Maps-sovelluksen kehitys alkoi, XHR ei ollut vielä käytössä kaikissa selaimissa, eikä sen tulevaisuudesta tarkemmin tiedetty. Näin ollen jo pitkään käytössä ollut iFrame saattoi tuntua varmemmalta vaihtoehdolta.

5.6 Vastausformaatit

5.6.1 XML

XML on paitsi vaihtoehto raskaammille tietokantasovelluksille, myös oivallinen formaatti tiedon välittämiseen. Palvelimelta tuleva tieto, ja joskus asiakkaalta lähteväkin, tulee kääriä johonkin ymmärrettävässä muodossa olevaan pakettiin. Siihen XML soveltuu erittäin hyvin. XML-muotoisesta vastauksesta on helppo ottaa kiinni XHR-olion responseXML-ominaisuudella.

Jos vastaanotettu tieto on XML-muodossa ja sen MIME-tyyppi on oikein määritelty, voidaan sitä käsitellä suoraan. Muussa tapauksessa se tulee jäsentää XML-muotoon. Useimmissa selaimissa on valmis XML-jäsennin tähän tehtävään, mutta kuten yleensä, Internet Explorer vaatii erityiskohtelua [24]. Siinä käytetään ActiveX-oliota ja sen loadXML()-metodia merkkijonon jäsentämiseen, kun muut selaimet käyttävät natiivia DOMParser-oliota [24].

```
if (window.DOMParser)
{
   var parser = new DOMParser();
```

```
   xmlDoc = parser.parseFromString(xhr.responseText,"text/xml");
}
else   // IE
{
   xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
   xmlDoc.async = "false";
   xmlDoc.loadXML(xhr.responseText);
}
```

Jäsentäminen ei ole välttämätöntä, jos huolehtii palvelimella MIME-tyypin eli tiedostotyypin otsikkotiedon oikeellisuudesta. PHP:ssä XML-tiedosto oikealla MIME-tyypillä voitaisiin muodostaa seuraavasti:

```
header('Content-type: text/XML');
echo "<?xml version='1.0' encoding='UTF-8'?>";
echo "<cities>";
echo "<city>Helsinki</city>";
echo "<city>Turku</city>";
echo "<city>Tampere</city>";
echo "</cities>";
```

Kuten aiemmin jo mainittiin, XHR:n Level 2 -suosituksessa vastauksen MIME-tyypin pystyy pakottamaan halutuksi. Tätä ei voi ainakaan vielä voi suositella, koska ominaisuutta ei ole toteutettu vielä ainakaan Internet Explorer 8:aan. Jos kuitenkin sitä halutaan käyttää, on ainakin ensin syytä tarkistaa, onko se käytettävissä:

```
if (xhr.overrideMimeType)
   xhr.overrideMimeType('text/XML');
else
   // käytetään parseria...
```

OverrideMimeType on kuitenkin muistettava asettaa oikeaksi jo ennen XHR-kutsun lähettämistä. Kun vastaus on muunnettu, keinolla tai toisella, XML-puuksi, siitä voidaan poimia halututut tiedot DOMin avulla ja tehdä tarvittavat muutokset HTML-sivulle. Edellä PHP:llä muodostettu XML-tiedosto voitaisiin tulostaa sivulle esimerkiksi seuraavasti:

```
var body = document.getElementsByTagName("body")[0];
var cities = xmlDoc.getElementsByTagName("city");
for (var i=0; i<cities.length; i++)
   body.innerHTML += "<p>"+cities[21].firstChild.nodeValue+"</p>";
```

Toinen vaihtoehto XML:n käsittelyyn selaimen puolella on XSLT. XSLT:n avulla XML-tiedosto saadaan muunnettua XHTML-tiedostoksi ja näin ollen JavaScript-koodin osalta päästään helpommalla. XSLT:stä ei tässä kuitenkaan sen enempää.

5.6.2 JSON

JSON eli JavaScript Object Notation on JavaScriptin olioiden serialisointiin luotu standardi. Tämän avulla voidaan välittää XML:n tapaan monimutkaisiakin rakenteita asiakkaan ja palvelimen välillä. JSON-viestin rakenne muodostuu merkkijonosta, johon on määritelty nimi-arvo pareja. Arvoihin voi halutessaan sisällyttää myös useita arvoja, jolloin ne ovat käytännössä taulukoita. Palvelimella JSON-viestien jäsentämiseen voidaan käyttää valmista kirjastoa. Esimerkiksi PHP:lle tehdyllä JSON-PHP kirjastolla viestin muuttaminen PHP-olioksi ja toisinpäin käy helposti omilla metodeillaan. JavaScriptissä JSON-viestin voi suorittaa käyttämällä eval()-funktiota. [10, luku 9.7.]

```
var json = "{ 'value':'hello','values': ['123', '456', '789'] }";

var msg = eval("(" + json + ")");
alert(msg.value);
```

Koska JavaScript-koodia ei käännetä vaan se suoritetaan ajonaikana, mahdollistaa se eval()-funktion hyödyntämisen. Pidemmänkin JavaScript-koodin lataus palvelimelta Ajaxin avulla ja sen suorittaminen on mahdollista. Tämä voi kuulostaa jopa pelottavalta. Edes sovelluskehittäjä ei pysty suoraan sivun lähdekoodia katsomalla sanomaan, millaisia funktioita sivulla on ja mitä se oikeastaan tekee, koska funktioita on voitu jälkeenpäin luoda dynaamisesti palvelimelta tulevasta viestistä. Usein kuitenkin tällaista On-Demand JavaScript -suunnittelumallia käytetään vain funktiokutsujen lähettämiseen palvelimelta. Sen sijaan että palvelin lähettäisi

esimerkiksi XML-vastauksen, jonka perusteella asiakaspuolen koodissa suoritettaisin haluttu funktio, lähetetäänkin suoraan funktion nimi tekstijonona asiakkaalle. Tämä tekstijono voidaan sitten vain sokkona suorittaa. [10, l. 6.5.]

5.6.3 Plain Text

Usein vastaukset eivät kuitenkaan ole monimutkaisia tietorakenteita, jolloin voidaan käyttää pelkästään tekstimuotoista vastausta. Tämä voi silloin helpottaa keskustelua asiakkaan ja palvelimen välillä, vähentää turhaa koodia ja myöskin vähentää dataliikennettä.

5.7 REST

Kutsuja voidaan siis lähettää eri HTTP-metodeilla ja vastaanottaa eri muodoissa. Voitaisiin esimerkiksi kutsua palvelua GET-metodilla, CGI-tyylisellä URL:illa varustettuna (esimerkiksi http://localhost/service.php?id=d23fa49b). Tai voitaisiin käyttää POST-metodia ja lähettää tarvittavat tiedot HTTP-viestin rungossa. Tai miksei lähetettäisi kyselyäkin XML-muodossa? Vaihtoehtojen kirjo on lukematon. Ja voisikin ajatella, ettei valinnalla ole mitään merkitystä, kunhan itse vain tietää mitä käyttää. Ja onhan asia niinkin. Kuitenkin internet on pullollaan erilaisia palveluita ja ohjelmointirajapintoja (API), ja jos jokainen palvelu käyttäisi omaa mielivaltaisesti valittua tyyliään, olisi niiden hyödyntäminen paljon hankalampaa. [10, luku 9.1.]

Internet-palveluiden tekemiseen on sovittu muutamia malleja. REST-palvelut esittävät palvelimen resurssina, jolle asiakkaat voivat antaa erilaisia käskyjä. Jokaisella resurssilla on oma URL, ja resurssille suoritettavaa operaatiota edustaa HTTP-metodin tyyppi. GET-metodia käytetään kysymiseen, DELETEä poistamiseen, PUTia lisäämiseen tai päivittämiseen. REST-palvelu on siinä mielessä tilaton, että se ei saa olla riippuvainen asiakkaan aiemmin tekemistä kyselyistä. Saman kyselyn useampaan kertaan tekemisellä ei saa olla lisävaikutuksia palvelimella. Asiakkaan puolella ei tietenkään mikään estä historiatietojen tallentamista ja niiden hyödyntämistä. [10, l. 9.1.]

Vastauksissa REST ei pakota käyttämään tiettyä formaattia, mutta XML on ehkä käytetyin [10, luku 9.1]. RESTin lisäksi muita käytettyjä palvelunpyyntöformaatteja ovat muun muassa RPC ja SOAP. Hyvän käsityksen RESTin ja muiden formaattien käytöstä saa tarkastelemalla esimerkiksi Flickr-kuvapalvelun monipuolista APIa (http://www.flickr.com/services/api). RESTin kaltaisten sääntöjen noudattaminen on usein kuitenkin turhaa, mikäli tarkoituksena ei ole rakentaa yleiskäyttöistä palvelua. Niinpä jää sovelluskehittäjän pohdittavaksi, mikä on juuri omaan tarkoitukseen sopivin pyyntöjen ja vastausten formaatti. Ainoaa oikeaa ratkaisua ei ole.

6 Palvelinohjelmointi Ajaxin kanssa

6.1 Palvelinteknologiat

Aikaisemmin palvelimen roolina web-ohjelmoinnin saralla on usein ollut ohjelman logiikan sekä tiedon esittämisen toteuttaminen. Nyt tiedon esittäminen voidaan haluttaessa eristää kokonaan asiakaspuolelle ja jättää logiikka palvelimen hoidettavaksi [10, luku 1.6]. Toimiva Ajax-sovellus saadaan, kun nämä osat saadaan keskustelemaan keskenään.

Ajax ei rajoita mitenkään palvelimen puolella käytettävää teknologiaa. Jää ohjelmoijan päätettäväksi, mitä ohjelmointi- tai skriptauskieltä haluaa käyttää. Tämän työn esimerkeissä on käytetty PHP:tä. Ohjelmointikielien lisäksi myös asiakkaan ja palvelimen välisen keskustelun "kieliä" on myös monia. Ne eivät ole mitenkään riippuvaisia palvelinpuolella käytettävästä tekniikasta. Pääajatus kaikissa on kuitenkin request-response tyylisen keskustelun toteuttaminen. Asiakas voisi esimerkiksi kysyä, onko käyttäjänimi "John" käytössä. Palvelimen tehtäväksi jää pelkästään tarkistaa tämä tietokannasta ja palauttaa vastaus jossain ymmärrettävässä muodossa. Kuten jo mainittiin, mikä tämä ymmärrettävä muoto on ja missä muodossa kysely tehdään, on täysin sovelluskehittäjän päätettävissä.

6.2 Cross-domain-pyynnöt

Usein tehdään kutsuja vain omalle palvelimelle, mutta joskus on tarve päästä käsiksi myös ulkopuolisiin resursseihin, kuten ulkopuolisiin ohjelmointirajapintoihin. Tämä aiheuttaa kuitenkin turvaallisuusriskin. Selaimet eivät normaalisti suostu keskustelemaan muun kuin saman palvelimen kanssa, josta sivu noudettiin. Ratkaisuna voisi olla selaimen turvallisuusasetusten muuttaminen, mutta tämä ei tietenkään tule kysymykseen, koska sivujen tulee toimia riippumatta käyttäjän asetuksista. Parempana ratkaisuna tähän on käyttää niin sanottua cross-domain proxyä. [10, luku 10.6.]

Cross-domain proxyn ideana on nimensä mukaisesti toimia välityspalvelimena ulkopuolisen resurssin ja asiakaspuolen skriptin välillä. Omalla palvelimella sijaitseva ohjelma käy hakemassa tietoa ulkoiselta palvelimelta välittäen ne asiakkaalle. Käytännössä tämä voi yksinkertaisimmillaan olla PHP-skripti, joka vain hakee ja tulostaa kopion ulkoisen palvelimen vastauksesta. Vastausta voi toki halutessaan muokata omalla palvelimellaan sopivammaksi. XML-tiedosto voitaisiin kokonaisuudessaan hakea ja tulostaa PHP:lla esimerkiksi seuraavasti:

```
header('Content-type: application/xml');
echo file_get_contents('http://www.externalsite.com/file.xml');
```

7 Ajax-työkalut

Kaikki tämän työn esimerkit on kirjoitettu käyttäen yksinkertaista tekstieditoria. Kuitenkin web-sovellusten kehittämiseen on tehty lukemattomia erilaisia työkaluja, joten niiden käyttöäkin kannattaa harkita. Jotkut työkalut mahdollistavat Ajax-sovelluksen tekemisen tietämättä mitään Ajaxin takana olevasta tekniikasta, generoiden automaattisesti palvelinpuolen ja asiakaspuolen koodin keskustelemaan keskenään. Koska erilaisten työkalujen ja kehysten määrä on niin valtava, ei niihin perehdytä tässä työssä sen tarkemmin. Tarkoituksena on oppia ymmärtämään Ajaxin toimintaa, eikä vain käyttämään tiettyä työkalua.

Kun koodi kirjoitetaan alusta alkaen itse, usein kaikki ei aina toimikaan heti niin kuin pitäisi. Yleensä unohdetaan, että paras koodin debuggeri on saatavilla ilman mitään lisätyökaluja. Se on nimittäin sivu itse. Sivulle voi luoda oman div-elementtinsä, johon kaikki tapahtumat kirjataan ylös. Näin ollen jonkin mennessä pieleen historiatieto on tallessa, ja vikaa on helpompi lähteä selvittämään.

Jos yhden työkalun nimen joutuu mainitsemaan, mainittakoon se tässä. Firebug on Firefox-selaimeen asennettava lisäosa. Sen avulla voi muun muassa katsoa sivun dynaamisesti muokattua HTML-koodia reaaliajassa. Selainten oma lähdekoodin katselemiseen tarkoitettu ominaisuus ei tätä näytä, ja kuten tähän mennessä on selvinnyt, sivuston sisältö voi muuttua radikaalistikin sen noutamisen jälkeen. Palvelinkutsujen seuraamista auttaa Firebugin Net-välilehti. Se tallettaa tietoa kaikista palvelimelle lähtevistä kutsuista, joista näkee koko lähetettävän ja vastaanotetun viestin sisällön otsikkotietoineen. XMLHttpRequest-pyynnöt näytetään jopa omassa XHR-välilehdessään. On kuitenkin muistettava, että kaikki Ajaxilla tehdyt kutsut eivät siinä näy, vaan iFrame-kutsuja pitää seurata All-välilehdeltä.

Firebug on oman koodin kehityksen lisäksi hyvä työkalu muiden sivujen analysointiin. Mielenkiinnosta voisi esimerkiksi selvittää, mitä tietoja Facebook hakee Ajaxilla ja mitä formaattia niissä käytetään. Vastaus tähän selviää Firebugilla käden käänteessä. Esimerkiksi "Näytä kaikki kaverit"-linkki näyttäisi aiheuttavan GET-kutsun parametrinaan ainakin käyttäjän ID-numero. Palvelimen vastaus näyttäisi olevan JSON-muodossa sisältäen muun muassa body-kentän, jossa sivulle lisättävä kaverilista on suoraan HTML-koodina.

8 Ajax-sovellusesimerkkinä peli

8.1 Vuoropohjainen peli

Esimerkkinä tähän työhön tehtiin Ajaxia hyödyntävän vuorovaikutteinen peli. Kyseessä on Worms-klassikon tapainen kahden pelaajan kesken vuorotellen pelattava peli, jossa tarkoituksena on tuhota kaikki vastustajan "örkit" kentältä ampumalla niitä kranaateilla. Pelin kulkua voisi demonstroida seuraavan esimerkin avulla.

Matti surffaa sivulle, jossa peli sijaitsee. Ruudulle ilmestyy teksti, jossa pyydetään odottamaan kunnes vastustaja löydetään. Toisaalla Pekka löytää tiensä samalle sivulle, jolloin kummankin selaimessa ilmoitetaan vastustajan löytymisestä ja pelin alkamisesta. Odottamaan joutunut pelaaja saa sijoittaa oman "örkkinsä" ensin haluamaansa paikkaan pelikentällä. Pekan vuoro on sen jälkeen, ja sijoittamista jatketaan, kunnes kaikki örkit ovat sijoitettuina.

Matti aloittaa pelin. Ampuakseen kranaatin hän painaa hiiren napin pohjaan ja päästää sen haluamassaan kulmassa tykin suhteen, jolloin kranaatti singahtaa tykistä voimalla, joka määräytyy hiiren napin pohjassaoloajasta. Kranaatti räjähtää sopivasti Pekan sijoittaman örkin kohdalla, jolloin se katoaa pelistä. Seuraavaksi on Pekan vuoro. Pelaajat ampuvat vuorollaan, kunnes Matin ampuessa Pekan viimeisenkin örkin ruudulle ilmestyvät voitto-onnittelut. Matti haluaa vielä pelata, ja hänen valitessaan uuden pelin, peli alkaakin välittömästi, koska odottamassa oli jo toinen pelaaja.

Mutta mitä tekemistä Ajaxilla on tämän kanssa? Pelin tapahtumat riippuvat toisen pelaajan ratkaisusta ja pelitilanteiden täytyy välittyä pelaajalta toiselle. Ajax antaa mahdollisuuden keskustella jatkuvasti toisen osapuolen kanssa palvelimen välityksellä. Tämän kaltaisen pelin toteuttaminen ilman Ajaxia olisi käytännössä mahdotonta, jollei sitten tyydyttäisi käyttämään kolmannen osapuolen selainlisäkkeitä.

8.2 Palvelin

8.2.1 Istunnot

Toisin kun esimerkiksi urheilutuloksia päivittävän sivuston kohdalla, sekä asiakkaan että palvelimen tulee olla tietoisia pelin tilasta ja tapahtumista. Tämä rajaa pois ainakin esimerkiksi RESTin käytön, koska sen mukaan palvelimen vastaukset eivät saa riippua edellisistä kyselyistä ja se käytännössä kieltää evästeiden käytön. Tarvitaanhan jokin keino tunnistaa, mistä kysely tulee, ja evästeet ovat siihen oivallinen apu.

Koska pelin tietoja ei tarvitse tallentaa pysyvästi asiakkaan tietokoneelle, vielä parempi vaihtoehto on käyttää istuntoja. Istuntotieto lähetetään aina automaattisesti HTTP-viestin otsikkotiedoissa, joten lähettämisestä ei tarvitse huolehtia. Samalla selaimella voi kuitenkin olla käynnissä vain yksi peli kerrallaan. Se on tässä pelkästään hyvä asia. Jos pelaaja sattuisi avaamaan saman pelin uuteen välilehteen, vanha peli keskeytyisi ja uusi alkaisi. Pelin toteutuksessa käytetyt istuntomuuttujat ovat esitelty liitteessä 1.

8.2.2 Tietokanta

Istuntojen avulla siis tiedot pelaajasta säilyvät palvelimella, mutta se ei pelkästään riitä. Koska pelaajat pelaavat keskenään, täytyy tietoa pelaajien välillä välittää. Tähän tehtävään hyvä apu ovat tietokannat. Eri tietokantasovellusten joukosta valittiin MySQL, koska se on ilmainen, suosittu ja helppokäyttöinen. Tietokannan taulujen konfigurointi tehtiin phpMyAdmin-web-käyttöliittymän kautta.

Tietokantaan tarvittiin taulu käyttäjille, käynnissä oleville otteluille ja vuoroa odottaville pelaajille. Tietoja pelaajien tekemisistä päivitetään ja luetaan tauluista, joten kanava tietojen välittämiseen kahden pelaajan välillä on nyt olemassa. Taulujen tarkempi rakenne on esitelty liitteessä 2.

8.2.3 Vastaukset

MySQL:n kumppanina palvelimella käytettiin PHP-skriptauskieltä. Istunnon tunnus lähetetään aina automaattisesti, kun ensimmäinen tunnistautuminen palvelimella on tehty. Näin palvelin saa aina tiedon siitä, kuka sitä on kutsunut. Lisäksi asiakkaalta lähetetään pelin senhetkinen tila. Tiloista tarkemmin myöhemmin. Tilan ja muidenkin lisäparametrien välittämiseen päätettiin käyttää CGI-tyylistä URL-osoittetta. Näin ollen HTTP-viestin runkoa ei tarvita ja GET-metodia voidaan käyttää. Riippuen vallitsevasta tilasta, kutsu on seuraavassa muodossa:

```
game.php?f={tilanumero}   // tai
game.php?f={tilanumero}&a={A}&b={B}
```

Palvelin vastaa kyselyihin aina lähettämällä vastauksen taulukon 5 mukaan. Koska vastaukset ovat hyvin yksinkertaisia, päätettiin esimerkiksi XML:n sijasta käyttää vain tekstimuotoisia vastauksia. Vastausta olisi voinut entisestään lyhentää käyttämällä sovittuja vastauskoodeja, mutta tässä työssä sille ei nähty tarvetta. Vastausten käsittelyn tekevän PHP-skriptin käytännön toteutus on esitetty liitteissä 3 ja 4.

Taulukko 5. Palvelimen vastaukset.

TILA	**KYSELY**	**VASTAUS**	**SELITYS**
0	NOT ATHED	OK	Käyttäjä tunnistettu.
1	AUTHED	QUEUED	Käyttäjä asetettu jonoon.
		FOUND	Vastustaja löytynyt.
2	QUEUED	FOUND	Vastustaja löytynyt.
		NOT FOUND	Käyttäjä edelleen jonossa.
3	MATCH FOUND	OK	Käyttäjä valmistautunut peliin.
4	READY	READY	Vastustaja valmistautunut.
		NOT READY	Vastustaja ei valmistautunut.
5	MATCH-WAIT	NO	Vastustaja ei liikkunut.
		A|B|C	Tiedot vastustajan liikkeestä.
6	MATCH-PLAY	OK	Liike päivitetty.
7	ONLINE UPDATE	YES	Vastustaja linjoilla.
		NO	Vastustaja poistunut.
0-7		ERROR	Virhe.

8.3 Asiakas

8.3.1 Palvelimen kutsuminen

Koska tässä työssä oli tarkoitus esitellä Ajaxin käyttöä, ei valmiin Ajax-kehyksen käyttö tullut kysymykseen. Kuitenkin pelissä kutsuja lähetetään palvelimelle vähän väliä pelitietojen välittämiseksi. Tarvitaan siis komponentti, joka pystyy lähettämään useita kutsuja helposti ja jopa samanaikaisesti.

Yksi mahdollisuus olisi käyttää taulukkoa, johon luotaisiin aina tarvittaessa uusi XHR-olio. Mutta tässä kohtaa ehkä paras ratkaisu olisi soveltaa olio-ohjelmoinnin ajattelutapaa. Tarkoitukseen tehtiin oma luokka, AjaxCaller, joka hoitaa kommunikoinnnin palvelimen kanssa. AjaxCaller-luokasta pyrittiin tekemään sellainen, ettei siihen tarvitse myöhemmin enää viitata. Tästä syystä sillä ei ole ollenkaan julkisia metodeita. Olion voi luoda tallentamatta sen osoitetta mihinkään ja se poistuu sitten aikanaan automaattisesti JavaScriptin "roskien kerääjän" toimesta. AjaxCaller-luokan toteutus on esitetty liitteessä 5.

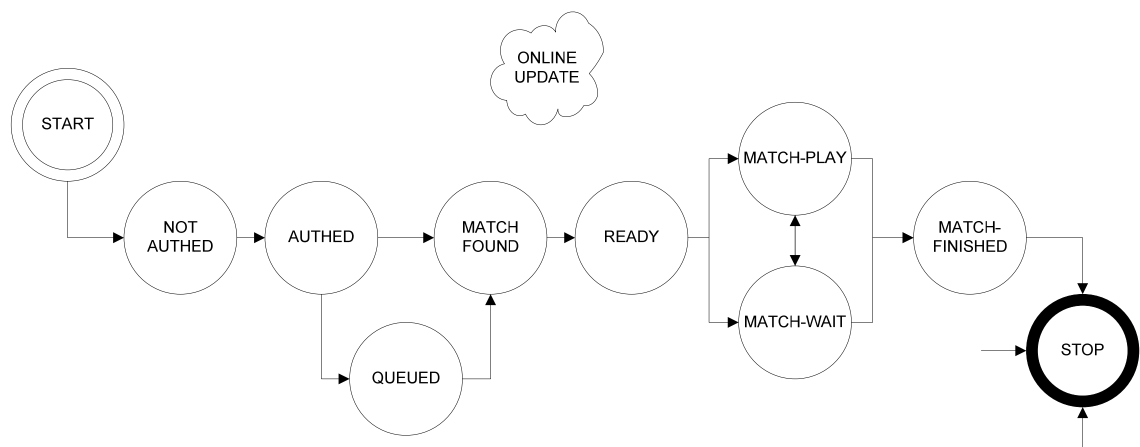
Kun peli on kesken, mitä tapahtuu silloin, kun vastustaja sulkee selainikkunan? Toinen pelaaja jää odottamaan vastustajan tekemää liikettä hamaan tulevaisuuteen. Tämä voidaan estää käyttämällä niin sanottua Heartbeat-suunnittelumallia [10, luku 17.5]. Mallissa palvelinta kutsutaan aina tietyin väliajoin, ikään kuin muistuttaen viestillä "olen vielä aktiivinen". Tätä mallia on sovelletettu myös tämän pelin toteutuksessa. Vastauksen saavuttua uusi kutsu lähetetään välittömästi palvelimelle. Jos yhteys palvelimelle on nopea, aiheutuu turhaa tiedonsiirtoa. Vielä parempi tapa olisikin asettaa kyselyjen välinen aika kiinteäksi. Kyselyn lähtiessä ajankohta tallennettaisiin ja ennen uutta kyselyä vertailtaisiin, joko määrätty ajanjakso on kulunut. Jos ei, uusi kysely ajastettaisiin lähtemään tämän ajanjakson kuluttua edellisestä.

Jatkuvasta palvelimen kutsumisesta voi aiheutua myös muita ongelmia. Heartbeat-suunnittelumallin yhdistäminen Timeout-malliin [10, luku 17.4] olisi myös hyvä lisä.

Näin selainten välilehtien aikakaudella pelaaja voi myös helposti unohtaa pelin päälle siirtyessään selaamaan muita sivuja. Uusia kutsuja palvelimelle lähtee jatkuvasti, täysin turhaan, kuluttaen palvelimen prosessointiaikaa ja kaistaa. Järkevää olisi siis "haistella" pelaajan todellista aktiivisuutta esimerkiksi hiiren liikkeen tunnistamisella. Kun pelaaja tunnistettaisiin passiiviseksi, voitaisiin kaikki kutsut keskeyttää. Tämän pelin tapauksessa aikarajan määrääminen pelaajan vuorolle ja koko pelille voisi jo olla riittävä ratkaisu. Peli päättyisi näin aina varmasti. Pelin päätyttyä kutsuja ei enää muutenkaan lähetetä.

8.3.2 Vastausten käsittely

Istuntotietojen ja tietokantojen avulla tiedot pysyvät tallessa palvelimella. Suuri osa prosessista kuitenkin tapahtuu asiakaspuolella. Koska pelin tilan ei tarvitse säilyä uudestaan sivulle palattaessa ja kaikki muutokset tapahtuvat Ajaxia käyttäen, voidaan tietoja tallentaa selaimen välimuistiin eli käyttää JavaScriptin muuttujia. Niiden hallintaa ja koko prosessin kulkua helpottamaan laadittiin tilataulu kaikista tiloista, joissa sivu voi olla. Tilakaavio on esitetty kuvassa 3.



Kuva 3. Sovelluksen tilakaavio.

Tilakaavion tilat eivät välttämättä aina yksiselitteisesti kuvaa sovelluksen täydellistä tilaa. Ne voivat olla riippuvaisia myös aikasemmista tiloista. Esimerkiksi READY-tilasta

siirtyminen voi tapahtua joko MATCH-PLAY- tai MATCH-WAIT-tilaan riippuen siitä, kumpi pelaajista on joutunut jonottamaan. Jonottava pelaaja saa aina aloittaa pelin.

Tilasta toiseen siirryttäessä lähetetään aina kutsu palvelimelle. Palvelimelta tullut vastaus taulukon 5 mukaan määrää, mihin tilaan siirrytään, jos siirrytään. Vastaukset käsitellään aina omissa callback-funktioissaan, joiden toteutukset on tarkemmin esitelty liitteessä 6.

8.4 Pelin ohjelmointi

8.4.1 Vaihtoehtona HTML 5

Tarkoituksena oli tehdä peli, joka toimii kaikilla selaimilla ilman mitään lisäosia. Tämä rajasi pois pelin toteuttamisen esimerkiksi Flashiä käyttäen. Ajatuksena oli joka tapauksessa käyttää Ajaxia, joten perinteinen HTML oli käytännössä ainoa vaihtoehto. HTML:ää ei kuitenkaan ollut alun perin suunniteltu pelien tekemiseen. Kiitos HTML:n ja CSS:n dynaamisen muokkauksen DOM:in ja JavaScriptin avulla, tämä on kuitenkin mahdollista.

Pelin toteuttaminen HTML:ää käyttäen asettaa kuitenkin rajoituksia. HTML ei tällä hetkellä tue missään muodossa esimerkiksi sivulla olevien kuvien kääntämistä tai animaatioita. Näistä pitää joko tinkiä tai yrittää toteuttaa vastaava toiminnallisuus "kepulikonstein". Parannusta moniin ongelmiin tuovat HTML5:n uudet ominaisuudet, mutta koska monet niistä eivät ole vielä käytössä kaikissa selaimissa, ei niitä käytetty tämän pelin toteuttamisessa.

Joka tapauksessa pelisuunnittelijoille iloa on tuomassa W3C:n HTML5-suositus. Yhtenä merkittävänä uutena ominaisuutena HTML5 tuo canvas-elementin eli piirtoalustan. Canvas on oma elementtinsä, joka määritellään loogisesti tagilla <canvas>. Se on kuin <img> paitsi sen sisältö on itse muutettavissa. Canvasiin voi piirtää muun muassa

viivoja, erilaisia kuvioita, kuvia ja värejä. Lisäksi piirtoalustalla on oma koordinaatistonsa, joka on vapaasti skaalattavissa ja käännettävissä. [25.]

Canvasille voi tällä hetkellä määritellä vain kaksiulotteisen "maailman", mutta kolmiulotteinen tuki on mitä suurimmalla todennäköisyydellä tulossa [26]. Epävirallisia toteutuksia löytyy jo, ainakin WebGL:ää hyödyntävä CopperLicht. Sen avulla kolmiulotteista grafiikkaa voidaan ajaa web-selaimessa JavaScriptillä kontrolloiden ilman selainlisäkkeitä [27]. Kolmiulotteisuus antaisi täysin uuden ulottuvuuden web-sivujen suunnitteluun. Miltä esimerkiksi kuulostaisi se, että Wikipedian Formula 1 -sivulla olisikin tylsän kuvan sijasta vapaasti pyöriteltävä 3D-malli autosta? Tämä voi olla tulevaisuudessa hyvinkin mahdollista.

Vaikkei canvasiakaan suoranaisesti ole tarkoitettu pelien tekemiseen, tekee se kuitenkin tämän helpommaksi. Huonojakin puolia on. Kun jotakin piirtää canvasille, se pysyy samanlaisena koko ajan. Jos elementin paikkaa halutaan siirtää, joutuu kaikki piirtoalustan elementit piirtämään uudestaan. Tämä vie aikaa ja syö koneen tehoja. Tosin nykyajan tietokoneilla tämäkään ei ole suuri ongelma. [26.]

Monia HTML5:n määrittelemiä ominaisuuksia on jo toteutettu moniin selaimiin. Esimerkiksi canvas-tuki löytyy jo Chromen, Firefoxin, Operan ja Safarin uusimmista versioista.

8.4.2 Pelin aloittaminen

Ottelu on valmis alkamaan, kun tilasta MATCH-FOUND lähetetystä kutsusta on saatu vastaus "READY". Peli on eriytetty omaan luokkaansa, eikä sen tarvitse huolehtia palvelimen kanssa käytävästä keskustelusta. Pelin hallintaan tarkoitetun luokan toteutuksessa alustuksien jälkeen luodaan ajastin, jonka avulla eri pelin tapahtumat voidaan päivittää. Lopuksi jäädään odottamaan pelaajan tai vastustajan tekemää liikettä. Tarkempi luokan toteutus lohkokaavioineen on liitteessä 7 ja olion luovan callback-funktion totetutus liitteessä 8.

8.4.3 Liikkuminen

Kun vuoro on pelaajalla, eli ollaan tilassa MATCH-PLAY, hiiren alaspainaminen joko sijoittaa örkin pelikentälle tai aloittaa ampumisvoiman laskemisen. Hiiren napin ylösnostaminen laukaisee kranaatin suuntaan, jonka sen hetkiset hiiren koordinaatit määrittelevät. Pelaajan liike käsitellään mousedown- ja mouseup-tapahtumien avulla, ja niiden toteutus löytyykin liitteestä 9.

Näiden funktioiden avulla ei tietenkään mitenkään voida saada tietoa siitä, mitä vastustaja on tehnyt, kun ollaan tilassa MATCH-WAIT. Mistä vastustajan liike saadaan? Tietenkin palvelimelta Ajaxin avulla. Tiedon saavan callback-funktion käsittely on esitetty liitteessä 10.

Riippumatta siitä, tuleeko tieto pelaajan liikkumisesta hiirenpainalluksen kautta vai saadaanko se palvelimelta, tiedot liikkeestä täytyy välittää Game-luokan käsiteltäväksi. Nämä liikkumisen hoitavat metodit on esitelty liitteessä 11.

8.4.4 Lähtökulma ja voima

Kulma tykin suhteen voidaan laskea perustrigonometriaa hyödyntäen hiiren ja tykin koordinaattien avulla. Huomion arvoista on, että y-koordinaatisto on selaimen koordinaatistossa aina käänteinen, eli nollapiste on vasemmassa ylänurkassa. Kranaatin laukaisukulma saadaan näin ollen kaavasta:

$$\alpha = \tan^{-1}\frac{y_1 - y_2}{x_2 - x_1}$$

α on kranaatin lähtökulma radiaaneina
x_1, y_1 ovat hiiren koordinaatit pelialueen suhteen
x_2, y_2 ovat tykin koordinaatit pelialueen suhteen.

Voima saadaan puolestaan nykyisen ajan ja hiiren painalluksen aloittamisajan erotuksena. Kranaatti ammutaan pelikentällä näiden tietojen mukaan. Tiedot käytetystä voimasta ja kulmasta lähtevät lopuksi palvelimelle.

8.4.5 Tapahtumien päivitys

Pelin tapahtumien päivittämistä varten peliin tarvitaan ajastin. Tapahtumat päivittyvät aina ennalta määrätyin intervallein. Tässä intervalliksi on valittu 40 ms eli 25 fps. Liike on silloin tarpeeksi sulavaa, eikä näytä tökkivän pelaajan silmissä. Nopeampi päivitystaajuus voi aiheuttaa vähemmän tehokkailla järjestelmillä nykimistä.

Ainakaan yli 50 fps:n päivitystaajuuksien käyttö ole suositeltavaa, koska silmä ei juurikaan sitä suurempia päivitystaajuuksia pysty erottelemaan. Joka syklillä päivitetään kaikki pelikentän tapahtumat, tai käytännössä kutsutaan käytettyjen Grenade- ja Orc-luokkien omia päivitysmetodeitaan. Päivityssyklin lohkokaavio ja toteutus ovat liitteessä 12.

8.4.6 Kranaatti

Kranaatin hallinta on siis eriytetty omaan luokkaansa ja sen tarkempi toteutus on esitelty liitteessä 13. Ennen kuin kranaatti voidaan ampua, sen lähtönopeus täytyy selvittää. Nopeus saadaan kyllä suoraan hiiren painalluksen pohjassaoloajasta, mutta koska nopeus on vektorisuure, täytyy erikseen laskea sen x- ja y-suuntaiset komponentit. Ne saadaan kaavoista

$v_x = \cos(\alpha) \cdot v$ ja

$v_y = \sin(\alpha) \cdot v$,

jossa v_x ja v_y ovat kranaatin x- ja y-suuntaiset nopeudet. Kranaatin tulee myös käyttäytyä mahdollisiman realistisesti, joten se ei voi esimerkiksi pudota alaspäin

vakionopeudella. Realistinen putoamisnopeus määräytyy gravitaatiokiihtyvyyden avulla. Kaavan

$$v_y = v_{y0} - g \cdot t \qquad | \, t = 1$$

$$v_y = v_{y0} - g$$

jossa v_{y0} on aikasempi nopeus ja g gravitaatiokiihtyvyys, mukaan saadaan uusi nopeus y-suuntaiseen liikeeseen. Koska nopeus päivitetään aina jokaisena ajanjaksona, voidaan aikakomponentti t jättää huomioimatta. X-suunnassa hidastavaa voimaa ei tässä ole, vaikka todellisuudessa ilmanvastus sellainen olisi. Kranaatin sijainti voidaan helposti päivittää nopeuden perusteella kaavojen

$x = x_0 + v_x$ ja

$y = y_0 + v_y$,

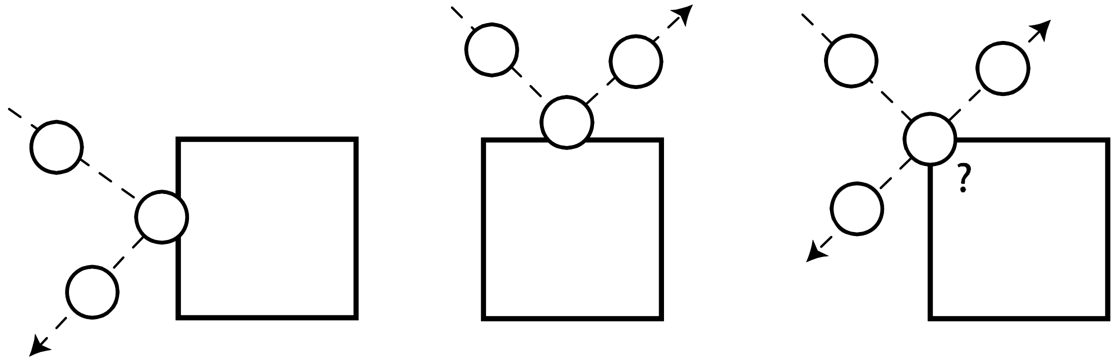
jossa x_0 ja $y_0$ovat sijainnit ennen päivitystä, avulla. Aikaa ei myöskään tässä huomioida, koska päivitys tehdään joka syklillä.

Pelissä ajanjakso määräytyy päivitystaajuudesta, eli se on 40 ms. On siis muistettava, että määritelty gravitaatiokiihtyvyys on myös riippuvainen käytetystä päivitystaajuudesta. Lisäksi yksikkönä on metrien sijasta pikselit, joten realistisen kiihtyvyyden laskeminen lienee joka tapauksessa turhaa. Sopivan tuntuiseksi g:n arvoksi saatiin kokeilemalla 0.9.

8.4.7 Törmäykset

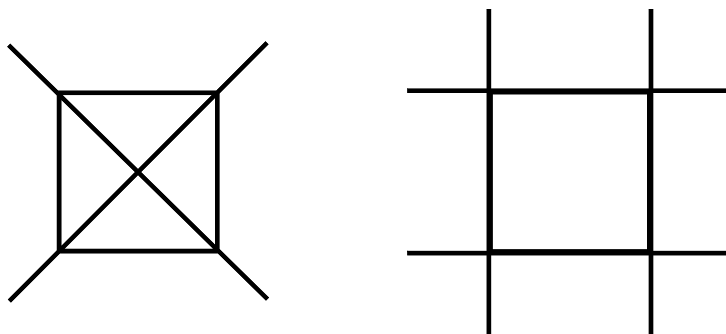
Kranaatin sijainti ei riipu pelkästään sen nopeudesta. Pelialueella on erilaisia elementtejä, johon voi tulla törmäys. Törmäyksestä kranaatin tulee kimmota oikeaan suuntaan. Törmäys on mahdollista tunnistaa vertailemalla pommin ja elementin

keskinäistä sijaintia. Reagointia varten tulee selvittää myös se, mille sivulle elementtiä kranaatti on osunut. Vertailemalla kranaatin aikaisempaa sijaintia nykyiseen sijaintiin saadaan selville mistä suunnasta törmäys on tapahtunut.



Kuva 4. Kranaatin törmäys elementtiin

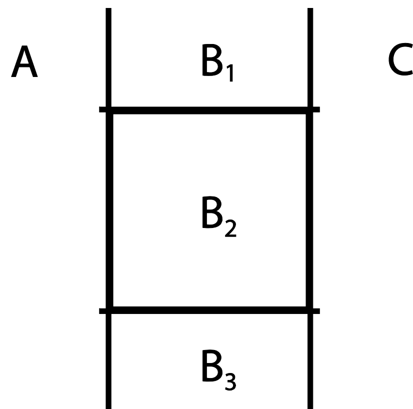
Kuten kuva 8 havainnollistaa, valinta kimpoamisen suunnaksi ei aina ole yhtä selkeä. Tätä varten apuna käytettiin törmäyssektoreita. Vertailemalla kranaatin aikaisempaa ja nykyistä sektoria saadaan selville kranaatin tulosuunta. Sektorin laskentaan on varmasti monia keinoja, ja sektorit voitaisiin jakaa esimerkiksi kuvan 9 osoittamalla tavalla.



Kuva 5. Malleja törmäyssektorin laskentaan

Vasemman puoleinen ratkaisu olisi parempi, vaikkei täydellinen. Tämä kuitenkin tekee toteuttamisesta trigonometrisesti vaikeamman, koska sektorit ovat 45 asteen kulmassa. Tämän pelin tapauksessa yksinkertaisempikin malli riittää tarpeeksi

realistiseen lopputulokseen. Oikeanpuoleisessa mallissa kulmasektorit eivät anna juurikaan lisäinformaatiota siitä onko kranaatti tulossa ylhäältä vai sivusta. Lopulta päädyttiinkin kuvassa 10 esitetyyn ratkaisuun, joka oli vielä helpommin toteutettavissa.



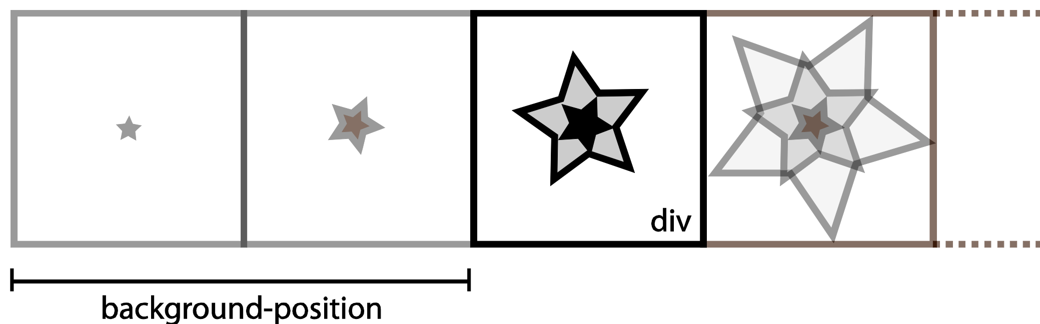
Kuva 6. Törmäyssektorit

Jos kranaatti on sektorin B2 alueella, törmäys on tapahtunut. Sen jälkeen vertaillaan kranaatin aikaisempaa sijaintia. Sijainnin ollessa alueella A, kranaatti kimpoaa vasemmalle ja alueella C oikealle. B1 kimmottaa kranaatin ylöspäin ja B3 alaspäin. Malli ei tietystikään ole lähellekään täydellinen. Vääriä ratkaisuja tulee, koska sivuille kimpoamista suositaan. Kuitenkin liike pelissä on sen verran nopeaa, että mitään epäloogisuutta on käytännössä lähes mahdotonta huomata. Törmäysteen liittyvien Grenade-luokan metodien toteutukset voi katsoa liitteestä 14.

8.4.8 Räjähdykset

HTML ei salli kovin kummoista grafiikan rakentamista peliin. Kuitenkin olisi hyvä, että kranaatin räjähtäessä voitaisiin käyttää animaatiota. Vaikka aiemmin mainittiin, ettei HTML tue animaatioiden toteuttamista, on siihen kuitenkin muutama konsti. Ainoa kuvaformaatti, joka tukee animaatioita, on GIF. GIF-animaatioita ei kuitenkaan pysty mitenkään kontrolloimaan. Kun animaatio on ladattu, se pyörii samalla tavalla koko ajan. Se ei siis sovi tämän räjähdysanimaation toteuttamiseen.

Pelisuunnittelijoille tutun Sprite-luokille ominaisen toiminnallisuuden toteuttaminen on kuitenkin myös mahdollista. Ratkaisuna on käyttää taustakuvaa div-oliossa. Taustakuva on "animaatioliuska", jossa jokainen animaatioon kuuluva kuva on asetettu vierekkäin. Kuten kuvasta 11 nähdään, div:n leveys on aina yhden animaatiokuvan levyinen jättäen näkyviin aina vain yhden kuvan. Taustakuvan sijoittelua pystyy hallitsemaan background-position-ominaisuuden avulla. Näin ollen taustaa voidaan aina siirtää paljastamaan vain haluttu kohta kuva animaatiosta, yleensä jokainen animaatioliuskan kuva järjestyksessä. Vaikka kuvien pitää olla suorakulmion muotoisia, voidaan PNG-kuvien läpinäkyvyyttä käyttää hyväksi ja muodostaa näin ollen minkä tahansa muotoisia kuvia ja animaatioita. Tarkempi animaation hallintaan käytetty koodi on esitetty liitteessä 15. [10, luku 15.2.]



Kuva 7. Animaation näyttäminen div:n taustakuvan avulla.

8.4.9 Örkit

Örkit ovat myös animoituja, mutta niissä päädyttiin käyttämään yksinkertaisempaa GIF-animaatiota. Se ei ole kontrolloitavissa, mutta tässä tapauksessa sillä ei ollut suurta merkitystä, ja luokan toteutuksessa päästiin helpommalla. Örkkien hallintaan tarkoitetusta Orc-luokasta pyrittiin muutenkin tekemään yksinkertainen. Toisin kuin kranaatit, örkit tippuvat vakionopeudella. Sijoitettaessa örkki tippuu alaspäin, kunnes vastaan tulee elementti. Tämän jälkeen se ei voi enää liikkua. Lisäksi törmäysten tarkastelussa voitiin käyttää yksinkertaisempaa algoritmia, koska sivuttaissuuntaista liikettä ei ole. Orc-luokan tarkempi toteutus on esitetty liitteessä 16.

9 Yhteenveto

Ajax osottautui varteenotettavaksi vaihtoehdoksi web-sovellusten kehitämisessä. Jo nyt sitä käytetään lukemattomissa verkon sovelluksissa. Tulevaisuus näyttää hyvin valoisalta. Kun taustalla oleva teknologia kehittyy entisestään, tulee se mahdollistuttamaan entistä monipuolisempien sovellusten rakentamisen. Ajax ei vielä sovellu aivan jokaiseen tarkoitukseen, mutta ei sen tarvitsekaan. Se on ehkä parhaimmillaan hoitamaan pieniä asioita, vaikka tässä työssä sitä käytettiinkin hieman monimutkaisempaan tehtävään. Se on löytämässä oman paikkansa internetin perustekniikkana.

Ajaxista ei ainakaan vielä ole korvaamaan kolmannen osapuolen teknologioita, kuten Flashiä. Flashillä on omat käyttötarkoituksensa, ja jotkut tehtävät hoituvat paremmin sillä. Ajax ja Flash voivat kuitenkin yhdessä käytettynä toimia erittäin hyvin, Flashin esimerkiksi hoitaessa sivun graafiset valikot ja Ajaxin toimiessa muiden askareiden apuna. Tosin jos monimutkaisempia ominaisuuksia ei tarvita, antaa Ajax mahdollisuuden tehdä täysin suositusten mukaisia web-sivuja, jotka toimivat riippumatta järjestelmästä, selaimesta tai siihen asennetuista lisäosista.

Ajaxin tulevaisuudesta ei voida puhua, jollei mainita HTML5:tä. Viime aikoina tämä lyhenne on alkanut näkyä entistä enemmän alaa käsittelevissä artikkeleissa. Vielä ei täysin tiedetä, mihin HTML5 tulee kykenemään. On toki mahdollista, että HTML5 tulee pitkällä aikavälillä syrjäyttämään myös Flashin. Flash on jo niin vakiintunut teknologia, ettei siitä ihan heti päästä eroon. HTML5:n etuna on kuitenkin se, että teknologia on varmasti tulossa, ja se tulee olemaan osa kaikkia selaimia lähitulevaisuudessa. HTML5-sovelluksia voidaan alkaa kehittää jo nyt. Myös Ajax tulee olemaan osana teknologiaa.

Tämän työn esimerkkisovelluksesta onnistuttiin luomaan toimiva kokonaisuus. Pelistä jäi kuitenkiin uupumaan useita ominaisuuksia, joten sen julkaisemista internetissä ei vielä tässä vaiheessa kannattanut tehdä. Yksinkertaisen grafiikan toteuttaminen ilman kolmannen osapuolen teknologioita onnistui kohtuullisen hyvin. HTML5 tulee

kuitenkin mahdollistamaan vielä paremman grafiikan toteuttamisen canvas-piirtoalustan avulla. Vaikka joitain ongelmia oli, tämän pelin toteutuksessa onnistuttiin hyvin käyttämään olio-ohjelmoinnin ajattelutapaa ja eriyttämään eri toiminnallisuudet omiin luokkiinsa. Sovelluksen laajentamisesta tehtiin näin ollen helpompaa. Julkaisukelpoisen sovelluksen toteuttaminen tästä on siis hyvin mahdollista, ja se tullaankin toteuttamaan lähitulevaisuudessa.

Tällä hetkellä puhtaasti Ajaxilla tehtyjen pelien osuus internetissä on hyvin marginaalinen Flashin hallitessa. Tämä työ kuitenkin osoitti, että Ajaxia pystytään soveltamaan hyvin myös interaktiivisten pelien ohjelmoinnissa, eikä olla aina pakotettuja käyttämään kolmannen osapuolen teknologioita. Vaikkei Ajax ehkä parhaimmillaan tässä tarkoituksessa ole ja rajoituksia on paljon, tulevan HTML5-standardin ja uusien XMLHttpRequestin päivityksien myötä mahdollisuudet entistä hienompien pelien ohjelmoimiseen tulevat parantumaan olennaisesti. On mahdontonta sanoa, tuleeko Ajax-pelien osuus tämän seurauksena radikaalisti nousemaan. Tämä paljolti riippuu siitä, onko HTML5:stä Flashin korvaajaksi myös pelisovelluksien osalta.

Joka tapauksessa internetin rakenne on muutoksen partaalla. Sovelluksista ja palveluista halutaan tehdä entistä enemmän työpöytäsovellusten kaltaisia. Ajax on mahdollistanut tämän kaltaisten vuorovaikkuteisten sovellusten kehittämisen, ja sovelluskehittäjät ovat vihdoin alkaneet hyväksyä Ajaxin olevan osa internetiä. Tosin internet muuttuu koko ajan, ja vain aika näyttää, tuleeko joku muu tekniikka viemään Ajaxin paikan. Mutta miksei Ajaxkin voisi muuttua? Uusien teknologioiden lisääminen Ajaxiin ei kuulosta mitenkään mahdottomalta. Voihan olla myös, että koko termi Ajax hiljalleen poistuu käytöstä kokonaan, koska tekniikasta tulee niin vakiintunut. Se ei kuitenkaan tarkoita, että Ajax olisi hävinnyt. Ajaxhan on enemmänkin ajattelutapa kuin mikään teknologia, ja tämä ajattelutapa on mielestäni ainoa ajattelutapa tulevaisuuden web-sovellusten rakentamiseen. Se ei tule katoamaan mihinkään, ei ehkä koskaan.

Lähteet

1 Sulopuisto, Olli. HTML5, paras keksintö sitten salmiakkijäätelön? (WWW-dokumentti.) <http://www.mbnet.fi/net.nyt/juttu.aspx?id=3126>. 23.3.2010. Luettu 24.3.2010.

2 Hopmann, Alex. The story of XMLHTTP. (WWW-dokumentti.) <http://www.alexhopmann.com/xmlhttp.htm>. Updated January 31, 2007. Luettu 10.1.2010.

3 Native XMLHTTPRequest object. (WWW-dokumentti.) The Windows Internet Explorer Weblog <http://blogs.msdn.com/ie/archive/2006/01/23/516393.aspx>. Published January 23, 2006. Luettu 3.2.2010.

4 Jung, Eric. nsIXMLHttpRequest. (WWW-dokumentti.) Mozilla Developer Center. <https://developer.mozilla.org/en/nsIXMLHttpRequest>. Page last modified 16 May 2008. Luettu 22.1.2010.

5 The XMLHttpRequest Object, W3C Working Draft. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>. 05 April 2006. Luettu 18.1.2010.

6 Taft, Darryl K. Study: Leading Retail Sites Slow to Adopt AJAX. (WWW-dokumentti.) eWeek. <http://www.eweek.com/c/a/Application-Development/Study-Leading-Retail-Sites-Slow-to-Adopt-AJAX/>. 2007-01-05. Luettu 6.1.2010.

7 Swartz, Aaron. A Brief History of Ajax. (WWW-dokumentti.) <http://www.aaronsw.com/weblog/ajaxhistory>. December 22, 2005. Luettu 6.1.2010.

8 Garrett, Jesse James. Ajax: A New Approach to Web Applications. (WWW-dokumentti.) Adaptive Path. <http://www.adaptivepath.com/ideas/essays/archives/000385.php>. February 18, 2005. Luettu 4.1.2010.

9 Mangalindan, Mylene & Buckman, Rebecca. New Web-based Technology Draws Applications, Investors. (WWW-dokumentti.) The Wall Street Journal. <http://online.wsj.com/public/article/SB113098635587487074-3diFzslPm_iutdYLU2C5e4DinUA_20061103.html>. November 3, 2005. Luettu 1.2.2010.

10 Mahemoff, Michael. Ajax Design Patterns. First Edition. Sebastopol: O'Reilly Media, 2006.

11 Castledine, Earle. Using the XMLHttpRequest Object and AJAX to Spy On You. (WWW-dokumentti.) DevX. <http://www.devx.com/webdev/Article/28861>. August 9, 2005. Luettu 1.2.2010.

12 Mahemoff, Michael. Spying on Users With XMLHttpRequest. (WWW-dokumentti.) <http://softwareas.com/spying-on-users-with-xmlhttprequest>. August 10th, 2005. Luettu 1.2.2010.

13 Naone, Erica. HTML 5 Could Challenge Flash. (WWW-dokumentti.) <http://www.technologyreview.com/web/24844/?a=f>. March 23, 2010. Luettu 24.3.2010.

14 Justus, Chris. Google Suggest Dissected. (WWW-dokumentti.) <http://serversideguy.blogspot.com/2004/12/google-suggest-dissected.html>. December 14, 2004. Luettu 1.3.2010.

15 Chapman, Stephen. JavaScript and JScript : What's the Difference? (WWW-dokumentti.) About.com. <http://javascript.about.com/od/reference/a/jscript.htm>. Luettu 25.1.2010.

16 Koss, Mike. Object Oriented Programming in JavaScript (WWW-dokumentti.) <http://mckoss.com/jscript/object.htm>. January 14, 2006. Luettu 25.1.2010.

17 Le, Alex. SetInterval() Scope problem & solutions for Firefox and Internet Explorer 6. (WWW-dokumentti.) <http://alexle.net/archives/169>. December 22, 2006. Luettu 1.2.2010.

18 Document Object Model (Core) Level 1, W3C Recommendation. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html>. 1 October, 1998. Luettu 7.2.2010.

19 Document Object Model (HTML) Level 1, W3C Recommendation. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-html.html>. 1 October, 1998. Luettu 7.2.2010.

20 XMLHttpRequest, W3C Working Draft. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/2009/WD-XMLHttpRequest-20091119/>. 19 November 2009. Luettu 9.2.2010.

21 About Native XMLHTTP. (WWW-dokumentti.) Microsoft. <http://msdn.microsoft.com/en-us/library/ms537505%28VS.85%29.aspx>. Luettu 3.3.2010.

22 XMLHttpRequest Level 2, W3C Working Draft. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/2009/WD-XMLHttpRequest2-20090820/>. 20 August 2009. Luettu 9.2.2010.

23 Webber, Joel. Mapping Google. (WWW-dokumentti.) <http://blog.j15r.com/2005/02/mapping-google.html>. February 09, 2005. Luettu 20.3.2010.

24 XML Parser. (WWW-dokumentti.) W3Schools. <http://www.w3schools.com/XML/xml_parser.asp>. Luettu 9.1.2010.

25 HTML5, W3C Working Draft. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/2010/WD-html5-20100304/>. 4 March 2010. Luettu 20.3.2010.

26 Smith, Andi. Canvas Tutorial. (WWW-dokumentti.) Mozilla Developer Center. <https://developer.mozilla.org/en/Canvas_tutorial>. Page last modified 20 Oct 2009. Luettu 11.2.2010.

27 CopperLicht Features Overview. (WWW-dokumentti.) Ambiera. <http://www.ambiera.com/copperlicht/features.html>. Luettu 20.2.2010.

Liite 1: Istuntomuuttujat

Istunnot toimivat niin, että asiakkaan puolella on tallennettuna istunnon numero. Numero lähetetään aina palvelimelle automaattisesti HTTP-viestin otsikkotiedoissa. Palvelimella on omissa tiedostoissaan tallessa istuntonumerot ja niihin liittyvät tiedot, joita pysytään haluttaessa lukemaan ja kirjoittamaan.

Tässä sovelluksessa istuntotiedoissa säilytetään tieto käyttäjän käyttäjätunnuksesta eli uid:sta. Lisäksi tallennetaan tieto pelaajan mahdollisesta käynnissä olevasta ottelusta ja siitä, onko pelaaja aloittavana pelaajana. Nämä eivät välttämättä olisi pakollisia, mutta ne helpottavat ja nopeuttavat tietokantahakuja. Taulukossa 1 on esitelty istuntomuuttujat. Istuntomuuttujiin voidaan viitata PHP-koodissa automaattisesti käytössä olevan $_SESSION-muuttujan avulla.

Taulukko 1. Istuntomuuttujat

PHP-VIITTAUS	SELITYS
$_SESSION["uid"]	Käyttäjälle luotu tunnus.
$_SESSION["mid"]	Käynnissä olevan ottelun tunnus. "null", jos peliä ei ole käynnissä.
$_SESSION["me"]	"x", jos pelaaja luonut ottelun. "y", jos peli vastustajan luoma.
$_SESSION["you"]	Päinvastoin kun edellinen.

Liite 2: Tietokanta

Palvelimen MySQP-tietokantaan sijoitettiin seuraavat tiedot. Kuvassa 1 on esitetty users-taulun rakenne. Uid asetettiin automaattisesti kasvavaksi, jottei sen arvon antamisesta tarvitse huolehtia. Lisäksi luotiin lastOnline kenttä, joka pitää sisällään käyttäjän edellisestä palvelin-kontaktin ajankohdan. Tämän avulla voidaan seurata, onko vastustaja vielä paikalla.

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	**uid**	int(10)			No	*None*	auto_increment
☐	**lastOnline**	timestamp			No	CURRENT_TIMESTAMP	

Kuva 1. Users-taulu.

Matches-taulusta tehtiin kuvan 2 mukainen. Mid, eli ottelun tunnus, luodaan myös automaattisesti juoksevalla numeroinnilla. Uidx ja uidy sisältävät otteluun osallistuvien pelaajien uid-numerot. Uidx on aina pelin luonneen pelaajan uid. UidxStatus ja uidyStatus sisältävät arvon "1", kun peli on luotu ja "2", kun pelaaja on vahvistanut olevansa valmis aloittamaan ottelun. UidxMoves ja uidyMoves pitävät sisällään kyseisen pelaajan edellisen liikkeen. Kun vastustajan käy lukemassa tämän tiedon, nollataan tietue merkiksi siitä, että arvo on jo luettu.

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	**mid**	int(10)			No	*None*	auto_increment
☐	**uidx**	int(10)			No	*None*	
☐	**uidy**	int(10)			No	*None*	
☐	**uidxStatus**	int(1)			No	1	
☐	**uidyStatus**	int(1)			No	1	
☐	**uidxmoves**	varchar(100)	latin1_swedish_ci		Yes	*NULL*	
☐	**uidymoves**	varchar(100)	latin1_swedish_ci		Yes	*NULL*	

Kuva 2. Matches-taulu.

Liite 2: Tietokanta

Kuvassa 3 esitetyssä queue-taulussa on aina vain enintään yksi pelaaja tai käytännössä sen uid numero. Kun pelaaja pääsee aloittamaan ottelun poistetaan uid jonosta.

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	**uid**	int(10)			No	*None*	

Kuva 3. Queue-taulu.

Liite 3: MySQL-funktiot

Käytännön syistä mySQL-tietokantakutsujen apuna on käytetty muutamia apufunktiota, jotka ovat eriytetty omaan PHP-tiedostoonsa. Virheiden käsitteleminen jätettiin tässä työssä huomioimatta.

```
function db_open()
{
   @mysql_connect(DB_HOST, DB_USER, DB_PASSWORD)
      or die(mysql_error());
   @mysql_select_db(DB_NAME)
      or die(mysql_error());
}

function db_close()
{
   mysql_close();
}

function db_query($query_str)
{
   $result = mysql_query($query_str)
      or die(mysql_error());
   if (is_resource($result))
      return mysql_fetch_array($result);
   else
      return null;
}
```

Liite 4: PHP-skripti

Kutsuttaessa palvelimen PHP-skriptiä käydään päivittämässä ja hakemassa tietoja MySQL-tietokannasta. Vastaus muodostuu yleensä näiden tietojen perusteella. Käytännössä koodissa on switch-case -rakenne. Kyselyn parametri "f" eli asiakkaan tila määrää, mikä koodin haara suoritetaan.

```
switch($_GET['f'])
{
   case "0":  // NOT AUTHED
      // ...
      break;
   case "1":  // AUTHED
      // ...
      break;
   // ...
}
```

Asiakkaalta voisi tulla esimerkiksi seuraavanlainen kutsu PHP-skriptille:

```
game.php?f=0
```

Nähdään, että tilakoodi on 0 eli NOT AUTHED, joka tarkoittaa sitä, ettei käyttäjää ole vielä tunnistettu palvelimella. Niinpä tarvittavat toimenpiteet käyttäjän tunnistamiseksi täytyy tehdä. Ensin tarkistetaan, löytyykö jo aiempaa istuntotietoa käyttäjästä, ja lisätään se tarvittaessa tietokantaan. Tietokannan generoima uid täytyy vielä erikseen lisätä istunnon tietoihin. Jos käyttäjällä on jo istuntotietoja, tarkistetaan kuitenkin, että uid löytyy vielä tietokannasta. Lopuksi palautetaan vastaus, eli tulostetaan tekstimuotoinen viesti palautettavalle sivulle.

```
case "0":     // NOT AUTHED

   if ($_SESSION['uid'] == null)   // uusi käyttäjä kantaan ja istuntoon
   {
      db_query("INSERT INTO users (uid) VALUES (null)");   // Huom! uid=autoinc
      $row = db_query("SELECT * FROM users ORDER BY users.uid DESC");
      $_SESSION['uid'] = $row['uid'];
   }
```

Liite 4: PHP-skripti

```
   else   // vanha käyttäjä kantaan jos ei vielä ole siellä
   {
      $row = db_query("SELECT * FROM users WHERE uid=".$_SESSION['uid']);
      if ($row == null)
         db_query("INSERT INTO users (uid) VALUES (".$_SESSION['uid'].")");
   }
   $_SESSION['mid'] = null;
   echo "OK";
   break;
```

Jokaiselle tilalle on siis oma segmenttinsä. Lopuksi pävitetään aina online-status eli asetetaan käyttäjä nähdyksi tietokantaan.

```
db_query("UPDATE users SET lastOnline='".date("Y-m-d H:i:s",time())."' WHERE
users.uid=".$_SESSION['uid']);
```

Kyselyn tullessa tilasta 7 eli ONLINE UPDATE tarkistetaan, onko yhteiseen peliin kuuluva vastustaja ollut aktiivinen ennalta määrätyn ajanjakson aikana. Tässä TIMEOUT-vakioksi on määritelty viisi sekuntia. Näin ollen pelaaja tippuu pois pelistä myös, mikäli yhteys palvelimeen on liian hidas.

```
case "7":     // ONLINE UPDATE

   $row = db_query("SELECT * FROM users,matches WHERE
   users.uid=matches.uid".$_SESSION['you']." AND lastOnline > '".date("Y-m-d
   H:i:s",time()-TIMEOUT)."'");

   if($row == NULL)
      echo "NO";   // poistunut linjoilta
   else
      echo "YES";  // linjoilla
   break;
```

Liite 5: AjaxCaller

Määritellään aluksi luokan konstruktori. Siinä ensimmäisenä parametrina saadaan kutsuttava URL-osoite. Koska palvelimen vastausten käsittelyyn ei voida käyttää nyt yhtä ja samaa funktiota, halutaan callback-funktion osoite toisena parametrina. Tiedot tallennetaan luokan sisäisiin muuttujiin.

```
function AjaxCaller( _url, _callback )
{
   this._url = _url;
   this._callback = _callback;

   this._xhr = this._createXHR();
   this._start();
}
```

XHR-olio luodaan sisäisessä metodissa _createXHR(), joka palauttaa viittauksen kyseiseen olioon riippumatta käytössä olevasta selaimesta. Konstruktorissa viittaus XHR-olioon sijoitetaan omaan muuttujaansa.

```
AjaxCaller.prototype._createXHR = function()
{
   if (window.XMLHttpRequest)
      return new XMLHttpRequest();
   else   // IE
      return new ActiveXObject("Microsoft.XMLHTTP");
}
```

Lopuksi konstruktorissa kutsutaan _start()-metodia, jossa aloitetaan pyynnön lähettäminen GET-metodia käyttäen. Onreadystatechange-ominaisuuden määrittelemisessä joudutaan käyttämään taas erikoisempaa ratkaisua. Käyttämällä viittausta "this._onReady" kutsu tapahtuisi window-olion _onReady()-metodiin, mutta sellaistahan ei ole olemassa. Sen sijaan tarvitaan viittaus AjaxCaller-olion onReady()-metodiin. Ratkaisuna viittaus AjaxCaller-olioon tallennetaan _this -apumuuttujaan.

```
AjaxCaller.prototype._start = function()
{
```

Liite 5: AjaxCaller

```
   this._xhr.open("GET", this._url, true);
   var _this = this;
   this._xhr.onreadystatechange = function(){ _this._onReady(); };
   this._xhr.send(null);
}
```

Tilan vaihtuessa tarkistetaan luonnollisesti, onko vastaus saapunut. Vastauksen ollessa valmis, kutsutaan tallennettua callback-funktiota. Parametrina annetaan XHR-olion palauttama vastaus. Koska callback-funktion nimi on tallessa, saadaan se suoritettua evaluoimalla JavaScriptin eval()-funktiota käyttäen.

```
AjaxCaller.prototype._onReady = function()
{
   if (this._xhr.readyState == 4 && this._xhr.status == "200")
      eval(this._callback+"(\""+this._xhr.responseText+"\");");
}
```

Uusi kutsu palvelimelle voidaan tehdä mistä tahansa yksinkertaisesti luomalla AjaxCaller-luokasta uusi olio. Ainoastaan callback-funktiot on kirjoitettava erikseen käsittelemään eri kyselyiden vastauksia. Callback-funktioista tarkemmin liitteessä 6.

```
new AjaxCaller("game.php?f=0", "notaAuthedCb");
```

Liite 6: Callback-funktiot

Palveimen lähettäessä vastauksen takaisin AjaxCaller-luokalle, kutsutaan ennalta määriteltyä callback-funktiota. Palvelimen vastaus saadaan parametrina. Callback-funktioihin on selkeyden vuoksi tässä työssä lisätty pääte "Cb".

Tietoa sovelluksen tilasta ei ole varsinaisesti tallennettu mihinkään muuttujaan, vaan esimerkiksi siiryminen tilaan READY tarkoittaa uuden AjaxCaller-olion luomista "readyCb" callback-osoitteella. Tarvittavat callback-funktiot ja niiden toiminta on esitelty taulukossa 1.

Taulukko 1. Callback-funktiot

function notAuthedCb(_response)
Jos vastaus on "OK", siirrytään tilaan AUTHED.

function authedCb(_response)
Jos "QUEUED", siirrytään tilaan QUEUED. Jos "FOUND", siirrytään tilaan MATCH-FOUND ja aloitetaan ONLINE UPDATE. Aloittaja asetetaan edellisen vastauksen mukaan.

function queuedCb(_response)
Sama kuin authedCb, paitsi aloittaja on jo määrätty.

function matchFoundCb(_response)
Jos "OK", vastustaja on löytynyt, ja siirrytään tilaan READY.

function readyCb(_response)
Jos "READY", aloitetaan peli. Jos aloittaja on vastustaja, siirrytään tilaan MATCH-WAIT. Aloitetaan peli luomalla olio Game-luokasta.
Jos "NOT READY", vastustaja ei ole valmis, ja pysytään tilassa READY.

function matchPlayCb(_response)
Jos "OK", oma liike on siirtynyt palvelimelle, ja siirrytään tilaan MATCH-WAIT.

function matchWaitCb(_response)
Jos "NO", vastustaja ei ole vielä liikkunut, ja pysytään tilassa MATCH-WAIT.
Muuten parsitaan vastuksena tullut vastustajan liike ja kutsutaan Game-luokan metodia makeMove().

function onlineCb(_response)
Jos "NO", vastustaja on poistunut, ja peli loppuu.
Jos "YES", läheteään taas uusi ONLINE UPDATE.

Liite 6: Callback-funktiot

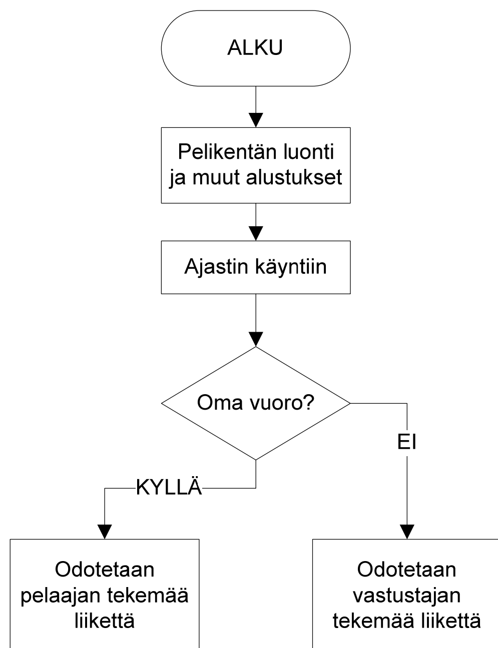
Esimerkiksi tilassa QUEUED asiakas on jonossa. Jonosta päästään pois silloin, kun joku muu pelaaja on aloittanut pelin lisäten jonossa olevan pelaajan tähän uuteen peliin. Jonottajan tehtävänä on siis kysyä palvelimelta, onko uusi ottelu lisätty, jossa olen mukana. Vastaus voi olla "FOUND", "NOT FOUND" tai "ERROR". Kaikki vastaukset on käsiteltävä. "FOUND" aloittaa valmistautumisen seuraavaan tilaan, "NOT FOUND" lähettää uusia kutsuja aina niin kauan kunnes vastustaja löytyy, ja "ERROR" tulostaa virheviestin.

```
function queuedCb( _response )
{
   if (_response == "FOUND")
      new AjaxCaller("game.php?f=3", "matchFoundCb");
   else if (_response == "NOT FOUND")
      new AjaxCaller("game.php?f=2", "queuedCb");
   else alert(_response);
}
```

Kun pelaajalla on pelivuoro eli ollaan tilassa MATCH-PLAY, kutsu ei voi tulla automaattisesti callback-funktioista. Pelaajan täytyy ensin pelata pelivuoronsa eli liikkua klikkaamalla hiiren nappia. Vasta sitten tieto liikkeestä voidaan lähettää palvelimelle. Callback-funktioiden käytöstä pelin aloittamisessa ja liikkumistietojen välittämisessä lisää liitteissä 8 ja 9.

Liite 7: Game-luokka

Kuten kuvan 1 vuokaaviosta nähdään, peli alkaa pelikentän luonnilla. Sen jälkeen käynnistetään ajastin, joka hoitaa pelikentän tapahtumien päivittämisen. Lopuksi jäädään odottamaan seuraavaa tapahtumaa. Omalla vuorolla tapahtuma on pelaajan klikkaus pelikentällä, vastustajan vuorolla se on vastustajan liikkeen sisältävä vastaus palvelimelta.



Kuva 1. Game-luokka: Aloitus.

Game-luokan metodit ja niiden lyhyet selitykset on esitelty taulukossa 1. Game-luokalla on oma tilamuuttujansa, jotta se on tietoinen siitä, missä vaiheessa ollaan ja kenen vuoro on meneillään. STATE_START_ME- ja STATE_START_YOU-tiloissa ollaan örkkien sijoitusvaiheessa, STATE_PLAY_ME- ja STATE_PLAY_YOU-tiloissa varsinaisessa pelissä.

Liite 7: Game-luokka

Taulukko 1. Game-luokan metodit.

function Game(_starter)
Konstruktorissa luodaan pelikentän elementit ja asetetaan aloittaja parametrina saadun tiedon mukaan.

Game.prototype.makeMove = function(_a, _b)
Tekee tarvittavan liikkeen (sijoittaa örkin tai ampuu kranaatin).

Game.prototype._putOrc = function(_x, _y)
Asettaa örkin kentälle _x ja _y parametrien määräämään paikkaan.

Game.prototype._update = function()
Päivittää pelikentän tapahtumat (tarkemmin liitteessä 8) .

Liite 8: Pelin aloittaminen

Pelin aloittaminen tapahtuu seuraavalla tavalla. Game-luokan olio luodaan, kun READY-tilan callback-funktiota kutsutaan. Pelin aloittaja on jo aiemmin määräytynyt jonottamaan joutuneen pelaajan eduksi.

```
function readyCb( _response )
{
   if (_response == "READY")
   {
      if (starter)   // pelaaja aloittaa
         game = new Game(true);
      else   // vastustaja aloittaa
      {
         new AjaxCaller("game.php?f=5", "matchWaitCb");   // MATCH-WAIT
         game = new Game(false);
      }
   }
   // ...
}
```

Liite 9: Mousedown ja mouseup

Koska pelialue on asetettu klikattavaksi, täytyy hiirenpainallukset "kaapata" ja käsitellä omassa funktiossaan. Sijoittamisvaiheessa hiiren alaspainallus aiheuttaa kutsun Game-luokan makeMove()-metodiin. Lisäksi palvelinta informoidaan tästä liikkeestä, ja uusi Ajax-kutsu lähetetään parametreinaan hiiren x- ja y-koordinaatit.

```
function mouseDown( e )
{
   if (game)
   {
      if (game.state == STATE_START_ME)   // örkkien sijoitusta
      {
         game.makeMove(getMouseX(e), getMouseY(e));
         new AjaxCaller
            ("game.php?f=6&a="+getMouseX(e)+"&b="+getMouseY(e),"matchPlayCb");
      }
      else if (game.getState() == STATE_PLAY_ME)
         shoot_cnt = new Date().getTime();
   }
}
```

Pelin ollessa käynnissä, hiiren napin päästäminen aiheuttaa kranaatin laukaisun voimalla, jonka napin pohjassaoloaika määrää, ja kulmassa, jonka sijainti tykin suhteen määrää. Nämä tiedot toki lähetetään myös palvelimelle.

```
function mouseUp( e )
{
   if (game)
   {
      if (game.state == STATE_PLAY_ME && !$("grenade"))
      {
         var angle = Math.atan((parseInt($("cannon").style.top)-getMouseY(e))/
           (getMouseX(e)-parseInt($("cannon").style.left)));
         var force = new Date().getTime()-shoot_cnt;
         game.makeMove(force, angle);
         new AjaxCaller("game.php?f=6&a="+force+"&b="+angle, "matchPlayCb");
         shoot_cnt = 0;
      }
   }
}
```

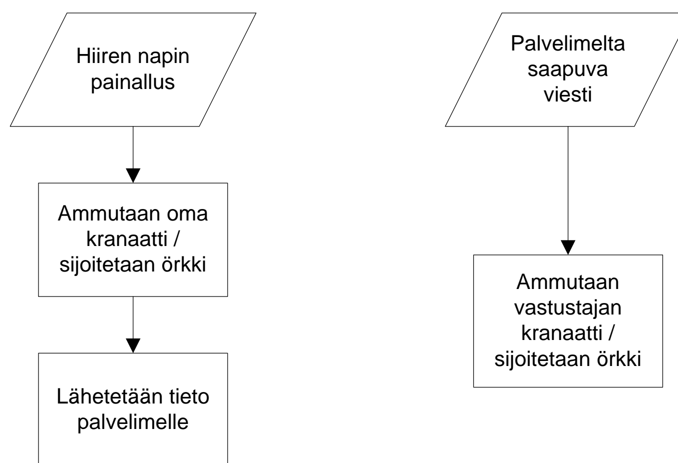
Liite 10: Liikkumistiedon hakeminen

Myös tieto vastustajan liikkeistä täytyy hakea ja suorittaa. Tilan 5 eli MATCH-WAIT callback-funktiossa vastaanotetaan tämä parametrina saatu liike. Liikkeen voima- ja kulmatieto ovat eroteltuina putki-operaattorilla. Liikkuminen voidaan suorittaa samanlailla kuin omakin pelaajan liikkuminen, eli Game-luokan makeMove()-metodilla. Game-luokan oliollahan on jo tieto vuorosta, joten liikkeet eivät mene siltä osin sekaisin.

```
function matchWaitCb( _response )
{
   if (_response == "NO")
      new AjaxCaller("game.php?f=5", "matchWaitCb");
   else if (_response == "ERROR")
      alert(_response)
   else   // vastustaja liikkunut
   {
      var temp = _response.split("|");
      game.makeMove(parseFloat(temp[0]), parseFloat(temp[2]));
   }
}
```

Liite 11: Liikkuminen

Kuvan 1 vuokaaviosta on nähtävissä, miten Game-luokka reagoi pelaajien liikkumisiin. Viimeisenä tapahtumana voisi olettaa vuoron vaihtuvan. Vuoro vaihtuu kyllä örkkiä sijoittavassa metodissa _putOrc() välittömästi, mutta jos ollaan varsinaisessa pelissä ja ammutaan kranaatti, vuoro vaihtuu vasta kun kranaatti on räjähtänyt.



Kuva 1. Game luokka: Liikkuminen.

Kun pelaaja on oman liikkeensä tehnyt tai palvelimelta on tullut tieto vastustajan liikkeestä, työhön joutuu Game-luokan makeMove()-metodi. Se asettaa oman tai vastustajan örkin kentälle tai ampuu kranaatin.

```
Game.prototype.makeMove = function( _a, _b )
{
   if (this._state == STATE_START_ME || this._state == STATE_START_YOU)
      this._putOrc(_a, _b);

   else if (this._state == STATE_PLAY_ME || this._state == STATE_PLAY_YOU)
      this._grenade = new Grenade(_a, _b);

}
```

Metodi _putOrc() nimensä mukaisesti sijoittaa örkin pelilaudalle ja vaihtaa vuoroa. Jos kaikki örkit ovat jo sijoitettuna, varsinainen peli alkaa.

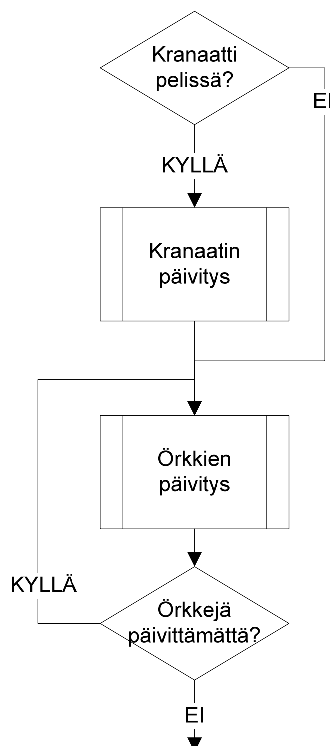
Liite 11: Liikkuminen

```
Game.prototype._putOrc = function( _x, _y )
{
   if (this._state == STATE_START_YOU)
   {
      this._your_orc_cnt++
      this._orcs.push(new Orc("your"+this._your_orc_cnt, false, _x, _y));
      this._orcs_divs.push("your"+this._your_orc_cnt);

      if (this._my_orc_cnt+this._your_orc_cnt == MEN_COUNT*2)
         this._state = STATE_PLAY_ME;
      else
         this._state = STATE_START_ME;
   }
   else if (this._state == STATE_START_ME)
   {
      // ...
   }
}
```

Liite 12: Päivityssykli

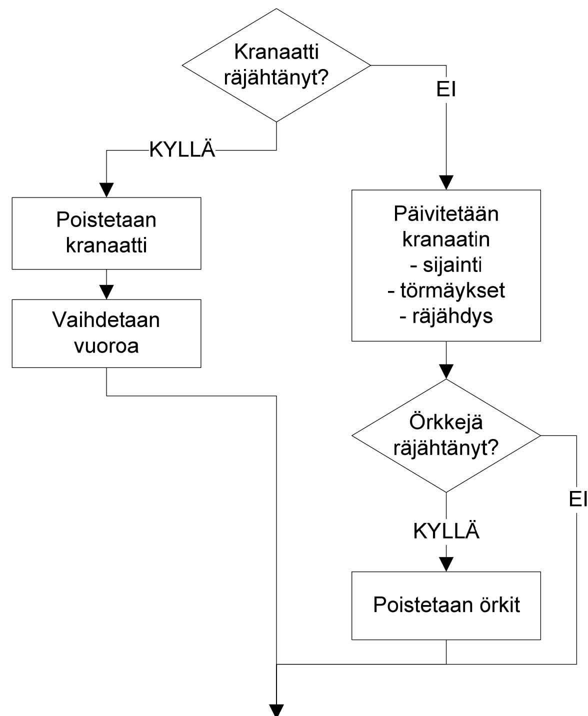
Jatkuvaa pelin tapahtumien päivittämistä varten asetettiin ajastin, joka suorittaa halutun metodin aina määrätyin intervallein. Päivityssyklin eli Game-luokan update()-metodin tapahtumat ovat pääpiirteissään esitelty kuvan 1 vuokaaviossa. Tässä metodissa hallitaan kaikkia pelin tapahtumia.



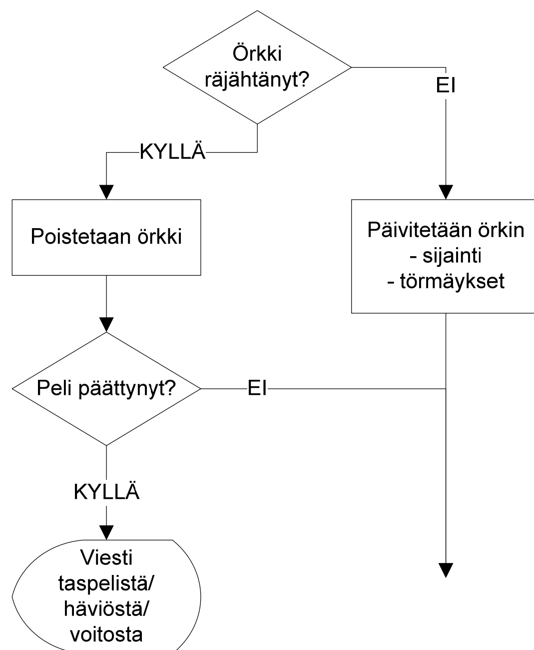
Kuva 1. Game-luokka: Päivityssykli.

Tarkemmin kranaatin päivitys tapahtuu kuvassa 2 esitetyn vuokaavion tavoin. Ainoastaan kranaatin päivitys aiheuttaa kutsuja Grenade-luokan metodeihin, muuten kaikki tapahtuu päivityssyklissä. Örkkien päivitykset puolestaan tapahtuvat kutakuinkin kuvassa 3 esitetyllä tavalla. Jokainen örkki käy tämän päivityssyklin läpi.

Liite 12: Päivityssykli



Kuva 2. Game-luokka: Kranaatin päivitys.



Kuva 3. Game-luokka: Örkkien päivitys.

Liite 12: Päivityssykli

Päivityssyklin voisi sanoa olevan koko pelin sydän. Se käy kaikki päivitettävät oliot läpi ja kutsuu tarvittavia metodeja. Kranaatin päivitykset tapahtuvat luokkansa omissa metodeissaan ja örkkien omissaan.

Kranaatin tapahtumia päivitettäessä örkki poistetaan pelikentältä, jos sen on tuhottu. Samalla olisi voitu poistaa myös örkin olio, mutta tässä örkkien hallinta on eriytytetty omaan silmukkaansa. Silmukassa katsotaan onko tuhottuja örkkejä _orc_divs-taulukossa, ja sen mukaan poistetaan örkin olio ja päivitetään pelin tilannetta. Orc-luokalla on julkinen muuttuja "mine", joka kertoo onko kyseeessä oma örkki. Tämän avulla voidaan pitää kirjaa siitä, montako örkkiä kummallakin pelaajalla on jäljellä, ja lopettaa peli tarvittaessa.

Päivityssykli eli Game-luokan _update()-metodi on kokonaisuudessaan seuraavanlainen:

```
Game.prototype._update = function()
{
   if (this._grenade)   // kranaatti pelissä?
   {
      if ($("grenade"))
      {
         this._grenade.update();
         this._grenade.checkCollisions(this._object_divs);
         var o = this._grenade.checkExplosions(this._orc_divs);   // örkki räjähtänyt?
         if (o) removeObject(o);   // poistetaan räjähtänyt örkki
         this._grenade.paint();
      }
      else   // kranaatti räjähtänyt, vuoro vaihtuu
      {
         this._grenade = null;
         if (this._state = STATE_PLAY_ME)
            this.state = STATE_PLAY_YOU;
         else
            this.state = STATE_PLAY_ME;
      }
   }
```

Liite 12: Päivityssykli

```
   for (var i in this._orcs)
   {
      if ($(this._orc_divs[i]))
      {
         this._orcs[i].update();
         this._orcs[i].checkCollisions(this._object_divs);
         this._orcs[i].paint();
      }
      else
      {
         if (this._orcs[i]._mine)
            this._my_orc_cnt--;
         else
            this._your_orc_cnt--;
         delete this._orcs[i];

         if (this._my_orc_cnt == 0 && this._your_orc_cnt == 0)
            msgBox.displayTie();
         else if (this._my_orc_cnt == 0)
            msgBox.displayLose();
         else if (this._your_orc_cnt == 0)
            msgBox.displayWin();
      }
   }
}
```

Liite 13: Grenade-luokka

Grenade-luokassa hoidetaan kaikki kranaatin päivitykset. Sen metodit on esitelty taulukossa 1.

Taulukko 1. Grenade-luokan metodit

function Grenade(_v, _deg)
Alustaa muuttujia ja asettaa lähtökulman ja nopeuden kranaatille.

Grenade.prototype.update = function()
Päivittää kranaatin nopeutta ja sijaintia.

Grenade.prototype.checkCollisions = function(_o)
Tarkastaa törmäykset parametrina saaman taulukon elementteihin ja muuttaa suuntaa tarvittaessa kutsumalla bounce-metodeita.

Grenade.prototype.checkExplosions = function(_o)
Tarkastaa kranaatin räjähtäessä osumat parametrina saatuun taulukkoon örkeistä ja palauttaa viittauksen räjähtäneeseen örkkiin.

Grenade.prototype.paint = function()
Piirtää kranaatin uuden sijainnin ja päivittää tarvittaessa räjähdysanimaatiota.

Grenade.prototype._setExplosionImage = function()
Asettaa räjähdyksen animaatioliuskan ja muut ominaisuudet paikalleen.

Grenade.prototype._getObjectSector = function(_o, _x, _y)
Palauttaa osumasektorin.

Grenade.prototype._bounceUp = function(_y)
Grenade.prototype._bounceDown = function(_y)
Grenade.prototype._bounceLeft = function(_x)
Grenade.prototype._bounceRight = function(_x)
Muuttavat kranaatin nopeutta ja sijaintia kimpoamisen seurauksena.

Liite 14: Törmäykset

Grenade-luokan checkCollisions()-metodi saa parametrinaan taulukon elementeistä. Taulukossa on elementtien nimet, joita vastaan törmäyksiä tarkastellaan.

```
Granade.prototype.checkCollisions = function( _o )
{
   for (var i in _o)        // käydään jokainen elementti läpi
   {
      // haetaan edellinen ja nykyinen sektori
      var sector = this._getObjectSector($(_o[i]), this._x, this._y);
      var old_sector = this._getObjectSector($(_o[i]),
         this._old_x_pos,this._old_y_pos);

      if (sector == 'B2')  // törmäys...
      {
         switch (old_sector)
         {
            case 'A':      // ...vasemmalta
               this._bounceLeft(parseInt($(_o[i]).style.left)-this._w);
               break;
            case 'C':      // ...oikealta
               //...
         }
      }
   }
}
```

Checkcollisions()-metodissa siis kutsutaan _getObjectSector()-metodia sekä tehdään päätöksiä edellisestä ja nykyisestä osumasektorista riippuen. Kranaatin tulosuunnan hahmottamista varten tarvittiin osumasektorin laskeva metodi, getObjectSector().

Olennaista törmäyksien tunnistamisessa on tietää vertailtavien elementtien sijainnit. Kranaatin sen hetkinen sijainti on tallessa this._x- ja this._y-muuttujissa, ja myös sen leveys ja korkeus on konstruktorissa tallennettu this._w- ja this._h-muuttujiin nopeampaa viittausta varten. Taulukossa olevan elementin vastaavat parametrit saadaan selville yksinkertaisesti sen style-atribuuttien avulla. Myös tietoa kranaatin edellisestä sijainnista tarvitaan. Se on tallennettu kranaatin update()-metodissa.

Liite 14: Törmäykset

Metodi _getObjectSector() laskee ja palauttaa kranaatin osumasektorin sille parametrina annettujen kranaatin sijainnin ja vertailtavan elementin perusteella. Parametrit _x ja _y ovat kranaatin sijaintitiedot (vanha tai nykyinen) ja _o vertailtava elementti.

```
Grenade.prototype._getObjectSector = function( _o, _x, _y )
{
   if (_x+this._w < parseInt(_o.style.left))
      return 'A';
   else if (_x > parseInt(_o.style.left)+parseInt(_o.style.width))
      return 'C';
   else
   {
      if (_y+this._h < parseInt(_o.style.top))
         return 'B1';
      else if (_y > parseInt(_o.style.top)+parseInt(_o.style.height))
         return 'B3';
      else
         return 'B2';
   }
}
```

CheckExplosions()-metodi käyttää myös _getObjectSector()-metodia. Siinä tarkastetaan, onko kranaatin räjähdysalue jonkun elementin (örkin) sisällä. Jotta noudatettaisiin olio-ohjelmoinnin ideologiaa, ei örkkiä voida suoraan poistaa kentältä sen räjähdettyä. Kyseessähän on Game-luokan sisäinen muuttuja. Vain viittaus tähän örkkiin palautetaan ja örkki poistetaan lopullisesti vasta Game-luokan _update()-metodissa.

```
Grenade.prototype.checkExplosions = function( _o )
{
   if (this._exploding)
   {
      for (var i in _o)
      {
         if ($(_o[i]))     // sisältää null-olioita jos olioita jo räjäytetty
         {
            if (this._getObjectSector($(_o[i]), this._x, this._y) == "B2")
```

Liite 14: Törmäykset

```
                return _o[i];
            }
        }
    }
    return null;
}
```

Liite 15: Räjähdys

Kranaatin update()-metodissa lasketaan aikaa kranaatin räjähtämiseen. Ja kun aika on täynnä kranaatti räjähtää asettaen apumuuttujan totuusarvon todeksi.

```
this._moveTime += 0.04;
if (this._moveTime > 4) this._exploding = true;
```

Kranaatin sijainti piirretään näytölle _paint()-metodia kutsuttaessa. Yleensä kranaatin kuva piirretään sijaintiin, jonka sen hetkiset this._x- ja this._y-muuttujat määrävät, käyttäen normaalia staattista kuvaa. Mutta jos on räjähdys käynnissä, piirretään räjähdyksen animaatio.

Räjähdyksen alkaessa kutsutaan _setExplosionImage()-metodia, joka vain asettaa animaatioliuskan paikalleen style-atribuuteilla näyttämään ensimmäistä kuvaa. Taustakuvaa siirretään aina yhden kuvan leveyden verran. Kun animaatio on käyty läpi, kranaatti on räjähtänyt ja se voidaan poistaa näkyviltä. Koko Grenade-oliota ei voida itse oliosta käsin poistaa, vaan se poistuu Game-luokan _update()-metodin kautta.

```
Grenade.prototype.paint = function()
{
   if (!this._exploding)
   {
      $(this._name).style.top = parseInt(this._y) +"px";
      $(this._name).style.left = parseInt(this._x) +"px";
   }
   else       // päivitetään räjähdysanimaatio
   {
      if (parseInt($(this._name).style.backgroundPosition) == 0)   // aloitetaan
         this._setExplosionImage();
      else if (parseInt($(this._name).style.backgroundPosition) == -500)   // lopetetaan
      {
         removeObject(this._name);
         return;
      }
      $(this._name).style.backgroundPosition =
         parseInt($(this._name).style.backgroundPosition) - 100 + "px";
   }
}
```

Liite 16: Orc-luokka

Örkkien hallintaan on myös oma luokka. Nämä Orc-oliot ovat sijoitettuina Game-luokan _orcs-taulukkoon ja niiden div-olioiden nimet _orc_divs-taulukkoon. Game-luokan _update()-metodissa kutsutaan myös örkkien metodeita päivittämään näiden liikkeet ja tarkistamaan törmäykset pelin elementtien kanssa. Nämä ovat kuitenkin yksinkertaisempia metodeja kuin kranaatin vastaavat metodit, koska örkit eivät voi liikkua, ainoastaan tippua alaspäin sijoittamisen jälkeen. Orc-luokan metodit on esitelty taulukossa 1.

Taulukko 1. Orc-luokan metodit.

function Orc(_name, _mine, _x, _y)
Alustaa muuttujia ja luo örkin kentälle parametrina annettuun sijaintiin.

Orc.prototype.update = function()
Päivittää örkin sijaintia.

Orc.prototype.paint = function()
Piirtää örkin päivitettyyn sijaintiinsa.

Orc.prototype.checkCollisions = function(_o)
Tarkastaa örkin törmäyksen elementteihin sen tippuessa ja muuttaa sijaintia tarvittaessa.