LAPPEENRANNAN TEKNILLINEN YLIOPISTO
Teknistaloudellinen tiedekunta
Tietotekniikan osasto

Diplomityö

Marko Suhonen

KERÄILYKORTTIPELIN TOTEUTUS HTML5- JA JAVASCRIPT-TEKNIIKOILLA

Työn tarkastajat: Professori Kari Smolander
DI Jani Rönkkönen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto

Tietotekniikan osasto

Ohjelmistotekniikan koulutusohjelma

Marko Suhonen

Keräilykorttipelin toteutus HTML5- ja JavaScript-tekniikoilla

Diplomityö

2013

70 sivua, 16 kuvaa ja 11 taulukkoa

Tarkastajat: Professori Kari Smolander

DI Jani Rönkkönen

Hakusanat: Animaatio, HTML5, WebSocket, piirtoalue, Socket.io, keräilykorttipeli

Keywords: Animation, HTML5, WebSocket, Canvas, Socket.io, collection card game

HTML5-tekniikka sekä Javascript tuen laajuus selaimissa vaihtelee. Tässä työssä kyseisiä tekniikoita tutkitaan ja selvitetään niiden toimivuus keräilykorttipelin tarvittavien ominaisuuksien osalta. Tuet kartoitetaan viiden yleisimmän selaimen osalta. HTML5 tukee WebSocket-ominaisuutta, mutta kaikki selaimet eivät tue ominaisuutta tai se on poistettu käytöstä. Työssä etsitään tiedonsiirtotekniikan korvaaja, jota testataan ja verrattaan yleisesti käytettäviin tekniikoihin. Socket.io oli nopea tekniikka ja viisi yleisintä selainta tuki kyseistä tekniikkaa. Tämän vuoksi Socket.io-tekniikka soveltuu keräilykorttipeliin hyvin. Työssä tutkitaan keräilykorttipeliin liittyviä ongelmia sekä ratkaistaan ilmenneet ongelmat. Keräilykorttipelissä kyseisiä ongelmia ilmeni hyvin vähän. HTML5 animaatio tutkitaan että se on optimoitu hyvin, jotta käyttäjälle tulee miellyttävä peli kokemus. Keräilykorttipeliin lisäksi tehdään käytännön toteutuksena pakkaeditorin prototyyppi, jossa käytetään *drag&drop*-tekniikkaa. Tämän vuoksi myös *drag&drop*-tekniikan tuki on selainten osalta kartoitettu myös työssä, sekä testattu käytännön toteutuksena prototyypissä. Prototyypin tarkoitus on kartoittaa mahdolliset tulevat ongelmat sekä auttaa varsinaisen pakkaeditorin tuotantoversiossa.

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology

Marko Suhonen

Collectioncard game implementation with HTML5 and JavaScript

Master's thesis
2013

70 pages, 16 images and 11 tables

Examiner: Professor Kari Smolander
DI Jani Rönkkönen

Keywords: Animation, HTML5, WebSocket, Canvas, Socket.io, collection card game

HTML5 and Javascript are seen as the fundamental techniques for building modern web applications. Unfortunately, their support in current state-of-the-art web browsers varies. In this thesis those technologies are researched and checked if the functionality of the collection card game can be done without any major changes. Support for HTML5 and Javascript are studied in five most popular web browsers. HTML5 defines WebSocket technology but all browsers do not support it or it disable it by default. In this work, we will find replacement for the data transfer technology, test it and compare it to the most common data transfer technologies. Socket.io was found to be the best choice for the collection card game because it is fast and is well supported by the five most popular web browser. Problems and solutions that occurred in collection card game is explained in this work. Only few problems occurred in the collection card game. HTML5 animation is checked in the work if it is optimized for the game, so the player will have good gaming experience. Deck editor prototype is made for practical work to test functionality of the upcoming production version. In the deck editor player can make own decks for the game with drag&drop-functionality. Drag&drop support is also researched with this thesis, and tested with practical work so the production version will have all the upcoming problems solved.

ALKUSANAT

Tämä diplomityö on tehty vuosina 2011-2013 aikana Lappeenrannan teknillisessä yliopistossa.

Aluksi haluaisin kiittää Seepia Games Oy:ä diplomityön aiheesta, sekä tuesta jota olen saanut. Kiitän erityisesti ohjaajaani, Jani Rönkkösen, tuesta ja neuvoista, joita sain koko diplomityön kirjoittamisen aikana. Hän ohjasi diplomityötä oikeaan suuntaan, kyseli kaikkea mahdollista ja laittoi ajattelemaan hieman avarammin ongelmia ja asioita, joita ilmeni työn tekemisen ohessa.

Lopuksi kiittäisin perhettäni tuesta ja kannustuksesta, jota olen saanut vuosien aikana.

Sisällysluettelo

LYHENNELUETTELO

API	Application programmin interface, Ohjelmointirajapinta
CERN	The European Particle Physics Laboratory
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
ECMA	European standards committee
FPS	Frames per second
HTML	Hypertext Markup Language
IETF	The Internet Engineering Task Force
IE9	Internet Explorer 9
IE10	Internet Explorer 10
LAN	Local Area Network
MAN	Metropolitan Area Network
PHP	Hypertext Preprocessor
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
WAN	Wide Area Network
WWW	World Wide Web
W3C	World Wide Web Consortium
XHTML	eXtensible Hypertext Markup Language
XML	Extensible Markup Language

1 JOHDANTO

Verkkosivujen toteutuksessa käytetyt tekniikat ja standardit ovat kehittyneet viime vuosikymmeninä nopeasti. Aluksi World Wide Web (WWW) oli tekstiin pohjautuva, mutta nykyisin on mahdollista liittää WWW-sivuille hyvin monipuolista sisältöä. WWW-sivujen tarkoitettua HTML (Hypertext Markup Language) tekniikkaa on kehitetty vuodesta 1991 lähtien. Uusin versio HTML5 tuo paljon mahdollisuuksia, kuten videon lisäystä verkkosivuille sekä piirtämistä, jolla voidaan tehdä animaatiota sekä erilaisia graafeja HTML-sivuille. Samalla se tuo rajoituksia verkkosivuihin, koska HTML5-tuki vaihtelee, eivätkä kaikki selaimet tue kaikkia uusia ominaisuuksia. Tämä johtuu selainten käyttämistä selainmoottoreista, jotka tukevat HTML5:stä eri tavoin. Selaimet käyttävät myös omia moottoreitaan JavaScriptin tulkitsemiseen ja suorittamiseen, minkä vuoksi selaimet käyttäytyvät eri tavalla, kun ne suorittavat JavaScriptillä ohjelmoituja sivuja.

1.1 Tausta

Yritys nimeltään Seepia Games Oy on tekemässä selaimella käytettävää keräilykorttipeliä. Keräilykorttipelin asiakasohjelma toteutetaan aluksi Chrome-selaimella, mutta aikomus on saada ohjelma toimimaan viidellä yleisimmällä selaimella (Chrome, Internet Explorer, Firefox, Safari ja Opera), sekä myös mobiililaitteissa (iPad, Android). Asiakasohjelma tehdään käyttäen HTML5- ja JavaScript-tekniikoita. Selaimet tukevat tekniikoita eri tavalla ja tukien laajuus vaihtelee. Tämän vuoksi on tarvetta tutkia kuinka selainten tuki näille tekniikoille eroaa, sekä kehittää menetelmiä joilla asiakasohjelma saadaan toimimaan vastaavalla tavalla eri selaimissa.

1.2 Tutkimusongelma, tavoitteet ja rajaukset

Tässä työssä keskitytään etenkin keräilykorttipelin käyttöliittymäosuuteen, kuten kuvien piirtoon ja toiminnallisiin osuuksiin. Diplomityössä selvitetään, miten eri selaimet tukevat HTML5- ja JavaScript-tekniikoita. Aihe rajataan keräilykorttipelin asiakasohjelmaan. Tässä

työssä keskitytään asiakasohjelman HTML5- ja JavaScript-osiin. Vertailuun sisällytetään viisi suosituinta selainta: Chrome, Internet Explorer, Firefox, Safari ja Opera. Selaimista käytetään tutkimusajankohdan aikana yleiseen käyttöön julkaistuja uusimpia versioita, mutta näiden lisäksi kartoitetaan selainten tulevien versioiden ominaisuudet. Lisäksi aihe rajataan asiakasohjelman käyttöliittymän toteutukseen sisältäen palvelin – asiakasohjelman kommunikaatiotapojen vertailun.

Työn tavoitteena on saada vastaukset seuraaviin kysymyksiin:

- Mitä puutteita eri selainten HTML5-tuessa on?
- Miltä osin JavaScript-koodi eroaa selainten välillä?
- Miten Chromelle suunniteltu sovellus saadaan toimimaan myös muilla selaimilla?

Lisäksi työssä tavoitteena on dokumentoida tulokset korttipelin testauksesta, sekä yhteensopivuudesta eri selainten kanssa. Lopullisena tavoitteena on saada korttipelin asiakasohjelma toimimaan kaikilla selaimilla. Tämän lisäksi toteutetaan käytännön työnä pakkaeditorin prototyyppi HTML5-tekniikalla, jossa pelattavat korttipakat rakennetaan. Pakkaeditorissa hyödynnetään työssä tehtyä tekniikoiden esiselvitystä.

Tässä työssä on tarkoitus selvittää aiempia tutkimuksia laajemmin HTML5:n sekä JavaScriptin tuki etenkin eri selainten osalta. Työssä keskitytään myös eri tiedonvälitystekniikoihin muun muassa Socket.io:on, HTML5 WebSocket:iin, XMLHttpRequest- ja Ajax-tekniikkoihin ja vertaillaan tekniikoita keskenään, sekä selvitetään tekniikoiden toimivuus viidellä yleisimmillä selaimella. Muita tiedonsiirtotekniikoita otetaan mukaan kirjallisuuden sekä löydettyjen viitteiden löytämien tietojen perusteella. Näitä ovat muun muassa Faye, Comet ja jWebSocket. Kyseisistä tekniikoista tutkitaan niiden tarpeellisuus, hyödyllisyys keräilykorttipelille, jossa vaaditaan nopea ja vähän kuormittava tekniikka.

1.3 Työn rakenne

Luvussa 2: Selainsovellusten tekniikat, esitellään työhön liittyvät teknologiat ja niiden historia. Luvussa lisäksi selvitetään, miten uusi HTML5-teknologia eroaa aikaisimmista versioista ja mitä uusi versio tuo mukanaan. Javascript teknologia esitellään sekä sen historia. Liitännäisteknologioista (Flash, PHP, CGI) kerrotaan lyhyesti mitä ne ovat ja mitä niillä voi tehdä. Myös tekniikoiden eroavaisuuksia toisiinsa nähden on selvitetty kyseisissä osioissa. Luvussa käydään myös läpi aiemmat tutkimukset, sekä paneudutaan enemmän selainten kehitystyökaluihin, sekä niiden eroihin.

Tutkimuskohteen ongelmat-luvussa (Luku 3) käydään läpi keräilykorttipelissä ilmenneet ongelmat, mahdolliset parantamisalueet, sekä ratkaistaan ongelmat. Ongelmina käydään läpi muun muassa satunnaislukugeneraattori sekä kommunikaatiotekniikan valinta. Tämän lisäksi tutkitaan voidaanko animaatiota parantaa ”Animaatio”-luvussa.

Tutkimuskohteen jälkeen tulee Luku 4,”Keräilykorttipelin pakkaeditori”, jossa selitetään kuinka tehtiin pakkaeditorin prototyyppi keräilykorttipeliin. Luku jakautuu vaatimukset, toteutus sekä lopputulos-osioihin. Vaatimukset luvuissa kerrotaan pakkaeditorin vaatimukset ja toteutuksessa mitä ja kuinka ominaisuudet toteutettiin. Lopputulos-osiossa verrataan vaatimuksia toteutettuihin ominaisuuksiin, sekä kerrotaan puuttuvat ominaisuudet. Diplomityön lopussa on yhteenveto luku, jossa kirjoitetaan pääasiat mitä työssä on tehty, sekä johtopäätökset.

2 Selainsovellusten tekniikat

2.1 HTML

Hypertext Markup Language (HTML) on kuvauskieli, jota käytetään tuottamaan dokumentteja WWW:n. HTML:llä kuvataan WWW-sivujen rakenne ja ulkoasun asettelu HTML:n elementtien avulla, jotka koostuvat tageistä (merkinnöistä) kuten ”*<html>*”. Näiden avulla voidaan lisätä tekstiä ja grafiikkaa WWW-sivuille. [1,2]

Merkintöjen avulla tehdään elementtejä kuten otsikoita ja taulukoita. Suurin osa HTML elementeistä alkaa aloitus merkillä, jonka jälkeen tulee elementin sisältö. Tämän jälkeen tulee lopetusmerkki. Tällä tavalla tehdään muun muassa otsikko seuraavalla tavalla: ”*<title>Otsikko</title>*”. Elementtien sisällä voi olla tekstiä tai toinen elementti. Elementtien pitää olla sisäkkäin, ilman että ovat toistensa päällä. Tämä tarkoittaa sitä, että elementit lopetetaan sisin elementti ensin. Esimerkiksi hyvästä rakenteesta, jossa strong-elementti lopetetaan ensin: ”* Tämä lause onoikein. ”*

Osalla elementeillä ei ole kuin aloitusmerkki ilman lopetusmerkkiä. Tällainen on esimerkiksi elementti, jolla tehdään rivinvaihto. Tällöin käytetään merkkiä *'
'*. [2, 3, 4]

Elementeillä voi olla ominaisuuksia (attribuutteja), jotka määräävät miten elementti toimii. Tällöin esimerkiksi hyperlinkki tehdään käyttäen *a*-elementtiä ja tällä on *href*-ominaisuus (attribuutti):

Esimerkki

Ominaisuudet sijaitsee aloitusmerkin sisällä ja sisältää nimen sekä arvon. Nämä erotetaan yhtäsuuruus merkillä (”=”). Ominaisuudet yleensä kirjoitetaan lainausten väliin, joko ” tai ' merkkiä käyttäen. Lainauksia ei tarvitse jos attribuuttien arvossa ei ole välilyöntiä tai seuraavia merkkejä " ' ` = < tai >. Selaimet jäsentävät nämä kuvaukset DOM (Document Object Model) puuksi.[2, 3]

Sivun rakenne aloitetaan ”*DOCTYPE*”-merkinnällä, joka on HTML4:ssa seuraavanlainen:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"#
"http://www.w3.org/TR/html4/loose.dtd">

Tämän jälkeen tulee *<HTML>*-merkintä, sekä otsikkotiedot. Otsikkotiedot aloitetaan *<head>*-merkinnällä. Otsikkotietoihin tulee merkistön määritys *meta*-elementissä:

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

Meta-elementteihin tarkennetaan www-sivun kuvaus kuten sivun tekijä, avainsanat, sekä muut mahdolliset tiedot sivustosta. *Meta*-elementin lisäksi otsikkotietoihin voidaan laittaa sivun otsikko *<title>*-merkintään ja tyylitiedostoon linkki esimerkiksi *<link rel=”stylesheet href=”tiedontonnimi.css”>*. Tällöin esimerkki HTML-tiedosto on alla olevan esimerkin mukainen (HTML5 mukainen esimerkki) [5]:

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8" >
                <title>HTML5</title>
                <link rel="stylesheet" href="html5.css">
        </head>
        <body>
        <!-- tähän tulee sivuston sisältö-->
        </body>
</html>
```

2.1.1 Historia

Vuonna 1991 ilmestyi tusina merkintöjä liittyen HTML:n [1]. Ensimmäiset määrittelyt julkaistiin vuosina 1990-1995 aluksi CERN:in ja myöhemmin IETF:n (The Internet Engineering Task Force) toimesta [2, 6]. Tällöin HTML oli tekstiin pohjautuva, eikä siinä ollut mahdollisuutta lisätä grafiikkaa. HTML 2.0 versio standardoitiin vuonna 1995, jolloin mukaan lisättiin muun muassa kuvat, täytettävät lomakkeet ja hyperlinkit [1,3, 4]. HTML

3.0:sta julkaistiin määrittely vuonna 1995, mutta kolmannen version luonnosmäärittely ei saanut suositusta standardiksi, koska sen ympärillä oli paljon väittelyä [2]. Kolmas versio olisi pitänyt sisällään taulukot, matemaattisia kaavoja ja kuvioita. HTML 3.2 julkaistiin 1996 W3C:n (World Wide Web Consortium) toimesta, koska IETF oli lakkautettu [1]. 3.2 versiossa oli 3.0 version ominaisuudet mukana. HTML 4.0 julkaistiin 1998, jonka mukana tuli mekanismi tyyliasuille (Style Sheets), upotetut objektit ja parannellut lomakkeet [1]. Ensimmäinen luonnos HTML5:sta julkaistiin 2008, joka tukee upotettua graafista ja multimediasisältöä [1, 3, 7]. HTML5-suosituksen on tarkoitus olla valmiina 2014. Nykyään HTML5 on niin sanotussa ”Last Call” vaiheessa. Kyseisessä vaiheessa on lähetetty kutsu yhteisöille, jolloin varmistetaan teknisen spesifikaation pätevyys ja eheys yhteisöjen avulla. Yhteisöihin kuuluu yrityksiä, jotka ovat niin W3C:n piirissä ja sen ulkopuolella. [8]

2.1.2 HTML5

Suurin ero HTML5:lla on edellisiin versioihin se, että sillä voi kirjoittaa niin HTML:n kuin XHTML:n (eXtensible Hypertext Markup Language) syntaksilla. HTML5 on suunniteltu etenkin multimedian ja graafisen sisällön parantamiseksi ja tätä varten siinä on paljon uusia ominaisuuksia HTML4:een verrattuna. Näitä ominaisuuksia ovat muun muassa video- ja audio-ominaisuudet, joiden avulla voidaan lisätä multimediasisältöä sivuille.

Toinen uusi ominaisuus ovat attribuutit, joita voidaan muun muassa käyttää ”*Drag and Drop*” ohjelmointirajapinnan (Application programming interface, API) kanssa. ”*Drag and drop*”:lla tarkoitetaan ominaisuutta, jolla voidaan ottaa tekstiä, kuvaa ja ”raahata” niitä toisten elementtien sisään, jopa toiseen selaimeen tai työpöydälle. ”*Drag and Drop*”-ominaisuutta saadaan aikaan attribuuttien avulla, jolloin esimerkiksi * *-elementille laitetaan ”*draggable*”-attribuutti seuraavalla tavalla: * *. Tämän lisäksi pitää tarkistaa vetotapahtumia kuten *dragstart*, *drag*, *dragenter*, *dragleave*, *dragover*, *drop*, *dragend*. Tapahtumien seuraamisella voidaan muun muassa varmistaa, että elementti siirretään oikeaan paikkaan. Siirto pitää tapahtua ”*dropzone*”-ominaisuudella olevaan elementtiin. Elementin pitää kuunnella ”*drop*”-tapahtumaa.[9]

Piirtoalue (”*Canvas*”) on yksi HTML5:en tärkeimmistä uudistuksista. Tämä antaa mahdollisuuden ohjelmoijan liittää animaatiota, videota ja grafiikkaa suoraan verkkosivuihin, sekä antaa mahdollisuuden luoda niitä. HTML-kieli ei sovi suoraan monimutkaisen ja dynaamisen sisällön määrittelyyn, vaan avuksi tarvitaan muita tekniikoita kuten esimerkiksi JavaScript-kieli. [1, 10]

HTML5-tekniikka mahdollistaa sovellukset, jotka toimivat ilman jatkuvaa internet-yhteyttä. Tämä onnistuu HTML5:n uuden ominaisuuden avulla: sivustot voivat määritellä evästeiden tallentamisen säännöt omalla manifestilla. Manifesti määrittää mitkä resurssit voidaan turvallisesti tallentaa evästeisiin. Tällöin kun asiakasohjelma ei ole kytkettynä internetiin, käyttäjä voi käyttää ohjelmaa normaalisti paitsi esimerkiksi silloin kun tarvitsee palvelimen tietoja, joita ei voitu turvallisesti tallentaa.[11] Muita keskeisiä ominaisuuksia HTML5-tekniikassa on geologinen sijainti, sekä ”*Server-sent*”-ominaisuus, jolla voidaan palvelimelta lähettää tietoa asiakasohjelmaan (selaimeen). Lisäksi HTML5 tukee *Scalable Vector Graphics* (SVG) -formaattia, jota käytetään kuvaamaan kaksiulotteista grafiikkaa XML-muodossa (Extensible Markup Language). Tätä käytetään etenkin vektorityyppisissä kuvissa, muun muassa piirakkadiagrammeissa. Vektorigrafiikan etuna on sen hyvä skaalautuvuus. [5]

HTML5:ssa on myös ohjelmointia helpotettu ja yksinkertaistettu, esimerkiksi HTML-tiedoston alkuun ei tarvitse enää kirjoittaa yhtä paljon kuin edellisissä versioissa. Tämän huomaa esimerkiksi HTML4:een verrattuna, jossa pitää kirjoittaa HTML-tiedostoon seuraava:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"#

"http://www.w3.org/TR/html4/loose.dtd">

HTML5:ssa tämä on yksinkertaistettu seuraavaan koodiin: *<!DOCTYPE html>*. [5]

Uusia elementtien ominaisuuksia on tullut HTML5:een lisää. ”*input*”-elementtiin on tullut paljon ominaisuuksia lisää, jotka monipuolistavat lomakkeita. Näitä ominaisuuksia ”*input*”-elementille ovat muun muassa *tel*, *url*, *email*, *date*, *month*, *week*, *number*, *range* ja *color*.

Esimerkiksi ”*url*”-ominaisuus antaa mahdollisuuden toteuttaa www-sivun osoitetta kysyvän lomakkeen. Käyttäjälle annetaan virheilmoitus, mikäli annettu arvo eroaa www-osoitteesta. [12] Kyseiset elementtien ominaisuudet tekevät myös www-sivujen lomakkeiden täytöstä helpompaa matkapuhelimilla. Tällöin ”*url*”-ominaisuuden kohdalla käyttäjälle annetaan erilliset ”/”, ”.” ja ”.com”-merkit, joiden avulla täyttäminen onnistuu paremmin. [13]

HTML5:n avulla ohjelmoijat voivat tehdä monipuolisempia verkkosivuja ja sovelluksia ilman, että pitää opetella tai lisensoida useampaa teknologiaa. Selaimet voivat tehdä enemmän ilman erikseen asennettavia lisäosia.[1] Lisäksi haittoja selainten lisäosissa on esimerkiksi se, että käyttäjä ei välttämättä pysty asentamaan lisäosaa (kuten julkiset tietokoneet esim. kirjastossa, joissa käyttäjällä on rajoitetut oikeudet). Toiseksi jotkut laitteista eivät tue kaikkia lisäosia kuten mobiililaitteet esim. iPad ja Apple iPhone. Tämän vuoksi lisäosien käyttäminen ei ole suotavaa, etenkään kun pelin halutaan toimivan lisäksi mobiililaitteissa. [14] Erilliset lisäosat voidaan myös estää tai kytkeä pois päältä. Lisäksi lisäosat voivat muodostaa turvallisuus uhan, sekä integrointi muuhun sivuun voi olla vaikeaa.[5]

2.1.3 Tuki

HTML5:sta tuetaan eri selaimissa eri tavalla, koska W3C:n suositukset eivät ole yksiselitteisiä. Internet Explorer ei ole aiemmissa versioissaan kunnolla tukenut HTML5:sta, mutta Internet Explorer 9 (IE9) parantaa tukea. Nykyisin IE9:n käännösmoottori Trident pystyy kääntämään niin HTML5:sta kuin Cascading Style Sheets 3.0:a (CSS). [15] Muissa selaimissa tuki on ollut laaja aiemminkin ja nykyisinkin HTML5:sta tuetaan laajasti.

Keräilykorttipelin asiakasohjelman toteutus on painottunut etenkin käyttöliittymään, kommunikointiin palvelimen kanssa, sekä piirtämiseen. Asiakasohjelman tuen kartoitus rajataan seuraaviin asioihin (joita tullaan käyttämään pelissä): Grafiikka ja sulautetun sisällön tuki, äänen ja videon tuki tarkemmin (mitä eri koodekkeja selaimet tukee), tietokannat ja tiedonvälitys. Mahdollisista lisäominaisuuksista on ”*Drag&drop*”-tuki (myöhemmin

toteutettava), joka on myös kartoitettu.

Näiden osalta Firefox, Opera, Safari- ja Chrome-selaimilla on piirrontuki (”c*anvas-compatible*”) [16]. Internet Explorer 9 tukee myös piirtoa kuten viivan, suorakulmion piirtoa sekä täyttöä, myös kuvien ja videon lisäystä, jotka aiemmin Internet Explorer 8:sta puuttuivat [17]. Taulukossa 1 näytetään grafiikan ja sulautetun sisällön tuen laajuus eri selaimissa. Taulukosta voidaan päätellä, että pelin graafinen osuus voidaan toteuttaa jokaisella selaimella, eikä tuen kanssa ole ongelmia. Keräilykorttipelissä on äänet, joten tarkka äänten ja videon kartoitus on tehty taulukossa 2. Videon tuki on kartoitettu, koska pelissä mahdollisesti toteutetaan välivideoita/ trailereita. Taulukossa 2 huomataan, että mitään koodekkia ei tueta täysin, joten tarvitaan vähintään kaksi eri koodekkia äänille.

Taulukko 1: HTML5 grafiikka ja sulautettu sisältö [16, 17, 18, 19]

	Chrome 14	IE9/IE10	Firefox 8	Opera 11.52	Safari 5.1
Piirtoalue elementti (Canvas Element)	X	X	X	X	X
2D konteksti	X	X	X	X	X
Teksti	X	X	X	X	X
Audio	X	X	X	X	X

Taulukko 2: Äänen ja videon tuki selaimissa [17, 18, 19, 20, 21, 22]

	Chrome	IE9/IE10	Firefox	Opera	Safari
ÄÄNI					
MP3	X	X			X
AAC	X	X			X
Ogg/vorbis	X		X	X	

Wave (wav)	X		X	X	X
VIDEO					
Video (h.264)	X	X			X
MPEG-4		X	X		
Ogg/theora	X		X	X	
WebM	X		X	X	

Taulukossa 3 on listattu tiedon tallentamiseen liittyvät asiat sekä ”*Drag and drop*”-tuki, joka mahdollistaa elementtien siirtelyn. ”*Local Storage*” eli paikallinen muisti, tarkoittaa sitä, että selain tallentaa väliaikaiseen tietokantaan tietoja. Paikallinen muisti on mahdollista toteuttaa evästeiden avulla myös sellaisissa selaimissa, joille ei virallista tukea löydy (esimerkiksi selainten vanhat versiot). Tällöin voidaan käyttää tapaa, joka käyttää evästeitä hyväkseen. Istunto muisti (*session storage*) on hieman samanlainen kuin paikallinen muisti, ainoana eroavaisuutena on, että muisti on sidottu sessioon /istuntoon. Istunto muistia käytetään silloin kun selain päivittyy vahingossa, jolloin estetään käyttäjän tietojen häviäminen [23].

Taulukko 3: Tietokannat [23, 24]

	Chrome	IE9/IE10	Firefox	Opera	Safari
Istunto muisti / Session Storage	X	X	X	X	X
Paikallinen muisti/ Local Storage	X	X	X	X	X
IndexedDB	X	(X=IE10)	X		
Drag and Drop	X	X	X		X

Keräilykorttipelissä on tiedonvälitystä palvelimen kanssa, joten tiedonvälitys on tärkeimpiä teknisistä vaatimuksista pelin kannalta. Pelin toiminnallisuus sijaitsee suurelta osin palvelin puolella ja kommunikointia on paljon. Sen vuoksi nopea ja tehokas kommunikointi on

kriittistä pelin toiminnan kannalta. Taulukossa 4 on kartoitettu HTML5:n tiedonvälitys ominaisuuksia. Näistä WebSocket on tarkoitettu tiedonvälitykseen asiakasohjelman ja palvelimen välillä.

WebSocket toimii TCP:n (Transmission Control Protocol) päällä [25]. Tällä hetkellä Chrome, Firefox ja Safari tukevat WebSockets-ominaisuutta. Opera tukee sitä, mutta se on kytketty pois päältä. Tämä johtuu WebSocketin turvallisuusongelmasta: WebSocket-protokollan suunnittelussa on puutteita, joita hyökkääjät voivat hyödyntää [26, 27]. IE9 ei tue ollenkaan tekniikkaa, mutta IE10 on tarkoitus tukea. Kuten taulukossa 4 näkyy, HTML5 tiedonvälitystekniikoista mikään ei tue viittä yleisintä selainta. Toinen tekniikka, HTML5 ”*Server-Sent*”-ominaisuus antaa mahdollisuuden lähettää tietoa palvelimelta asiakkaalle, mutta asiakas ei voi tämän avulla lähettää tietoa palvelimelle. Tämä ”*Server-Sent*”-ominaisuus ei ole riittävä keräilykorttipeliin, koska tarvitaan tekniikkaa jolla asiakas voi lähettää tietoa palvelimelle, esimerkiksi silloin kun asiakas on pelissä liikkunut, lähetetään palvelimelle tieto liikkumisesta, jotta liike saadaan näkymään toisen pelaajan ruudulla.

Taulukko 4: Tiedonvälitys [22, 25, 26, 28, 29, 30]

	Chrome	IE9/IE10	Firefox	Opera	Safari
Server-Sent	X		X	X	X
WebSockets	X	(X = IE10)	X	(X)	X

2.1.4 Javascript

Javascript on oliopohjainen järjestelmäriippumaton komentosarjakieli, jolla voidaan luoda dynaamisuutta www-sivuille. Javascript suoritetaan selaimessa ja sitä voidaan käyttää käyttöliittymän tehostamiseen sekä antamaan käyttäjälle monipuolisempaa kokemusta vuorovaikutuksesta. [31, 32, 33]

Alun perin JavaScript oli nimeltään Mocha, mutta pian nimi muutettiin LiveScriptiksi ja siitä JavaScriptiksi. Ensimmäinen version kehitti Netscape Communications Corporation, ja se ilmestyi vuonna 1995 samalla kun Netscape Navigator 2.0 ilmestyi. Microsoft kehitti Jscript-kielen, joka on yritetty tehdä mahdollisimman samanlaiseksi kuin JavaScript. JavaScriptistä on tullut monta versiota, joista 1.5 julkaistiin vuonna 2000. Uusin version on 1.8.5. ECMA-262 standardi on suunniteltu JavaScriptin pohjalta. Standardoitu JavaScript on ECMA-262 standardin mukainen. Standardin määrittelee ECMA-järjestö (European standards committee). Standardi helpottaa muiden toteuttajien omien Javascript versioiden kehittämistä [31, 32, 34]

JavaScriptia tuetaan melkein jokaisessa selaimessa, mutta selaimissa voi olla eri JavaScript versiot. Tämä johtuu siitä, että JavaScriptistä ei ole oikeaa ”standardia” (sitä voidaan tulkita eri tavoin) ja jokaisella selaimella on oma JavaScript käännösmoottorinsa. Käännösmoottoreita ovat seuraavat: Internet Explorerissa Chakra, Firefoxissa TraceMonkey, Safarissa Nitro, Chromessa V8 ja Operassa Carackan. [35, 36]

2.1.5 Liitännäisteknologiat

Tässä kappaleessa selvitetään liitännäisteknologiat, jotka liittyvät työhön. Kappaleessa käydään myös läpi tekniikoita, jotka joko tukevat HTML5 ohjelmointia tai jotka olisivat olleet vaihtoehtoisia tekniikoita pelin tai jonkun sen osa-alueen toteutukseen esimerkiksi Flash, CGI (Common Gateway Interface) ja PHP (Hypertext Preprocessor). Tekniikoiden esittelyn lisäksi kunkin hyvät ja huonot puolet kartoitetaan.

2.1.5.1 Cascading Style Sheets

Cascading Style Sheets:n (CSS) avulla voi määritellä kuinka piirto (”*render*”) sivustolla tulee näyttämään. Muunnoksella tarkoitetaan HTML-dokumentin kääntämistä eri laitteille esimerkiksi kääntämistä näytölle. CSS:n voi määritellä, joko suoraan sivulle, tai omaan tiedostoon, joka liitetään sivustolle. [37] CSS on yksinkertainen tyyliohje mekaniikka.

Teksti elementtejä voidaan muokata esimerkiksi seuraavalla tavalla:

H1 { color: blue }

Tämä muokkaa H1-elementtejä siniseksi. Yllä oleva koodi on yksinkertainen CSS-sääntö, joka koostuu kahdesta osasta valitsijasta (*selector*) sekä julistuksesta (*declaration*). Valitsijalla tarkoitetaan elementtiä, johon tyyli kohdistuu. Julistuksella puolestaan ominaisuutta. Valitsija toimii niin sanottuna linkkinä HTML-sivun ja tyyliohjeen välillä. [38]

CSS-tyylitiedoston määrittämää tyyliä voi käyttää jokaisella sivulla. Tämä säästää paljon aikaa etenkin kun on kyseessä iso sivusto. Kun jotain tyyliä pitää muuttaa pitää vain yhtä tiedostoa muuttaa (CSS-tiedostoa). Ilman CSS:ä pitäisi jokaista sivua muuttaa erikseen. Tämän lisäksi CSS:n avulla saadaan sivusto toimimaan nopeammin (siirtonopeus nopeampi). Tämä johtuu siitä, että tyylitiedosto on eri paikassa kuin itse HTML-sivulla ohjelmoituna. Tällöin tyylejä ei tarvitse ladata kuin kerran.

2.1.5.2 Flash

Aiemmin verkkosivuilla on käytetty muun muassa Flash-tekniikka, jonka ensimmäinen versio julkaistiin 1996. Flashiä käytettiin aluksi pelkkään animaatioon www-sivuille, mutta Flashin toinen ja kolmas versio toi perus skriptauksen, joka mahdollisti luoda enemmän dynaamisuutta www-sivuille. Neljäs versio toi vieläkin kehittyneempi ja dynaamisuutta oli kehitetty pidemmälle. Tämä mahdollisti paikan ja asetusten muuttamisen graafisilla elementeillä. Flash-tekniikalla voidaan tehdä laaja kirjo erilaisia WWW-sivuja, vieraskirjoista aina kokonaisiin e-kauppoihin [39]. Tekniikan avulla voi lisätä vektorigrafiikkaa, mp3-ääntä, musiikkia, kuvia, tekstiä, mutta se tarvitsee erillisen lisäosan toimiakseen selaimissa. [40]

Adobe (Flashin kehittäjä) on ilmoittanut lopettavansa Flash-tuen mobiililaitteille. Tämä johtuu HTML5-tuesta, joka on laaja etenkin mobiilipuolella. Adobe siirtyy kehittämään muiden mukana HTML5-tekniikkaa. [41]

2.1.5.3 PHP

PHP (Hypertext Preprocessor) on dynaaminen verkkosivujen luontiin tarkoitettu ohjelmointikieli. Se on tarkoitettu etenkin palvelinpuolen ohjelmointiin. PHP-sivu sisältää HTML-koodia, jossa lisäksi on lisätty PHP-koodi. Koodi aloitetaan erillisillä aloitus- ja lopetusmerkeillä, jotka ovat ”<*?php*” ja ”*?*>”. PHP-koodi suoritetaan palvelimella, josta generoitu HTML-koodi lähetetään asiakasohjelmalle. Asiakas saa skriptin tulokset näkyviin, mutta ei saa selville minkälainen perustana oleva skripti oli kyseessä. Palvelinta voidaan konfiguroida niin, että kaikki HTML-tiedostot prosessoidaan PHP:n avulla, jolloin asiakas ei saa tietää mitään alla olevista koodeista. [42] PHP:llä voidaan tehdä samoja asioita kuin CGI:llä (Common Gateway Interface), kuten kerätä tietoa, tehdä dynaamista sisältöä, lähettää ja vastaanottaa evästeitä. CGI.stä poiketen PHP:tä voidaan ohjelmoida itse www-sivuihin. Sillä voidaan myös ohjelmoida omia työpöytäsovelluksia, jotka ovat järjestelmäriippumattomia. [43] PHP:n huonoina puolina voidaan pitää palvelimen koodin suoritusta. Koodin ollessa monimutkaista tai asiakkaita paljon, vaatii sivuston päivitys aina tiedonvaihdon palvelimen kanssa.

2.1.5.4 CGI

CGI (Common Gateway Interface) on metodi, jolla vaihdetaan tietoja palvelimen sekä selaimen välillä. Tällä metodilla voidaan luoda dynaamisia sivuja. CGI on rajapinta, jonka avulla verkkosivut voivat kommunikoida palvelimen kanssa [44, 45] Tämä rajapinta toimii jokaisessa selaimessa, koska CGI-skriptit eivät toimi selaimessa vaan palvelimella. Huonona puolena CGI:ssä on se, että se voi viedä paljon palvelimen tehoja.[46]

2.2 Selainten kehitystyökalut

Selainten kehitystyökalut liittyy diplomityöhön, koska tällöin koodia voidaan helposti testata. Selaimissa löytyy valmiina työkaluja HTML-sivujen testaamiseen. Chromessa kehitystyökalun saa auki Ctrl-Shift-I komennolla. Internet Explorerissa kehitystyökalu avautuu F12-näppäimellä (Asetuksista ”Developer tools”). Operassa saa auki konsolin, jossa sivun virheet näkyvät (Crtl-Shift-O tai valitsemalla Page- Developer tools - Error console). Firefox-selaimen kehitystyökalut löytyvät WWW-kehitystyökalut valikosta ja tämän lisäksi on olemassa lisäosia, jotka antavat lisää toiminnallisuuksia. Näistä lisäosista Firebug on yksi vaihtoehto (jota muun muassa käytettiin testattaessa pelin prototyyppiä sekä pakkaeditorin prototyyppiä). Safarissa saa tarvittavat työkalut päälle ”Asetuksista”, josta laittamalla ”Show developer menu in the menu bar” saa tarvittavat komennot käyttöönsä (Ctrl-Alt-I ja Ctrl-Alt-C komennot). Kaikissa työkaluissa on konsoli, joka näyttää sivussa olevat virheet. Virheiden lisäksi työkalut näyttävät esimerkiksi tietoliikenteen, jonka sivu ottaa vastaan ja lähettää sekä viestien sisällön.

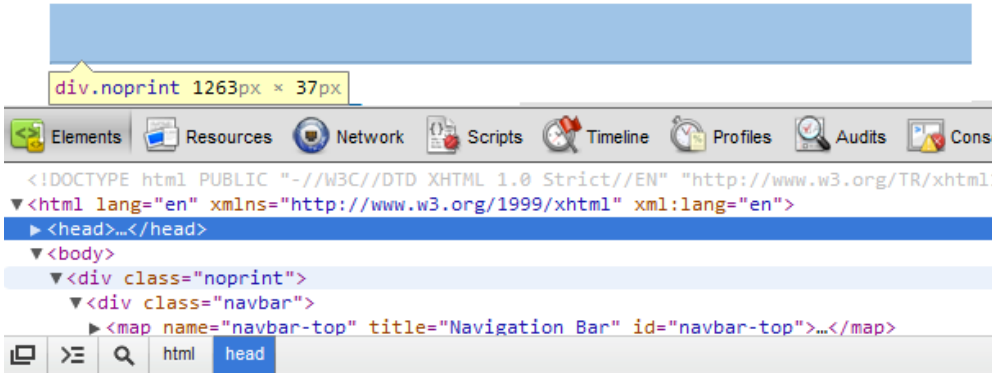
Taulukossa 5 on kartoitettu selainten kehitystyökalujen eroavaisuuksia. Monipuolisimmat työkalut löytyvät Chromesta ja Operasta. Firefoxilla, sekä Internet Explorerissa ovat huonoimmat kehitystyökalut. Firefoxin oma kehitystyökalu on hyvin minimaalinen verrattuna muihin selaimiin, sekä Firefoxiin saatavaan lisäosaan Firebugiin. Firebugin avulla Firefoxiin saa suurelta osin samat toiminnallisuudet kehitystyökalujen suhteen kuin muissakin selaimissa.

Taulukko 5: Selainten kehitystyökalut

	Chrome	Firefox (firebug)	Firefox	Opera	Safari	IE9
Elementit	x	x		x	x	(x)
Tietoliikenne	x	x	x	x	x	x
Skriptit	x	x		x	x	x
Konsoli virheet	x	x	x	x	x	x
Tyylit	x	x		x		
Resurssit	x			x	x	

Tallennus	x	x		x	x	
Profiilit	x	x			x	x

Internet Explorerissa oleva ”Elementtien näyttäminen” eroaa toisista hieman enemmän (taulukossa ominaisuus on suluissa). Muissa selaimissa elementit näytetään sivustolla selvästi maalaten elementti sivustolla, kun taas Explorerissa elementtejä ei näytetä sivulla. Alla kuva elementtien näyttämisestä Chrome-selaimessa (kuva 1).



Kuva 1: Elementtien näyttäminen

Tietoliikenne-termillä tarkoitetaan pystyykö kehitystyökalu seuraamaan tietoliikennettä ja sen sisältöä. Käytännössä tämä tarkoittaa vastaanotettujen HTML-sivujen sisältöjen kyselyitä, vastauksia ja aikaleimoja. Kaikissa selaimissa on tämä ominaisuus. Skripti osuus selaimista näyttää sivuston ohjelmointiosuuden, sekä antaa mahdollisuuden laittaa tiettyjä pysäytyspisteitä, johon ohjelma pysäytetään (debugger-ominaisuus). Tämä skripti-ominaisuus puuttuu ainoastaan Firefox-selaimelta.

Virheilmoitukset (Konsoli-virheet) näytetään jokaisessa selaimessa, mikäli sivustolla on virheitä koodissa. *Tyylit*-kohta tarkoittaa verkkosivun CSS-tyylin näyttöä selaimessa, sekä mahdollisesti värien näyttöä esimerkiksi koodin perusteella jolloin *#000000*-koodi näyttää mustaa erillisessä apuikkunassa.

Resursseilla tarkoitetaan kuvia ja tiedostoja, joita sivusto käyttää. Tällöin kehitystyökalut

näyttävät tiedostot, sekä mahdollisesti vaaditun polun tiedostoihin. Samassa näytetään myös tietokannat, evästeet ja muut mahdolliset tiedostot, joita sivusto käyttää. Tässä osiossa oli suurin eroavaisuus selainten suhteen: vain Chrome, Opera ja Safari tukevat tätä ominaisuutta.

Chrome-selaimen profiilitoiminto kertoo kuinka kauan suoritin aikaa Javascript-funktiot vievät verkkosivulla. Profiilit toimivat samoin Safarissa ja IE:ssä. Näissä profilointi aloitetaan ja lopetetaan itse. Jokainen selain näyttää sitten profiloinnin tulokset, mitä funktiota milloinkin kutsuttiin ja kuinka kauan mikäkin funktion suorittaminen kesti.

2.3 Kommunikaatioteknologiat

Alla on kartoitettu kirjallisuudessa esitettyjä vaihtoehtoehtoisia kommunikaatiotekniikoita, joita voidaan käyttää keräilykorttipelissä.

2.3.1 Teknologiat

Vaihtoehtoisia tekniikoita tiedonvälitykseen on kartoitettu taulukossa 6. Näistä Socket.io toimii jokaisella tutkitulla selaimella, myös vanhemmissa versioissa kuten Internet Explorer 5.5:sta, Firefox 3.sta, Safari 3:sta, Chrome 4:stä ja Opera 10.61:sta myöhemmät versiot. Tukea on myös mobiili puolella kuten iPhone, iPad, Android ja WebOs [47]. Toiminnallisuudeltaan Socket.io on hyvin samantyyppinen kuin WebSockets. Tämä tekniikka tukee kuutta eri siirtotekniikkaa, joista selain valitsee mitä käyttää (käyttäen hyväksi ”*feature detection*”-ominaisuutta selaimissa). Tämä tarkoittaa sitä, että jos käytetään Chrome-selainta, tällöin HTML5 käytetään WebSocket-ominaisuutta Socket.io valitsee selaimen mukaan parhaan siirtotekniikan ja käyttää sitä siirtämiseen. Socket.io käyttää palvelimessa Node.js:ä. [48, 49]

jWebSockets tukee melkein jokaista selainta, mutta Internet Explorerin tapauksessa tekniikkana joudutaan käyttämään FlashBridgeä. Tutkimuksen [51] mukaan tekniikka on todettu hyväksi pienille käyttäjäryhmille. Tutkimuksessa testattiin tekniikkaa käytännössä. Testauksessa oli monta samanaikaista käyttäjää interaktiivisessa pelissä, jolloin palvelimelle

meni kaikilta käyttäjiltä liikennettä. Liikkeet, joita pelissä tehtiin, päivittyi ongelmitta toisille. Isomman käyttäjäryhmän testaamista tutkimuksessa ei ollut. jWebSocketin pääpainona on ollut suorituskyky, sekä vähäinen muistin käyttäminen. jWebSocket tukee myös HTML5 WebSocket-tekniikkaa. [50, 51]

XMLHttpRequest (XHR) ei tue jokaista selainta suoraan, vaan tarvitsee selainkohtaista räätälöintiä. Vanhoissa IE-selaimissa (IE5 ja IE6) tämän sijalla käytetään ActiveX-objektia [52]. Tämä tekniikka kuormittaa verkkoa enemmän kuin WebSocket, koska HTTP-otsikko tietoja lähetetään paljon.[53] Lisäksi asiakaspuolella on olemassa Ajax joka on erityisen suosittu tekniikka tiedonsiirtoon. Google käyttää Ajax-tekniikkaa tiedonsiirtoon muun muassa Gmail, Google Maps-sovelluksissa [54]. Toiminnallisesti XHR- ja Ajax-tekniikka toimivat samalla tavalla: palvelimelle lähetetään kysely johon palvelin vastaa. XHR ja Ajax ovat pelkästään tiedonsiirtotekniikoita, eivätkä liitännäisiä tekniikoita kuten Socket.io. Tämän vuoksi ne voivat käyttää vain yhtä tekniikkaa tiedonsiirrossa toisin kuin Socket.io.

Faye käyttää palvelinpuolella Socket.io-tekniikan tavoin Node.js:ää. Vaihtoehtoisesti palvelinpuolella on käytettävissä myös ruby. Tiedonsiirtoon Faye käyttää WebSocketteja, HTTP Post ja JSONP-tekniikoita [55]

Comet on tekniikka, jota käytetään niin sanottuun ”*Server-sen*t”-ominaisuuteen (joka on jo HTML5-tekniikassa käytössä). Tällöin palvelin lähettää asiakasohjelmaan tietoa ilman, että asiakasohjelma on erityisesti kysellyt sitä. Tekniikka käyttää myös XHR:ä, sekä Ajax-tekniikkaa tiedonvälitykseen. Tällöin se toimii jokaisessa selaimessa, jossa toimii XHR sekä kuormitus on sama kuin XHR ja Ajax-tekniikoilla. Näiden lisäksi Comet voi kommunikoida myös WebSocket-tekniikalla. [56]

Taulukossa 6 on tiedonvälitystekniikat, joissa kuormitus on arvioitu kirjallisuudesta saatavien tietojen perusteella. Tekniikka, joka käyttää WebSocketia on nopein. Kuormituksella tarkoitetaan tiedonsiirtokaistavaatimuksia. Socket.io ja jWebSocket käyttävät WebSockettia,

joissa lähetetään vähemmän otsikkotietoja. Tällöin niiden kuormitus on alhaisempaa kuin yleisimmillä tekniikoilla. Comet-tekniikka ei kirjallisuuden [14] mukaan käytä HTML5:n WebSocketia joten kuormitus on suurempaa. Faye käyttää myös WebSockettia, mutta Socket.io:n verrattuna vaihtoehto tekniikoita on vähemmän [48, 55] joten tekniikkana Faye on luultavasti hitaampaa kuin Socket.io, mutta ei huomattavasti. XHR ja Ajax-kyselyt on laitettu taulukossa normaaliksi vertailukohteiksi toisiin tekniikoihin nähden.

Taulukko 6: Tiedonvälityksen tekniikat

	Selainten tuki	Kuormitus
Socket.io	IE, Firefox, Chrome, Safari , Opera [47]	Pieni [48]
jWebSocket	IE, Firefox, Chrome, Safari , Opera [57]	Pieni [51]
Faye	IE, Firefox, Chrome, Safari , Opera [55]	Pieni
XMLHttpRequest (XHR)	IE, Firefox, Chrome, Safari , Opera (Selainkohtaista räätälöintiä) [53]	Kohtalainen
Ajax	IE, Firefox, Chrome, Safari , Opera [58]	Kohtalainen
Comet	IE, Firefox, Chrome, Safari, Opera [56]	Kohtalainen

2.3.2 Aiempi tutkimus

HTML5 tuo mukanaan lukuisia uusia tekniikoita. Aiempaa tutkimusta näistä on tehty erityisesti kommunikaatioprotokolliin, kuten WebSocket-tekniikkaan liittyen. WebSocket-ominaisuutta käy läpi muun muassa Gutwin et al. [14]. He testaavat ja vertaavat WebSocket-tekniikkaa toisiin tekniikoihin, kuormituksen osalta. Kyseisessä tutkimuksessa käytetään vain Firefox-selainta, eikä muita selaimia ole tutkimuksessa otettu huomioon. Lisäksi tutkimuksessa ei ole kartoitettu ollenkaan tukea HTML5-tekniikan osalta eri selaimissa. Gutwin et al. ohjelmoivat pelejä tutkimuksessaan testatakseen tekniikoita. He tekivät palapelin sekä piirustusohjelman, joita he testasivat kolmella ja neljällä käyttäjällä yhtaikaa. Tutkimus on kuitenkin laaja ja tekniikkaa on tutkittu WAN (Wide Area Network), LAN (Local Area Network) ja MAN (Metropolitan Area Network) verkoissa. Tutkimuksessa todetaan, että WebSocket-tekniikka soveltuu hyvin jokaiseen mainittuun verkkoon. Comet-tekniikalla viive on suurempi kuin WebSocket-tekniikalla. Heidän testauksessaan kävi ilmi, että jokaisessa verkossa Comet-tekniikka oli hitaampi kuin WebSocket-tekniikka.

Bijin Chen ja Zhiqi Xu testaavat tutkimuksessaan [51] kuinka hyvin peli voi toimia selaimessa ja kuinka suuri sen käyttäjämäärä voisi olla. Kyseisessä tutkimuksessa keskityttiin yhteen tiedonvälitystekniikkaan nimeltä jWebSocket, joka käyttää HTML5:n WebSocketia. Tässäkään tutkimuksessa ei otettu kantaa eri selainten HTML5 tukeen. Julkaisussa keskitytään palvelimen kuormitukseen, sekä käyttäjämäärän arvioimiseen vasteaikojen perusteella. Kirjoittajat toteavat, että jWebsocket toimii hyvin etenkin pienillä käyttäjäryhmillä, mutta skaalautuvuus isommille käyttäjämäärille jää tutkimuksen ulkopuolelle.

Marco Rosario kirjoittaa kirjassaan [25] HTML5-ominaisuuksista. Kirjassa käydään WebSocketin tuetut selaimet läpi, sekä kuinka ohjelmoidaan WebSocket-yhteys. Tämän lisäksi hän käy läpi HTML5 piirtämistä. Piirtoalue-kappaleessaan hän kirjoittaa kuinka saadaan selville tukeeko selain piirtoalue elementtejä käyttäen HTML5:sta. Tämä saadaan

aikaa seuraavalla koodilla:

```
//seuraava rivi hakee piirtoalue (canvas) elementin:
var canvas=document.getElementById('canvas');
//mikäli elementtillä ”canvas” ei ole funktioita niin selain ei tue piirtoaluetta:
if(!canvas.getContext){
}
```

Tämän lisäksi Rosario selvittää kuinka ohjelmoidaan erilaisia asioita piirtoalueelle kuten varjoja. Kirjassa käydään piirtoalueissa kaikki tarvittavat: piirto, kääntö, skaalaus, varjot ym. Hän kirjoittaa kuinka piirto kannattaa tehdä, kun halutaan piirtää useita elementtejä piirtoalueelle yhtä aikaa. Tällöin hän ehdottaa, että piirtoalueita on monta, jotka sijaitsevat eri syvyyksissä, jolloin vain tarvittavat piirtoalueet piirretään uudestaan. Hän esittää myös tarkemmin vinkkejä kuinka saada animaatio toimimaan hyvin mobiililaitteissa. Kätevä vinkki on piirtoalueen leveyden ja korkeuden resetointi, joka suoraan puhdistaa piirtoalueen. Näiden lisäksi hän kirjoittaa, että sujuvan animaation saa myös laittamalla kaksi piirtoaluetta samaan kohtaan, ja näyttäen toisen piirtoalueen, eli niin sanotun puskuri piirtoalueen (”*buffer*”), sen jälkeen kun piirto on valmis.

Yllä olevista tutkimuksissa selviää, että WebSocket soveltuu hyvin verkkopeleihin ja kuormitus on vähäistä. Niissä ei kuitenkaan käydä läpi tarkemmin WebSocket-tukea eri selaimissa. Lisäksi näissä tutkimuksissa ei ole tarkempaa tietoa HTML5-ominaisuuksien tukemisesta. Rosario on käsitellyt kirjassaan [25] laajasti HTML5-ominaisuudet, sekä antanut hyviä tietoja kuinka voidaan tehdä sujuvaa animaatiota tavallisilla menetelmillä. Hän ei ole kuitenkaan tarkemmin selvittänyt eri selainversioiden HTML5-tukea, vaan käy yleisesti HTML5-tekniikan läpi.

3 Tutkimuskohteen ongelmat

Tutkimuskohteena on keräilykorttipelin asiakasohjelma, joka toimii selaimella PHP:n, Javascriptin sekä jQueryn avulla. Koska WebSocket-tekniikka ei toimi jokaisessa selaimessa, keräilykorttipeli tarvitsee toisen kommunikaatiotekniikan. ”Kommunikaatiotekniikat”-kappaleessa testataan olemassa olevia tekniikoita, sekä verrataan niitä muun muassa Socket.io-tekniikkaan. Muissa kappaleissa tutkitaan keräilykorttipelissä ilmenneet ongelmat ja ratkaistaan ne. Ongelmia esiintyi satunnaislukugeneraattorin ja jQuery *bind*()-funktion kanssa. Animaatio on tärkeä osa peliä, jonka vuoksi pelin animaatio tutkitaan ja tarkistetaan onko se tehokas, jotta käyttäjälle tulee hyvä pelikokemus. Lisäksi ”*drag&drop*”-tuki selvitetään, koska se on myös pelille oleellista. Sitä esimerkiksi käytetään pakkaeditorissa.

3.1 Kommunikaatiotekniikat

Kommunikaatiotekniikoista otetaan huomioon sellaiset tekniikat, jotka on tarkoitettu asiakasohjelman ja palvelimen välillä toimiviksi. Kommunikaation pitää tapahtua molempiin suuntiin eikä pelkästään palvelimelta asiakasohjelmaan kuten *”Server-sent”*-ominaisuus HTML5-tekniikassa. Peliin tarvitaan nopea, sekä tehokas kommunikaatiotekniikka. Nopeudella estetään käyttäjälle näkyvät pelikokemusta heikentävät viiveet. Pelin pitää skaalautua suurelle käyttäjämäärälle, joten on erittäin tärkeää valita teknologia, joka ei aiheuta turhaa kuormitusta palvelimelle tai tarpeetonta verkkoliikennettä. Lisäksi HTML5:ssa oleva WebSocket-tekniikka ei toimi kuin Chrome-, Safari- ja Firefox-selaimissa ja osassa tämä ominaisuus on poistettu käytöstä (Opera). WebSocket-tekniikalle tarvitaan korvaavan tekniikka jonka vaatimuksina olisi, että tekniikka olisi nopea, tekniikan pitää toimia jokaisella selaimella, sekä olisi mahdollisimman kevyt (tietoliikenteen kannalta liikennettä olisi vähän).

Kirjallisuuden perusteella parhaiten pelin kommunikointitekniikaksi sopii Socket.io ”*feature detection*”-ominaisuuden vuoksi. Lisäksi tekniikassa on pieni kuormitus ja selainten tuki on laaja, sekä mobiilipuolella tukea on laajalti. Toinen vaihtoehto olisi myös jWebSocket, joka myös käyttää HTML5 WebSockettia, mutta ei ole dokumentaatiota toimiiko se esimerkiksi iPadissa. Lisäksi Internet Explorerissa tarvitaan kyseiseen tekniikkaan Adobe Flash Player, jonka avulla (FlashSocket) tiedonvälitys tapahtuu. Socket.io on sopivampi, koska tarvitaan

teknologia, joka mahdollistaa mahdollisimman tehokkaan tiedonvälityksen. Tämä lisäksi toimii myös iPadissa. Kirjallisuudesta tekniikoista ei löytynyt kunnollista vertailua nopeuden osalta, joten tarvitaan lisäselvitystä (testausta), sekä vertailua tekniikkojen kesken etenkin yleisesti käytettäviin Ajax- ja XmlHttpRequest-tekniikoihin.

3.1.1 Kommunikaatiotekniikoiden testaus

Kommunikaatiotekniikoiden testauksiin otettiin Taulukon 6 mukaan paras, sekä nopein tekniikka eli Socket.io. JWebSocket-tekniikka käyttää WebSockettia, mutta esimerkiksi Internet Explorerin osalta FlashBridgeä, joka on hidas teknologia. Tämän vuoksi jWebSockettia ei oteta mukaan vertailuun. Palvelinpuolella Faye:ssa sekä Socket.io:ssa toimii Node.js, joten kummatkin tekniikat ovat melkein samoja. Kummatkin tekniikat käyttävät siirtämiseen Websocket-teknologiaa mikäli mahdollista, joten viiveet, sekä kuormitus on samaa luokkaa. Fayessa eri tiedonsiirtoteknologioita on kolme [59] ja Socket.Io:ssa viisi. Molemmat käyttävät samoja tekniikoita, joten Faye-tekniikkaa ei tarvitse erikseen testata, Socket.io ajaa saman asian. Tämän lisäksi otetaan yleiset kommunikaatiotekniikat mukaan vertailun vuoksi, joita on XMLHttpRequest ja Ajax.

Testauksen jQuery/ajax ja XMLHttpRequest kyselyissä palvelinohjelmistona oli Tomcat 7 ja Java 1.6. Testilaitteistona toimi Windows 7 64-bit, Intel Core i5 M460 @ 2.53 Ghz, 4 Gt RAM. Testiohjelmisto lähetti palvelimelle 1000 pyyntöä palvelimelle, ja palvelin palauttaa ajan mikrosekunteina edellisestä pyynnöstä. Palautetuista ajoista laskettiin keskiarvo. Kertojen määrä on rajattu, jotta testi valmistuu ennen kuin selain tulkitsee ajossa olevan skriptin lukkiutuneeksi. Kaikki testikäytössä olleet selaimet sisälsivät toipumismekanismin, joka pysäytti lukkiutuneeksi oletetun skriptin.

Testisivun lataaminen toistettiin 10 kertaa muutaman sekunnin väliajoin. Lataaminen tehtiin painamalla selaimen Refresh-painiketta. Tällä tavoin pyrittiin tasaamaan käyttöjärjestelmän taustaprosesseista johtuva suorituskyvyn vaihtelu testikertojen välillä.

Ajanmittaus:

Selainten JavaScriptissä ajan saa selvitettyä ”(new Date()).getTime()”-komennolla, tämän menetelmän tarkkuus on millisekunti [60]. Menetelmän todellinen tarkkuus riippuu selaimesta ja käyttöjärjestelmästä ja voi olla huonompi kuin 10ms. GetTime()-funktio lisäksi pyöristetään, joten jos aika on tarpeeksi pieni, metodi palauttaa nolla millisekuntia [61]. Tämän asemasta annetaan palvelimen Java-servlet toteutuksen mitata aika. Toteutukseen käytetään System.nanoTime()-metodia. Java SE 1.6 dokumentaation mukaan metodi palauttaa nanosekunnin tarkkuudella ajan [62]. Windows-alustalla Java käyttää toteutuksessa QueryPerformanceCounter Windows API kutsua [63]. Testilaitteistolla tarkistettu tarkkuus (QueryPerformance Frequency API kutsu): 2467978 Hz. Menetelmällä saadaan pyyntöön kulunut aika mikrosekunnin tarkkuudella. Ajanmittausmenetelmä on sama kaikille selaimille.

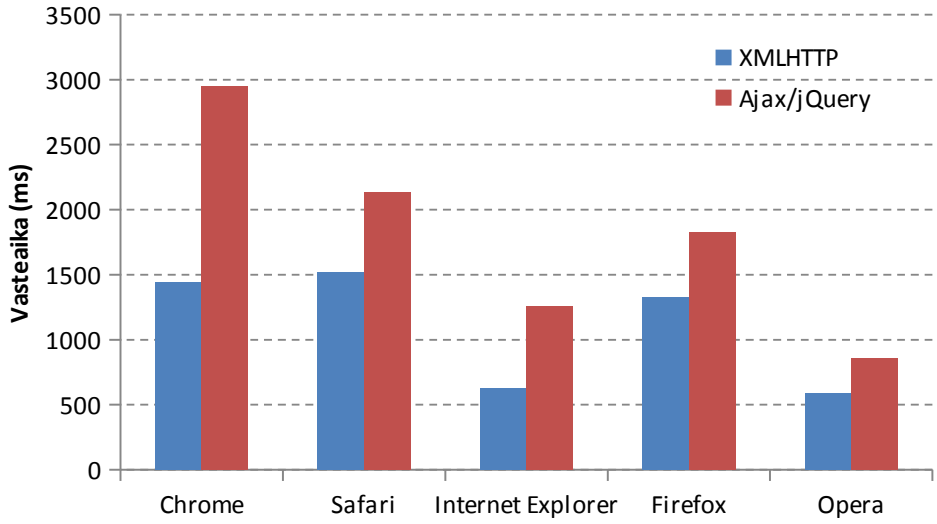
Taulukossa 7 näkyy XMLHttpRequestin ja Ajax/jQueryn mittauksien tulos. Tulokset tässä ovat mikrosekunnin tarkkuudella. Kuvasta 1 (Vasteajat), sekä taulussa 6 voi nähdä, että XMLHttpRequest on jokaisella selaimella nopeampi kuin Ajax-kysely. Testiohjelmaa varten Internet Explorerissa piti ottaa pois kyselyjen/vastausten otsikkojen ”cachettaminen” (tallettaminen välimuistiin), koska tämä vääristi testituloksia. Tulosten vääristys johtui siitä, että Internet Explorer palautti yhden vastauksen arvolla -1. Tämän jälkeen IE laittoi saman arvon muihinkin vastauksiin, jonka takia kyselyt näyttivät nolla-arvoa. Testiohjelmaa tehdessä huomattiin, että ainoana selaimena Firefox päivitti tekstikenttää (johon vastausten ajat tulivat näkyviin kyselyjen välissä). Kentän päivitys poistettiin, jotta tulokset olisivat vertailtavissa keskenään (tämä nopeutti huomattavasti Firefoxin tuloksia). Lisäksi Firefoxissa käytettäessä lisäosia, ne on poistettava käytöstä, jotta testaustulokset olisivat luotettavampia. Etenkin lisäosat (kuten Firebug), jotka seuraavat tietoliikennettä hidastavat liikennettä.

Taulukko 7: XmlHttpRequest (XHR) ja Ajax vertailu (mikrosekunneissa)

	Minimi		Keskiarvo		Maksimi	
	XHR	Ajax	XHR	Ajax	XHR	Ajax
Chrome	1200	1605	1445	2947	5904	322835
Safari	1179	1690	1518	2136	12848	15223

Internet Explorer	527	915	628	1256	10346	32031
Firefox	962	1313	1325	1830	47161	84376
Opera	455	658	583	858	9180	12385

Taulukosta 7 voi päätellä siis, että keskiarvo 1000 XHR-kyselylle vaihtelee selaimesta riippuen välillä 582-1517 mikrosekuntia ja Ajax-kyselylle välillä 857-2947 mikrosekuntia. Tällöin saadaan nopeimmillaan vastausajaksi yhdelle kyselylle 0,582 ms (XHR kysely), sekä Ajax-kyselylle 0,857ms. XHR on nopeampi kuin Ajax-kysely, jokaisessa selaimessa.



Kuva 2: Vasteajat

Kuvassa 2 on vertailu XmlHttpRequest:n ja Ajaxin kesken. Kuvaajasta voidaan päätellä,että XmlHttpRequest on hieman nopeampi jokaisella selaimella. Chromessa Ajaxin vasteajat olivat kaksinkertaisia verrattuna XHR:n vasteaikoihin.

Kyselyistä XHR oli selvästi nopeampi, joten seuraavaksi selvitettiin kuinka paljon menee aikaa Socket.Io:n kyselyihin ja verrataan tuloksia XHR-tekniikkaan. Socket.Io:n testaamisessa tehtiin 1000 kyselyä. Jokaisen kyselyn vasteajat laskettiin erikseen. Tämän jälkeen laskettiin kokonaisaika, joka meni 1000 kyselyyn.

Kun selaimiin tuli kaikki vastaukset, laskettiin selaimen puolella kyselyihin mennyt aika. Tämä tehtiin sen vuoksi, koska palvelinta ei voinut tehdä Java-servlet toteutuksella ja Fiddler-ohjelma (joka on tarkoitettu tietoliikenteen seuraamiseen) ei näyttänyt kaikista selaimista tietoja. Testaukseen laskettiin 10 otoksen keskiarvo, jolloin arvot ovat tarkempia kuin yhden otoksen ja saadaan tasattua käyttöjärjestelmän taustaprosesseista johtuva suorituskyvyn vaihtelua testikertojen välillä. Palvelin puolella Socket.io-ominaisuuksia pitää muuttaa niin, ettei asiakasohjelma (selain) käytä välimuistia hyväksi lisäämällä seuraava rivi Socket.io-palvelimen koodiin: *io.disable('browser client cache');*. Tämän lisäksi asiakasohjelma lähetti kyselyn numeron, jonka palvelin palautti takaisin. Tällä tavoin saadaan varmistettua, että jokainen viesti tulee takaisin.

Taulukossa 8 merkki ”x” tarkoittaa, että kyseinen ominaisuus ei toimi. Kaikki tulokset ovat millisekunteja. ”Normaali”-tekniikka merkinnällä tarkoitetaan palvelinpuolella sitä, että siellä sallitaan jokainen tekniikka. FlashSocket on oletuksena poissa päältä, tällöin Opera ei toimi. Tämä johtuu Socket.io:n ongelmasta jonka vuoksi muut tekniikat (kuten XHR ja jsonp) eivät näytä toimivan Opera-selaimessa [64, 65, 66, 67, 68]. Suluissa olevat tulokset ilmaisevat tuloksia, jotka eivät normaaliasetuksilla näy ilman erillisiä toimenpiteitä, esimerkiksi Operassa pitää laittaa WebSocketin päälle erikseen, jotta tekniikkaa voidaan käyttää. WebSocket-tekniikan päällä ollessa Opera-selaimella kyselyjen vastaukset nopeutuvat huomattavasti.

Taulukko 8: Socket.Io:n tekniikat selaimissa (millisekunneissa)

Tekniikka/Selain	Firefox	Chrome	IE9	Opera	Safari
WebSockets	190	147	x	(348)	202
FlashSocket	x	x	x	757	x
xhr-polling	204	162	285	x	1072
jsonp-polling	381	264	229	x	1073
Normaali	200	155	250	753 (224)	232

Testausten lisäksi on olemassa tutkimus XMLHttpRequest (XHR) ja WebSocket-tekniikoiden nopeuksista eri tietoliikenneverkkojen välillä [14]. Kyseisessä tutkimuksessa verrattiin kahta

tekniikkaa, joista toinen on WebSocket ja toinen XHR (Taulukko 9). Taulukosta nähdään, että WebSocket-tekniikka on paljon nopeampi kuin XHR. Tämä tutkimus vahvistaa myös testauksessa saadut tulokset.

Taulukko 9: Lan ja Wan [14]

	XHR			WebSockets		
	LAN	MAN	WAN	LAN	MAN	WAN
Keskiarvo viive (ms)	67.8	121.2	185.7	11.6	55.8	86.5
Keskiarvo vaihtelu (ms)	13.7	6.6	9.5	8.5	7.1	6.5

Tuloksista havaittiion, että WebSockettia oli nopein tekniikka kuin muut vertailussa olevat tekniikat. Testauksissa kävi myös ilmi, että WebSocket-tekniikkaa tukevat selaimet käyttivät tekniikkaa oletuksena. Socket.io tekniikan avulla yhden kyselyn tekemiseen menee aikaa noin 0,147ms. Ensimmäisen testisetin tuloksista nopein oli XHR (Taulukko 7) jonka kesto oli 0,582ms/ kysely. Nopein kysely oli Opera-selaimessa, muissa kyselyjen kesto oli 0,628ms – 1,518ms / kysely. Socket.io:ssa oleva XHR (Eli XmlHttpRequest) tekniikka antaa samoja tuloksia (0,162ms – 1072ms / kysely). Ero tuloksien välillä johtuu JavaScriptin käytöstä, josta aiemmin oli mainittu. Jotta tuloksista saataisiin mahdollisimman paikkaansa pitävät, ohjelmoitiin myös palvelimeen laskuri. Laskuri laski kyselyiden aloitus ja lopetusajat, jonka jälkeen näiden erotus näytettiin (aika joka meni 1000 kyselyn tekemiseen). Tulokset olivat taulukon 8:n mukaisia.

3.1.2 Yhteenveto

WebSocketin korvaajaksi Socket.io on paras vaihtoehto, koska se on nopea. Tekniikka käyttää WebSocketia mikäli pystyy, jolloin kuormituskin on vähäistä. Selainten tuki on laaja Socket.Io:ssa. Lisäksi sitä on helppo käyttää, sekä siitä pystyy vaihtamaan helposti tiedonsiirtotekniikkaa (muokattavuus). Tekniikka on vahva palvelinpuolella, se antaa paljon eri ominaisuuksia mitä voi tehdä tai muokata palvelin puolella. Lisäksi Socket.io käyttää monipuolisesti eri tekniikoita parhaan mukaan ja on jopa 115% tai 500% nopeampi kuin

Comet tekniikka [14]. Tekniikka ei tue tällä hetkellä XHR-tiedonsiirtoa Opera-selaimessa, mutta Socket.io-tekniikkaa kehitetään koko aika, joten tämä oletettavasti tulee tuetuksi.

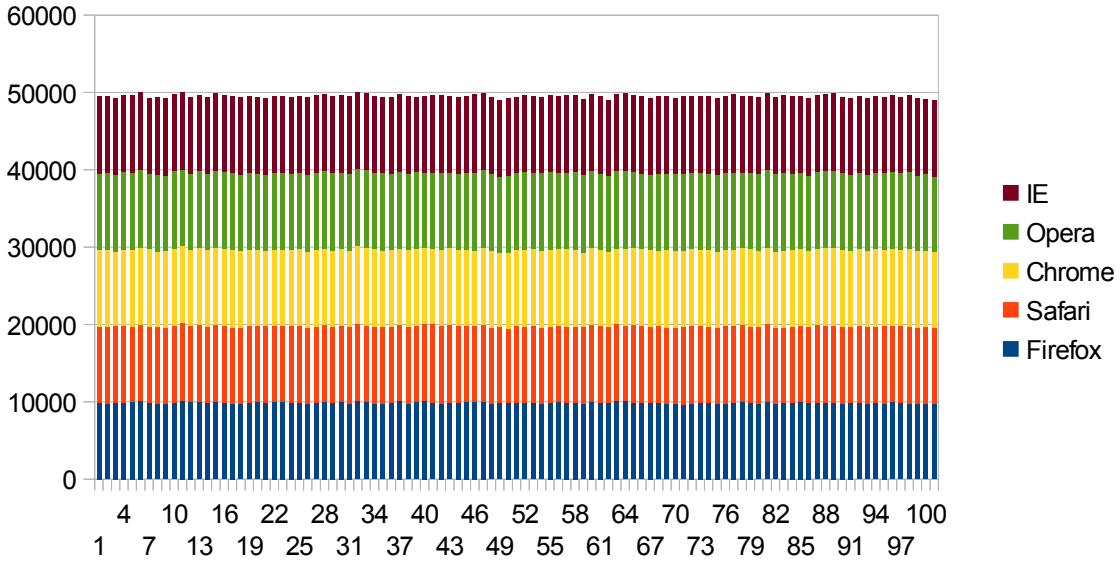
HTML5 tuetaan yhä enemmän ja selaimet päivittyvät nopeaa tahtia. Tulevaisuudessa jokainen selain mukaan lukien IE, sekä Opera tulee tukemaan WebSocketia, jonka avulla keräilykorttipeli tulee toimimaan entistä nopeammin, joten tekniikkana kannattaa käyttää Socket.io:ta.

3.2 Satunnaislukugeneraattori

Keräilykorttipelin prototyypissä käytetään Math.random()-funktiota korttien arpomisessa. Kyseisessä funktiossa on huomattu eroavaisuuksia selainten suhteen, se toimii vain osassa selaimista. Tätä ominaisuutta käytettiin korttien arpomiseen pelaajalle. Korttien arpomisessa huomattiin ongelmia etenkin Firefoxin korttien arpomisessa, jossa samaa korttia tuli monta kertaa peräkkäin kuten Kuvassa 3.

Kuva 3: Korttien järjestys kädessä

Tämän ongelman testaamiseksi tehtiin testiohjelma, jonka avulla testattiin *Random*()-funktion toimivuus jokaisessa selaimessa. Testiohjelma arpoi miljoona numeroa lukujen 0 ja 100:n välillä. Odotusarvo tälle otokselle on 50. Testattaessa jokaista selainta tulokset osoittivat, että kaikkien arvojen keskiarvo oli odotusarvon lähellä. Lisäksi Kuvassa 4 näkyy selainten generoimien arvojen lukumäärä, josta voidaan päätellä, että JavaScriptin *Math.random*()-funktio ei tulosten perusteella vaikuta tuottavan virheellistä satunnaislukujakaumaa. Tulokset jakautuvat normaalisti eli tasajakauman mukaisesti.



Kuva 4: Random tulokset selaimissa

Math.Random()-funktio ei itsessään siis ole ongelman lähde. Koodissa satunnaislukugeneraattoria käytetään seuraavalla tavalla:

```
this.deck.sort(function() {
        return 0.5 - Math.random();
});
```

Math.Random()-funktiossa ei ollut ongelmia testauksissa, joten täytyy selvittää *sort()*-funktion toiminta. Sort-funktio on Javascriptin oma funktio, joka on tarkoitettu taulukoiden järjestämiseen [69, 70]

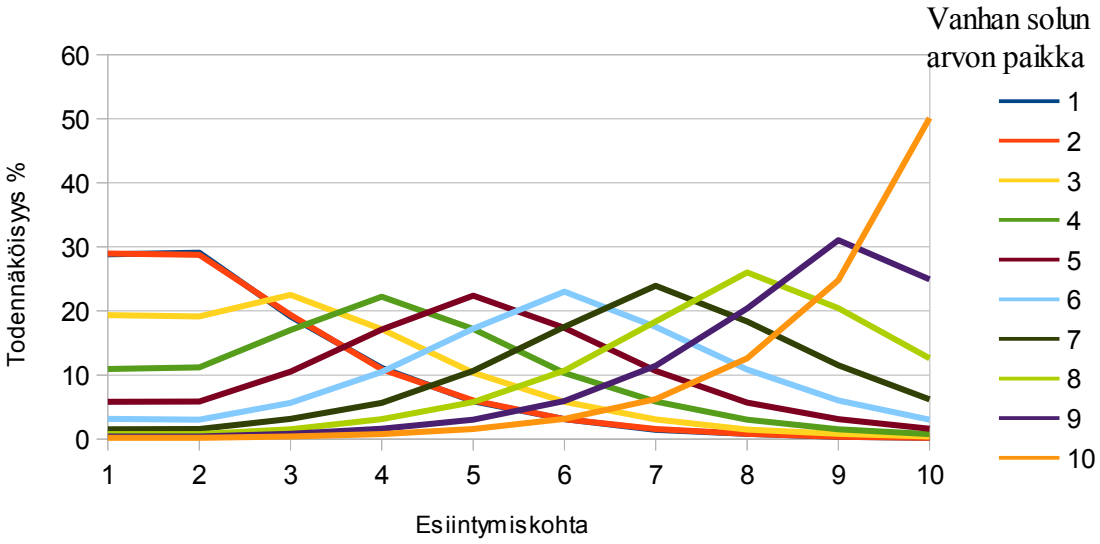
Sort()-funktion avulla arvotaan kymmenen solun taulukon sisältö 100 000 kertaa. Tämän jälkeen laskettiin kuinka monta kertaa tiettyjen taulukon solujen arvot esiintyivät kussakin paikassa. Kunkin taulukon arvojen pitäisi esiintyä noin 10 000 kertaa jokaisessa solussa (10% todennäköisyys), jotta koodi toimisi tasajakauman mukaan. Kyseisessä testissä huomattiin, että tulokset eivät ole tasaisesti jakautuneet (Taulukko 10). Taulukossa 10 tulokset näyttävät prosenttista mahdollisuutta arvojen eri esiintymiskohtiin (solun paikka). Esimerkiksi arvoa

neljä (taulukon neljännen solun arvo) esiintyy (*sort*-funktion jälkeen) ensimmäisessä solussa 10% todennäköisyydellä, toisin kuin kolmannessa solussa esiintymistodennäköisyys on 17%. Testiä testattiin useaan otteeseen ja tulokset olivat samanlaisia. Alla oleva taulukko on otos Chromen tuloksista.

Taulukko 10: Sort -funktion tulokset Chromessa (todennäköisyys %)

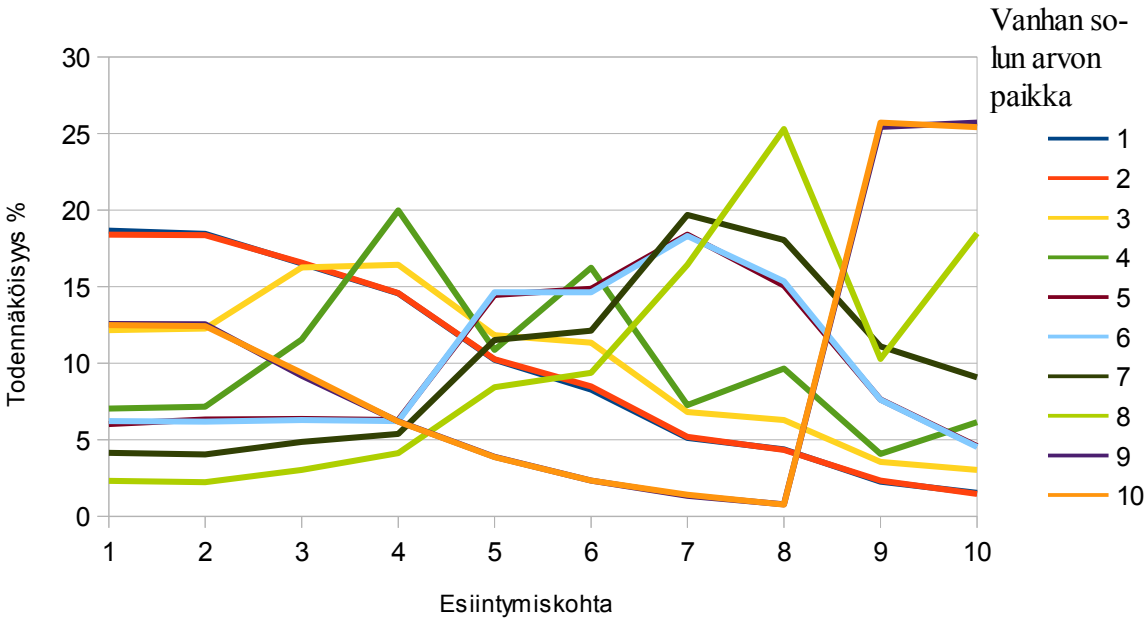
Solun sisällön uusi kohta	1	2	3	4	5	6	7	8	9	10
Alkuperäinen solu 1	28,85	29,099	19,122	11,088	5,892	3,119	1,455	0,821	0,362	0,192
2	28,975	28,737	19,368	10,884	6,005	3,088	1,566	0,819	0,361	0,197
3	19,349	19,149	22,51	17,136	10,352	5,814	3,048	1,469	0,792	0,381
4	10,951	11,185	16,99	22,222	17,163	10,325	5,857	3,014	1,525	0,768
5	5,828	5,844	10,522	17,074	22,364	17,347	10,66	5,674	3,091	1,596
6	3,15	3,046	5,638	10,49	17,23	23,033	17,516	10,83	6,026	3,041
7	1,531	1,552	3,159	5,627	10,644	17,514	23,915	18,367	11,51	6,181
8	0,762	0,789	1,507	3,112	5,751	10,671	18,34	26,031	20,447	12,59
9	0,394	0,411	0,795	1,605	3,043	5,941	11,422	20,4	31,059	24,93
10	0,21	0,188	0,389	0,762	1,556	3,148	6,221	12,575	24,827	50,124

Tulokset noudattivat hyvin paljon seuraavaa kuvaajaa (Kuva 5). Kuvaajasta voidaan selvästi nähdä, että tulokset eivät noudata tasajakaumaa. Taulukon ensimmäisen ja toisen solun arvoja esiintyi *sort()*-funktion käytön jälkeen eniten ensimmäisissä soluissa. Jokaisen solun esiintymistodennäköisyys pieneni, mitä kauempana alkuperäisestä solusta arvo oli. Tasajakaumassa mahdollisuus esiintymiskohtaan pitäisi olla sama kaikissa solujen arvoissa. Kuvaajassa oikealla vanhan taulukon solujen arvot.



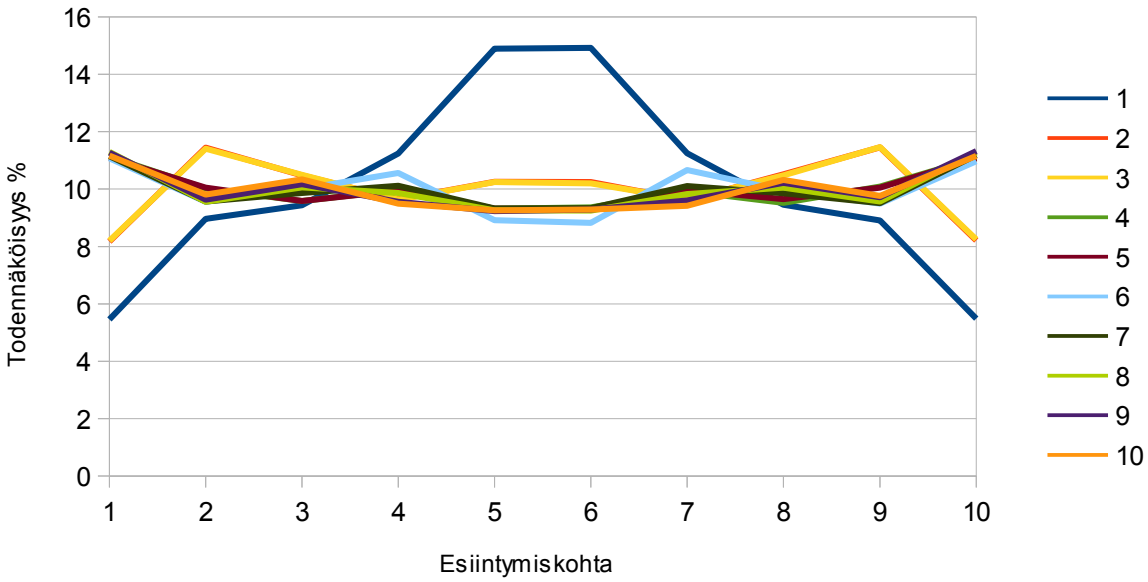
Kuva 5: Sort-funktion tulokset Chromessa

Firefox-selaimella luvut heittelivät pahasti (Kuva 6), jossa esiintymiskohdat vaihtelivat rajusti. Esimerkiksi vanhan taulun kymmenennen ja yhdeksännen solun arvon esiintymistodennäköisyys *sort*-funktion jälkeen oli yhden prosentin luokkaa soluun kahdeksan.



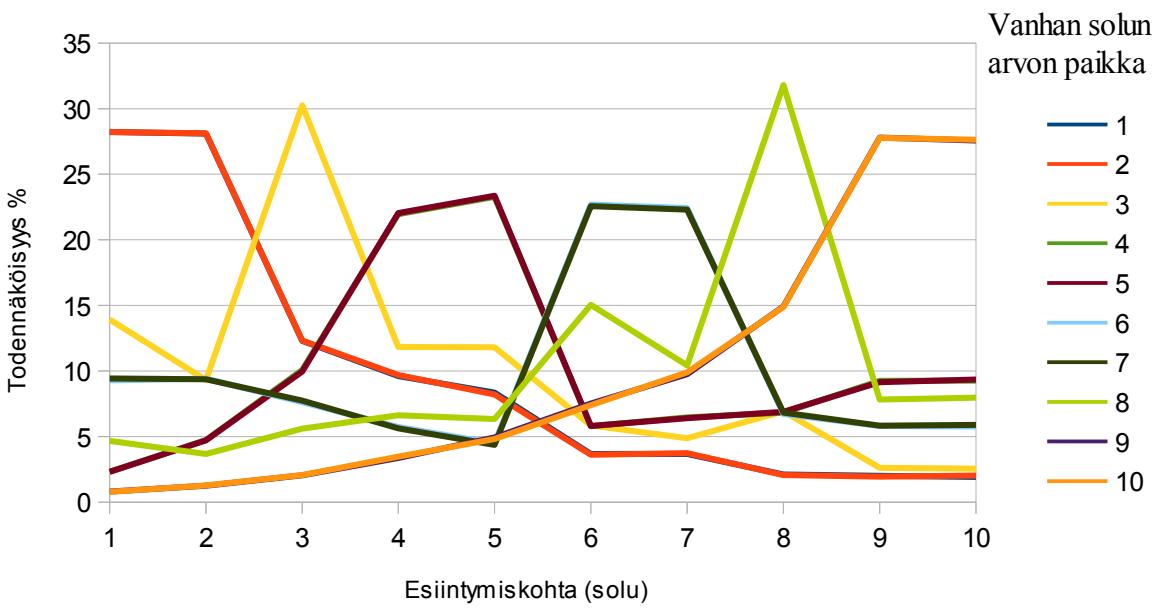
Kuva 6: Firefoxin tulokset Sort-funtion käytöstä

Safarissa tulokset näyttivät aluksi tottelevan tasajakaumaa, mutta tarkemmin tarkastellessa tuloksia ja tuloksista saatavaa kuvaajaa (Kuva 7) huomattiin, että tulokset myös vaihtelivat kuten muissa selaimissa. Tällöin esimerkiksi vanhan taulun solua yksi esiintyy kohdissa viisi ja kuusi suuremmalla todennäköisyydellä kuin kohdissa yksi ja kymmenen. Vanhan taulun solua yksi esiintyy *sort()*-funktion jälkeen kohdissa yksi ja kymmenen noin 5% todennäköisyydellä.



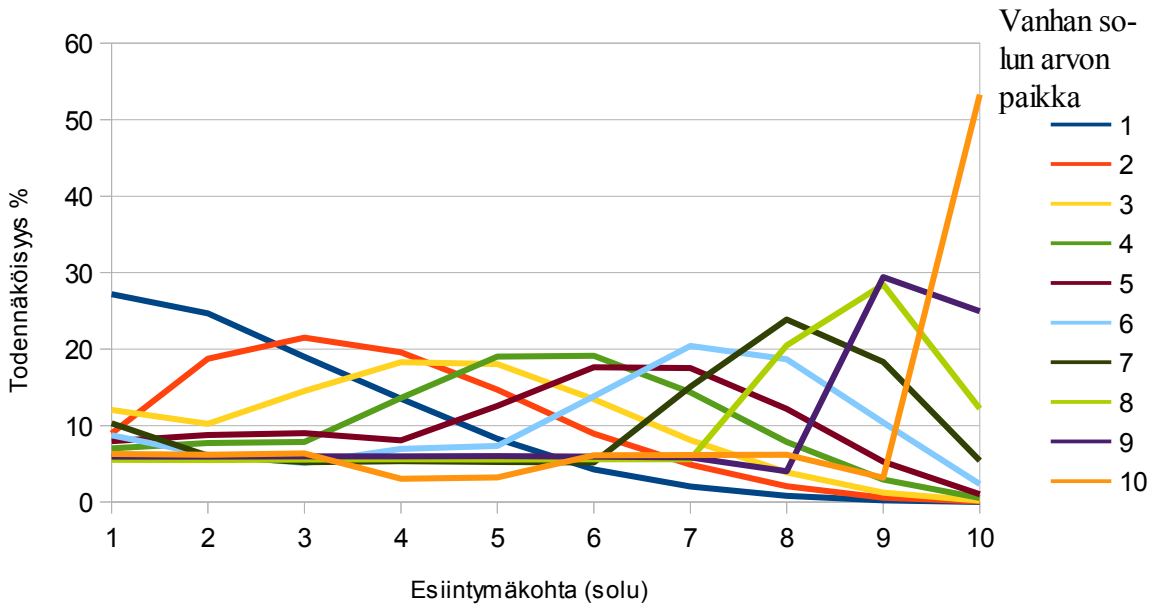
Kuva 7: Safarin tulokset Sort-funktion käytöstä

Operassa tulokset jakautuivat myös erittäin paljon (Kuva 8), joissakin tapauksissa tietyt arvot esiintyivät noin 30% todennäköisyydellä solussa, mutta toisten arvojen esiintyminen oli parin prosentin luokkaa (kymmenen ja yhdeksän solun arvot esiintyivät solussa yksi ja kaksi parin prosentin todennäköisyydellä) .



Kuva 8: Operan tulokset Sort-funktion käytöstä

Internet Exploresissa tulokset myös vaihtelivat, ja kuvassa 9 näemme että solun 10 arvoa tulee paljon useimmiten viimeiseen soluun (mahdollisuus tähän on noin 55%). Kun taas arvoa 1 tulee viimeiseen soluun 0,028% todennäköisyydellä.



Kuva 9: Internet Explorer tulokset Sort-funktion käytöstä

Tuloksista voidaan päätellä, että *sort*()-funktiota ei kannata käyttää taulukon satunnaiseen järjestelyyn. Tulokset eivät ole ollenkaan tasaisesti jakautuneita yhdessäkään selaimessa. Pelin kannalta tarvitaan *sort()*-funktion korvaaja, joka arpoisi taulukon arvot satunnaiseen järjestykseen.

3.2.1 Ratkaisut

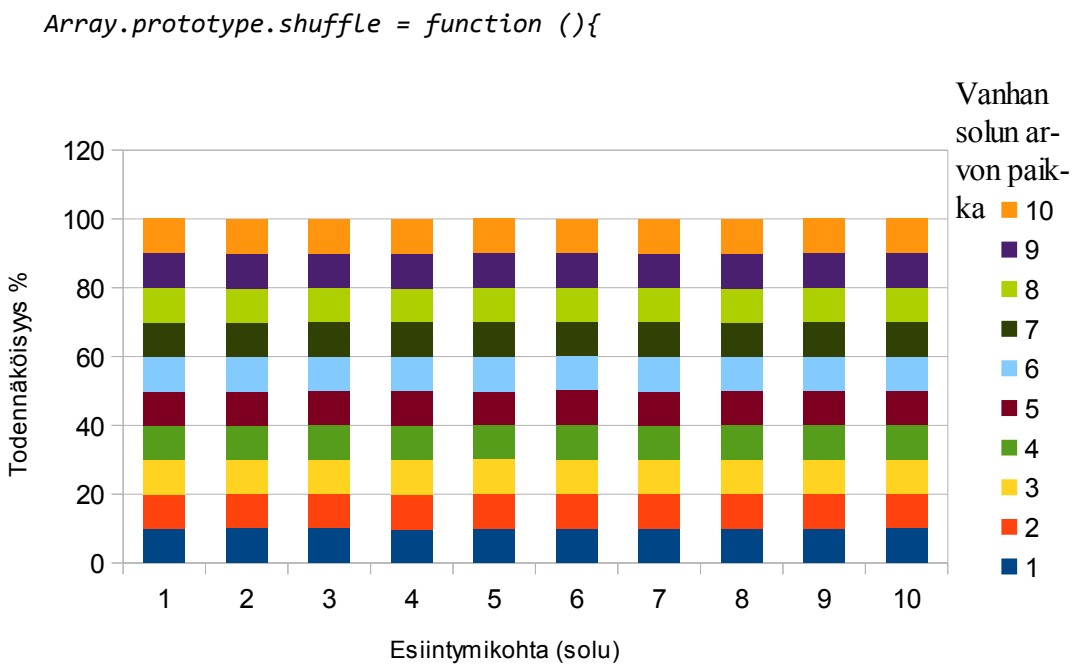
Käytettäessä lähteestä [71] löytyvää koodia hyväksi, voidaan taulukko arpoa paremmin. Kyseistä koodia käyttäen saadaan, jokaisesta selaimesta jokaiseen esiintymiskohtaan liki 10 000 kappaletta. Tulokset ovat tasajakauman mukaisia. Alla on tulokset koodin testauksesta (Taulukko 11). Taulukossa luvut ovat prosentteina.

Taulukko 11. Uuden satunnaislukugeneraattorin testauksen tulokset (% todennäköisyys)

Numero/Esiintymiskohta	1	2	3	4	5	6	7	8	9	10
1	9,985	10,09	10,101	9,82	9,969	9,911	9,931	9,978	10,059	10,156
2	9,907	9,996	10,057	9,907	10,02	10,043	10,095	10,097	9,863	10,015
3	10,025	9,784	9,91	10,223	10,127	10,017	9,974	9,844	10,129	9,967
4	9,957	10,011	10,035	10,086	9,882	10,103	9,828	10,098	9,997	10,003
5	9,932	9,981	10,044	9,971	9,836	10,126	10,118	10,04	9,972	9,98
6	10,05	10,076	9,941	9,94	10,161	10,084	9,938	9,962	9,906	9,942
7	10,103	9,777	9,989	10,033	10,014	9,855	10,161	9,9	10,123	10,045
8	10,05	10,076	10,035	9,869	10,059	9,978	9,892	9,937	10,066	10,038
9	10,052	10,086	9,861	9,987	10,017	10,025	10,093	10,013	9,971	9,895
10	9,939	10,123	10,027	10,164	9,915	9,858	9,97	10,131	9,914	9,959

Kuvassa 10 näkyy, että kuvaajat ovat samanlaisia ja todennäköisyydet ovat kullakin numerolla noin 10%.

Keräilykorttipelissä oleva koodi pitää korvata seuraavalla koodilla, joka on Fisher-Yates algoritmin mukainen, tällöin saadaan tuloksiin tasainen jakauma [71].



Kuva 10: Uuden funktion tulokset

```
Array.prototype.shuffle = function (){
        var i = this.length, j, temp;
        if ( i == 0 ) return this;
        while ( --i ) {
                j = Math.floor( Math.random() * ( i + 1 ) );
                temp = this[i];
                this[i] = this[j];
                this[j] = temp;
        }
        return this;
};
```

Koodin jälkeen pakan sekoittamiseen käytettiin seuraavaa kutsua: *return this.deck.shuffle();* Uusi koodi otettiin käyttöön, joka korjasi ongelman ja kortit sekoittuivat paremmin.

3.3 Animaatio

Animaatiot ovat tärkeä osa keräilykorttipeliä, sillä animaatioita tapahtuu esimerkiksi hyökkäyksissä, sekä liikkumisessa. Tämän vuoksi tutkittiin kuinka saadaan animaatio sujuvaksi ilman, että siinä näkyy viivettä tai muuta ”virheitä” kuten ääriviivoja piirtoalueelle. Ongelmana on saada tarpeeksi sujuvaa animaatiota, jotta käyttäjälle tulee hyvä pelikokemus ja ettei animaatiossa tapahdu hidastumisia.

Animaatio voidaan tehdä seuraavalla tavalla, jossa käytetään hyväksi *requestAnimFrame()* - ja *setTimeout*- funktiota, joiden avulla lasketaan kuvataajuus (FPS eli Frames per second) animaatiolle. Tämän jälkeen saadaan päivitettyä elementtejä piirtoalueelle. Tämä toteutus tosin ei aina anna animaatiolle parasta lopputulosta. Tämän lisäksi pitää seurata kuinka paljon aikaa on kulunut edellisestä piirtämisestä ja pikseleitä apuna käyttäen voidaan saada suhteellisen sujuvaa animaatiota. Pikseleiden käytössä koodissa kerrotaan kuinka monta pikseliä animaatio voi liikkua sekunnissa. [72]

Alla on koodi, jossa käytetään hyväksi *setTimeout*-funktiota, sekä *requestAnimFramea* (kommenteissa kirjoitettu auki koodia)[73]

```
window.requestAnimFrame = (function(callback){ //requestAnimFrame
    return window.requestAnimationFrame ||  //selainkohtaiset tarkistukset
    window.webkitRequestAnimationFrame ||
    window.mozRequestAnimationFrame ||
    window.oRequestAnimationFrame ||
    window.msRequestAnimationFrame ||
    function(callback){
        window.setTimeout(callback, 1000 / 60); //setTimeOut-funktion käyttö:
        //60 ms
    };
})();

(function animloop(){
      requestAnimFrame(animloop);
      render(); //funktio jolla kuva piirretään
    })();
```

Lisäksi Erik Möller (Opera-suunnittelija) kirjoittaa blogissaan, että *requestAnimFrame* pitää kutsua jokaista kuvaa ennen, mutta jos sen kutsuu kuvan jälkeen, niin animaatio hidastuu. Erik Möller antaa blogissaan ratkaisun kyseiselle ongelmalle ja alla on koodi [74]:

```
(function() {
    var lastTime = 0;
    var vendors = ['ms', 'moz', 'webkit', 'o'];
    for(var x = 0; x < vendors.length && !window.requestAnimationFrame; ++x) {
        window.requestAnimationFrame = window[vendors[x]
+'RequestAnimationFrame'];
        window.cancelRequestAnimationFrame = window[vendors[x]+
          'CancelRequestAnimationFrame'];
    }

    if (!window.requestAnimationFrame) //jos animaatio
        window.requestAnimationFrame = function(callback, element) {
            var currTime = new Date().getTime();
            var timeToCall = Math.max(0, 16 - (currTime - lastTime));
            var id = window.setTimeout(function() { callback(currTime +
timeToCall); },
              timeToCall);
            lastTime = currTime + timeToCall;
            return id;
        };

    if (!window.cancelAnimationFrame) //jos tapahtuu animaation keskeytys
        window.cancelAnimationFrame = function(id) {
            clearTimeout(id);
        };
}())
```

Yllä olevaa koodia voi kutsua missä tahansa, jälkeen tai ennen kuvaa ilman, että animaatio hidastuisi. Koodi tarkistaa edellisen kutsun ajankohdan `new Date().getTime()` -funktion avulla ja laskee viivästyksen, joka on lähinnä 60 FPS:ää. Viivästys tulee olemaan joko 4 ms tai 6 ms. [73, 74].

3.3.1 Kaksoispuskurointi

Animaation nopeuteen vaikuttaa suuresti myös se, että käytetäänkö ”*off-screen*”-piirtoaluetta. Tämä tarkoittaa sitä, että käytetään piirtoaluetta, jota ei näytetä käyttäjälle (on piilossa) ja piirretään animaatio sinne, jonka jälkeen kuva kopioidaan näkyvälle alueelle. Tätä etenkin käytetään jos piirtoalueella on paljon kuvia päällekkäin piirrettävänä. Esimerkiksi kuva voi

sisältää kaksi kuvaa päällekkäin, ja vielä tekstiä. Kun käytetään piilotettu piirtoaluetta, niin saadaan piirrettyä ennen animaatiota kuva ja parannettua animaatiota. Tällöin saadaan piirto päivittymään ilman keskeytyksiä. Tätä ominaisuutta kutsutaan myös kaksoispuskuroinniksi. [75, 76] Selaimet kuitenkin hoitavat kaksoispuskuroinnin automaattisesti. [76]

Animaation voidaan nopeuttaa lisäksi kahdella muullakin tavalla (*Clipping* ja *Blitting*). *Clipping*:ssä animaation tausta piirretään uudestaan, mutta vain niiltä osin, joissa sitä tarvitaan. Tällöin piirtäminen on nopeaa, koska ei tarvitse koko taustaa piirtää uudestaan vaan osa siitä. Jos animaatiossa on paljon osia, jotka liikkuvat yhtä aikaa, *Clipping* tulee olemaan hitaampaa kuin koko taustan piirtäminen uudestaan. *Blitting*:ssä taustakuva otetaan muistiin ei näkyvään piirtoalueeseen, josta se kopioidaan näkyvälle piirtoalueelle. Keräilykorttipelissä ei ole paljon liikkuvia osia animaatiossa, vaan esimerkiksi hyökkäykset ja liikkumiset tehdään vuorotellen vastustajan kanssa. Tästä syystä keräilykorttipelissä ”*Clipping*”-ominaisuutta käytetään, ja tämän lisäksi pelin animaatio käyttää *setTimeout*-funktiota, sekä *requestAnimFramea*. Tästä voidaan päätellä, että asiakasohjelmassa animaatio on kirjallisuuden ja muiden viitteiden mukaan ”kunnossa” ja optimoitu.

3.4 Drag and Drop – tuki

Elementtien siirtely on mahdollinen ominaisuus pelille, esimerkiksi korttien siirtely ja järjestely voidaan toteuttaa tällä tavoin. Ongelmana on tuki Opera-selaimen osalta, jossa HTML5 ”*drag and drop*”-tukea ei ole. Selaimesta toiseen siirtely ei tällöin onnistu käyttäen puhtaasti HTML5-tekniikkaa, joten vaihtoehtoisia toteutustekniikoita tarvitaan.

Opera-selaimessa ”*drag and drop*”-ominaisuuden voi tehdä JavaScriptin avulla. Tähän on olemassa myös kirjastoja apuun kuten jQuery, jolla tämän voi toteuttaa suhteellisen vaivattomasti. Esimerkiksi kun haluaa <*div*>-elementtiin ”*drag&drop”*-ominaisuuden tarvitaan seuraava koodi:

```
$( "#draggable" ).draggable();
```

Yllä olevalla koodilla annetaan sellaiselle elementille veto ominaisuus, jossa *id* on sama kuin ”*draggable*”. Kyseistä tekniikkaa käytettiin hyväksi kun tehtiin keräilykorttipeliin pakkaeditorin prototyyppi.

3.5 Safari jQuery bind

Aiemmin pelissä käytettiin jQueryn *bind*-funktiota, jota käytetään funktioiden liittämiseen objekteihin. Näin objektissa esimerkiksi napissa voi olla funktion kutsu. Myöhemmissä versioissa *bind*-funktio ei enää toiminut Safarissa. *Bind*-funktiota käytetään jQueryssä, ja sen toimimattomuudesta on viitteitä muuallakin [77]. Viitteen [77] mukaan jQuery 1.6.4 toimisi *bind:*n kanssa, mutta ei 1.7.1 versiossa. Kyseistä asiaa testattiin pelissä vaihtamalla jQueryn versio 1.6.4 versioon, mutta testattaessa ongelma pysyi myös vanhemmalla versiolla.

jQueryn-sivuilla kerrottiin, että 1.7.1 versiossa ”*bind*”-metodi on muuttunut ”*on*” metodiksi ja se olisi suositeltavampi käytettäväksi kuin *”bind”* [78, 79]. Muutettaessa ”*bind*”-metodi ”*on*”-metodiksi, peli ei enää toiminut ollenkaan. Tämän jälkeen selvitettiin milloin virhe oli tarkalleen tullut käyttäen TortoiseGit-ohjelman lokia, josta saatiin pelin aikaisemmat versiot esiin. Lokia selatessa huomattiin, että pelissä käytettiin aluksi Prototype.js Javascript liitännäistä, mutta myöhemmin se poistettiin ja siirryttiin täysin jQueryyn. Tämän jälkeen peli ei enää toiminut Safarissa. Kuitenkaan koodissa ei tullut mitään muutoksi *bind*:n käyttämiseen. Koodin olisi pitänyt tulla muutoksia mikäli käytettiin Prototypen *bind*-funktiota, koska *bind* ei toimi jQueryssä samoin kuin Prototypessa. [80]

Ratkaisuksi saatiin testaamisen kautta seuraava: Prototype.js liitettiin peliin mukaan ja *bind*()-funktio toimi tämän jälkeen myös Safari-selaimessa. Tämä vaihtoehto otettiin käyttöön pelissä, joka korjasi myös ongelman. Tämä oli yksinkertaisin vaihtoehto. Toinen ratkaisu (jota ei toteutettu peliin) olisi käyttää sulkeumaa kuten alla oleva esimerkki, jossa *bind*() muutetaan toiseen muotoon [80]:

```
var func = myFunction;
```

setTimeout(func.bind(this), 1000);

Muuttuisi seuraavaksi:

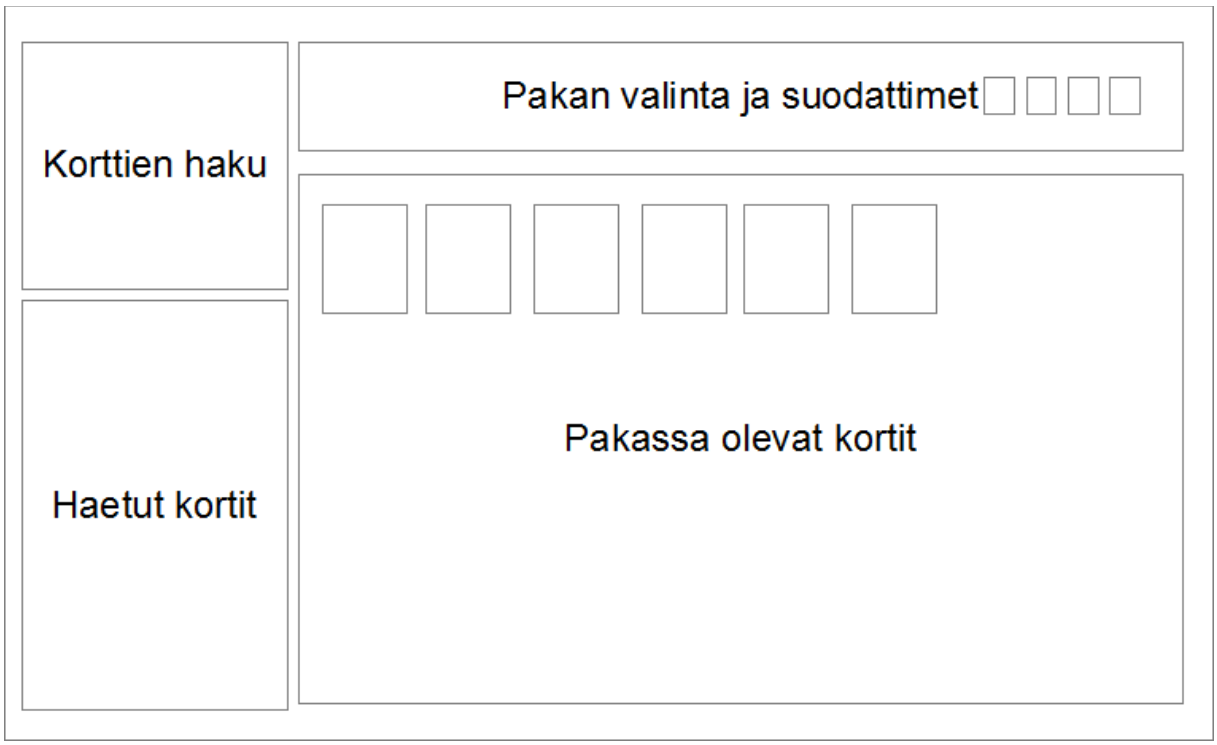
var that = this;

window.setTimeout(function(){ that.myFunction(); } , 1000);

Tämä ratkaisu kuitenkin vaatisi enemmän muutoksia itse pelin koodiin koska jokaista *bind()*-kutsua pitäisi muokata. Tämän vuoksi pelissä päädyttiin ensimmäiseen vaihtoehtoon, jossa liitettiin Prototype.js peliin mukaan, joka ratkaisi ongelman.

4 Keräilykorttipelin pakkaeditori

Käytännön toteutuksena diplomityössä toteutettiin keräilykorttipelin pakkaeditorin prototyyppi, jonka pohjalta Seepia Games Oy tulee toteuttamaan lopullisen pakkaeditorin. Editorin tarkoitus on antaa käyttäjälle mahdollisuus rakentaa oma pakka haluamistaan korteista ja hallinnoida omia pakkojaan (poistaa, lisätä pakkoja). Seepia Games Oy:n kanssa tehtiin alustava suunnitelma pakkaeditorin prototyypin ulkonäöstä, joka on esitetty kuvassa 11.



Kuva 11: Suunnitelma pakkaeditorin prototyypistä

Prototyypin näyttö jakautuu viiteen osaan (Kuva 11): korttien haku, korttien näyttäminen (haetut kortit), pakan sisällön näyttäminen (pakassa olevat kortit), sekä pakan valinta ja suodattimet. Korttien haussa voidaan käyttää, joko kortin luokkia tai tekstikentän hakua, sekä mahdollisesti molempia yhtä aikaa. Kortit näytetään käyttäjän haun perusteella. Mikäli hakua ei ole, niin näytetään kaikki kortit. ”Pakan valinta”-osassa tehdään käyttäjän pakkojen valinta, poisto ja lisäys. Pakan sisältö näytetään käyttäjän valitsemien suodattimien mukaan. Suodattimia pakan korteille on korttien pääluokat, jotka ovat ”*creature*”, ”*spell*”, ”*utility*” ja

”*battle*”. Sen lisäksi, että näytetään korttien pääluokat, voidaan myös näyttää kaikki kortit pakasta. Lisäominaisuutena korttien kokoa voisi muuttaa isosta pienemmäksi.

4.1 Tekniset vaatimukset

Pakkaeditorin tekemisessä käytetään lokaalia SQLite-tietokantaa. Sen pitää toimia selaimessa käyttäen PHP, jQuery ja Javascript-tekniikoita, sekä mahdollisesti tiedonsiirtotekniikoita kuten Ajax. Mahdollinen ominaisuus on myös ”*drag&drop*”, jonka avulla kortit siirretään pakkoihin. Ominaisuus voidaan myös toteuttaa toisella tavalla esimerkiksi lisäyspainikkeen avulla.

Pakkaeditorin pitää näyttää kaikki mahdolliset kortit käyttäjän haun ja suodattimien mukaan. Käyttäjällä on oltava mahdollisuus pakkojen lisäykseen ja poistamiseen. Tämän lisäksi pakkojen sisältö on oltava näytettävissä sekä muokattavissa. Korttien tiedot näytetään esimerkiksi terveys ja voimakkuus. Pakan sisällöt voidaan näyttää suodattimien avulla. Kortit pakasta pitää voida siirtää pakkoihin, joko ”*drag&drop*” tai muulla korvaavalla menetelmällä. Korttien kokoa voidaan muuttaa, sekä oman pakan kortteja voidaan suodattaa pääluokilla.

4.2 Toteutus

Alla olevissa osioissa on selvitetty tärkeimmät alueet toteutetun pakkaeditorin prototyypissä.

4.2.1 Sisällön päivittäminen

Asiakasohjelman (pakkaeditorin prototyyppi) pitää kommunikoida palvelimen kanssa. Esimerkiksi käyttäjä valitsee pakan, jolloin asiakasohjelma hakee pakan sisällön palvelimelta sijaitsevasta tietokannasta. Asiakasohjelman puolella kutsutaan funktiota, sekä php-tiedostoa (jossa funktio sijaitsee). Näiden avulla sisältö luodaan näytettäväksi selaimeen. HTML-tiedostossa voidaan näyttää php:n palauttama sisältö kahdella tapaa, joko ajax:n avulla tai jQuery-funktioilla. Alla oleva on yksinkertainen Ajax-kysely, joka hakee getDecks.php tiedoston ja lähettää ”*user*”-tiedon palvelimelle:

```
var queryString = "?user="+user ;
ajaxRequest.open("GET", "getDecks.php" + queryString, true);
ajaxRequest.send(null);
```

Tällöin tarvitaan myös asiakasohjelmaan tiedon palautus, jossa alla on yksi esimerkki. Kyseisessä esimerkissä odotetaan kunnes kysely on suoritettu. Tämän jälkeen vastaus otetaan muistiin:

```
ajaxRequest.onreadystatechange = function(){
        if(ajaxRequest.readyState == 4){
                //vastaus otetaan muistiin:
                var response= ajaxRequest.responseText;
        }
}
```

Prototyyppi tehtiin aluksi Ajax-kyselyillä, mutta pian toteutuksen jälkeen huomattiin, että Ajax-kyselyt voidaan myös korvata jQueryllä. Jolloin koodin voi kirjoittaa alla olevalla tavalla:

```
$.get("getDecks.php?user="+user, function(data){
        //tieto saatu palvelimelta, ja näytetään selaimessa:
        alert("Data Loaded: " + data);
});
```

Kummassakin kyselyt tehdään Ajax-tekniikalla, joten nopeus on sama kummassakin vaihtoehdossa. Pakkaeditoriin toteutus tehtiin aluksi ensimmäisen vaihtoehdon mukaan, mutta koska tekniikka ei eroa tai tuo lisänopeutusta pakkaeditoriin jQueryn käyttö on vain koodillisesti ”siistimpää” kuin Ajaxin käyttö. Pakkaeditorin prototyyppiin käytettävä aika oli rajallista, joten jQueryn käyttäminen Ajax kyselyissä jätettiin seuraavaan versioon.

4.2.2 Tietokanta

Keräilykorttipeliin on tehty valmiiksi tietokanta, jossa ovat tiedot korteista, pelaajista, sekä pelilaudasta. Tietokannassa on tarvittavat tiedot korteista. Keräilykorttipelissä käytetään SQLite-tietokantaa. Pakkaeditori käyttää aluksi lokaalia tietokantaa, jonka vuoksi editoriin

pitää tehdä ominaisuus, jolla luetaan tietokanta omasta koneesta. Tietokanta sijaitsee pelin lopullisessa versiossa palvelimella. SQLite-ominaisuus voidaan suoraan toteuttaa PHP:n avulla seuraavalla tavalla:

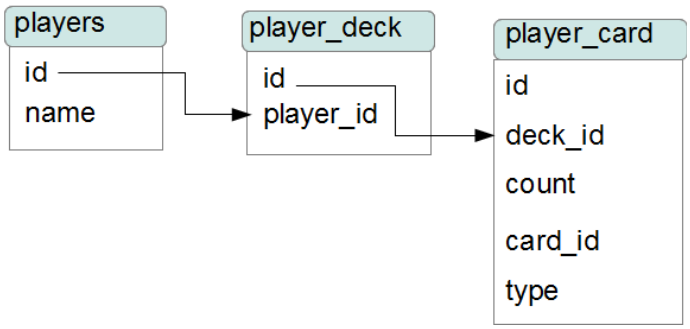
```
$db = new PDO("sqlite:new_permiaDB.sqlite");
```

Kyseinen tapa viittaa ”*new_permiaDB*”-tietokantaan. Kyselyt toimivat suurin piirtein samalla tavalla kuin esimerkiksi MySql-kyselyissä (alla esimerkki):

```
$db->query("SELECT count FROM player_deck " );
```

SQLite vaatii ”*php_pdo_sqlite*”-ominaisuuden päälle PHP:n ini-tiedostoon. Tällöin poistetaan kommentit php.ini-tiedostosta kyseisestä lauseesta, jolloin SQLite toimii.

Keräilykorttipelin tietokannassa oli valmiiksi annettu korttien tiedot kuten tyypit, nimet, hyökkäykset, erikoiset kyvyt, liikkumisnopeus. Tietokanta toteutettiin prototyypin yhteydessä seuraavanlaiseksi, joka koostuu kolmesta taulusta (kuva 12): *players*, *player_deck* ja *player_card*



Kuva 12: Pakkaeditorin tietokanta

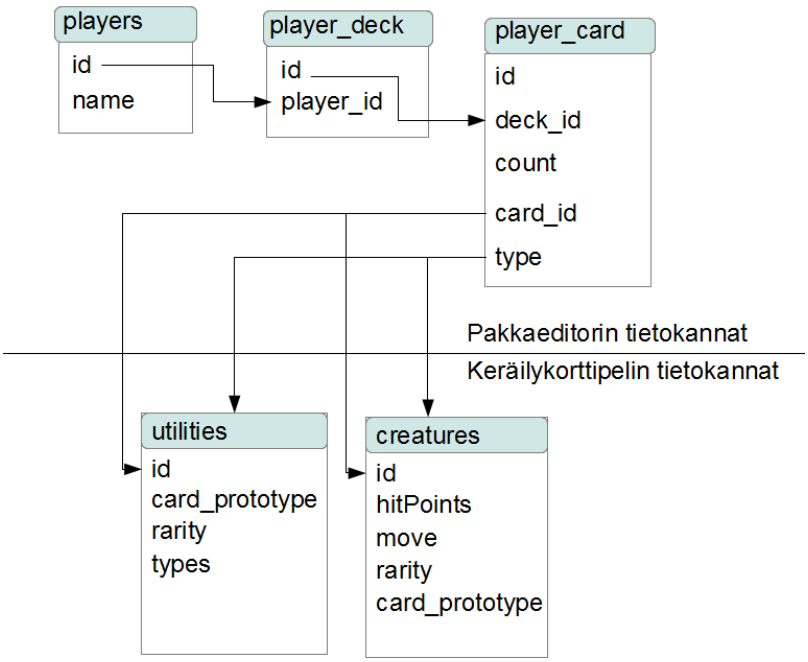
Tietokanta on rakennettu niin, että pelaajalla on tiedot ”*players*”-taulussa. Kyseessä ei ollut lopullinen tietokantataulu (tarkoittaen sitä, että pelaaja tauluun tullaan tallentamaan enemmänkin tietoa), joten pelaajan tiedoissa on ainoastaan pelaajan nimi. ”*Players.id*” (eli players-taulun id tieto) on taulukon pääavain, jonka avulla pelaaja tunnistetaan. Pääavaimia jokaisessa taulussa on niiden ”*id*” tieto. Pelaajan pakkoihin myös tullaan lisäämään myöhemmin enemmän tietoja kuten esimerkiksi pakan nimi, jota voi muuttaa.

Taulukossa ”*player_deck*” sijaitsee kaikki korttipakat, joita pelaajilla on olemassa. ”*id*” on pääavain myös ”*player_deck*”-taulussa. Tällöin jokaisella pakalla on yksilöllinen tunniste. Pakan lisäys ja poistaminen käy helposti taulukossa sijaitsevan ”*id*”-tiedon avulla. Alla esimerkki SQLiten poistosta, jossa *”$value1*”-muuttuja on pakan *”id”* :

$db->exec("DELETE FROM player_deck WHERE player_deck.id IN('".$value1."') ");

Aluksi tietokannasta poistetaan pakassa olevat kortit ja sen jälkeen pakka. Lisäksi kaikki pelaajan korttipakat on ne helppo etsiä *player_id*:n perusteella.

”*player_card*”-taulussa on tiedot korttipakkaan kuuluvista korteista. Korttien tietoja ovat kortin lukumäärä (”*count*”), sekä kortin sijainti (”*card_id*”). ”*Card_id*” (kortin identifiointi numero) on linkitetty kortin tietoihin (Kuva 13). Kortin tyyppi (”*type*”) kertoo minkälaisesta kortista on kyse. Kortin tyypin perusteella selvitetään mistä taulukosta kortin tietoja haetaan (*utilities*, *creatures*, *leaders*, *spells* tai *battle)*, ja kortin identifiointi numerolla (”*card_id”*) saadaan haettua oikea kortti kyseisestä taulusta. Kortin tiedot/taulut on valmiiksi annettu keräilykorttipelissä. Alla kuva miten korttieditorin taulukot liittyvät itse pelin tietokanta rakenteeseen:



Kuva 13: Keräilykorttipelin ja pakkaeditorin tietokanta

4.2.3 Drag and drop

Korttien lisäys pakkaan ja pakasta korttien poistaminen tehtiin ”*drag&drop*”-ominaisuuden avulla. Tämä tehtiin alla olevan jQuery-koodin avulla, jossa korttilista (koodissa *#cardList*) yhdistettiin pakkaan (koodissa *#sortable*). Kortteja ei poistu listalta kun niitä vedetään, ja kortin klooni liikkuu vedettäessä *helper:clone:*n avulla *:*

```
$( "#cardList li" ).draggable({
        connectToSortable: "#sortable",
        helper: "clone",
        revert: 'invalid',
        cursor: "cursor"
});
```

Tämän lisäksi pakkaan (*sortable*-lista) tehtiin funktiot, jotka lisäävät kortin tietokantaan aina kun kortti vedettiin korttipakkaan. Tämä saatiin toteutettua niin, että tarkistettiin milloin hiiri on pakan ulkopuolella, päällä, ja milloin pakkaan vedetään kortti. Kun kortti on pakka-alueen päällä, *sortable*-muuttuja vaihdetaan ykköseksi. Lisäksi kortti poistetaan jos *sortable*-muuttuja on nolla. Alla koodi, jolla lisääminen pakkaan tunnistetaan ja poisto:

```
$( "#sortable" ).sortable({
        cursor: "cursor",
        tolerance: 'pointer',
        receive: function(e, ui) { //kun lisätään jotain pakkaan, kutsutaan funktiota joka lisää tietokantaan :
                sortableIn = 1; add_card(Deck, ui.item.attr('id2'),ui.item.attr('class2'), user);
        },
        over: function(e, ui) { sortableIn = 1; }, //kun on pakan päällä, sortableIn=1
        out: function(e, ui) { sortableIn = 0; }, //kun on pakan ulkopuolella sortableIn=0
        beforeStop: function(e, ui) {
                if (sortableIn == 0) {  //jos hiiren osoitin pakan ulkopuolella,
                //kutsutaan funktiota joka poistaa kortin id:n ja tyypin mukaan:
```

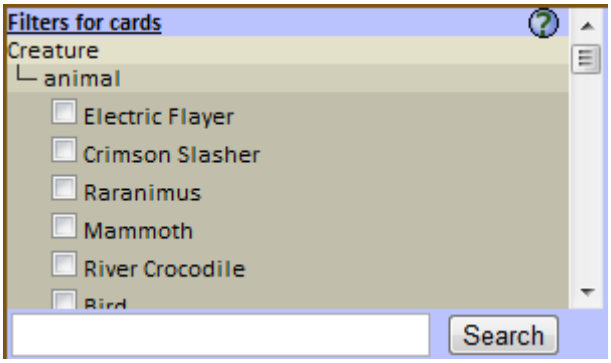
```
                remove_card(Deck, ui.item.attr('id2'),ui.item.attr('class2'), user);ui.item.remove();
            }
}).disableSelection();
```

4.3 Lopputulos

Pakkaeditorin prototyyppi toteutettiin halutuilla ominaisuuksilla. Pakkojen korttien muuttaminen jouduttiin jättämään prototyypistä pois ajan puutteen vuoksi. Prototyypissä pakkoja voi lisätä ja poistaa. Pakkojen poistaminen tapahtuu ”*drag&drop*”-ominaisuuden avulla. Tällöin pakka siirretään pakan poiston päälle (kuvassa 14 miinusmerkki). Pakan lisäys toimii painamalla kuvan 14 ”+”-merkkiä. Tällöin pakka lisätään käyttäjälle ilman mitään kortteja.

Kuva 14: Pakan poisto ja lisäys

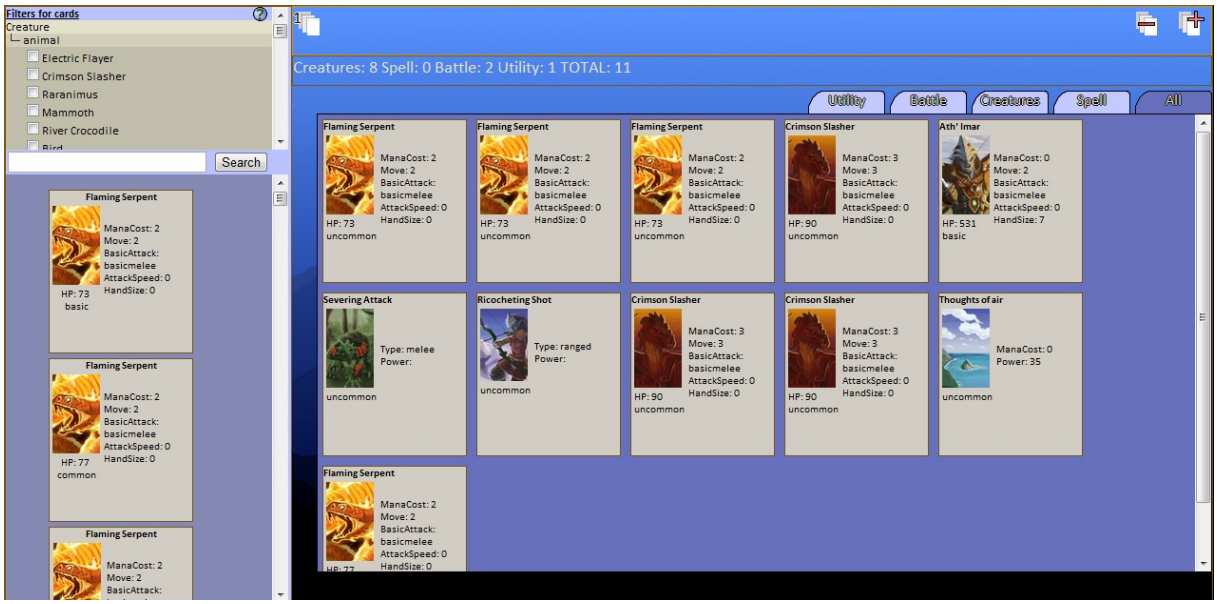
Pakkojen sisältämät kortit näytetään käyttäjälle, sekä käyttäjä pystyy muuttamaan pakkojen sisältöjä (korttien poisto ja lisäys). Kortteja pystyy myös hakemaan. Korttien hakemiseen voidaan käyttää kahta tapaa, korttien tyyppien valinnoilla sekä hakukentällä (Kuva 15). Hakuja voidaan tehdä kumpaakin ominaisuutta käyttäen joko yhtä aikaa tai erikseen. Erikseen hakukenttään kirjoittamalla, ilman mitään valintaa korttien tyypeistä, saadaan kaikki hakukenttään osuvat tulokset. Mikäli hakukentässä ei ole mitään tekstiä ja valitaan korttien tyypeistä, niin pakkaeditori hakee pelkän tyypin mukaan. Mikäli hakukentässä on tekstiä ja jokin tyyppi valittu, otetaan kummatkin huomioon haettavissa tuloksissa.



Kuva 15: Pakkojen haku

Lisääminen ja poistaminen suoritetaan ”*drag&drop*”-ominaisuuden avulla. Kun kortti viedään pakka alueen ulkopuolelle, kortti poistetaan pakasta. Kortin lisäys toimii siten, että kortti vedetään pakkaan. Mikäli samaa kortti tyyppiä on liikaa, niin näytetään virheilmoitus ja korttia ei lisätä pakkaan. Muulloin kortti lisätään pakkaan ja päivitetään pakan näkymä.

Kuvassa 16 on lopullinen pakkaeditorin prototyyppi, jossa on vasemmalla korttien haku ja oikealla puolella pakka alue (johon kortit laitetaan). Vasemmassa yläkulmassa on pakan suodattimet, sekä pakkojen lisäys- ja poistotoiminnot.



Kuva 16: Pakkaeditori

4.3.1 Hyödynnettävyys

Prototyypistä on tarkoitus ottaa tarpeelliset ja hyödylliset osat käyttöön tuotantoversioon. Mahdolliset ongelmatilanteet voidaan välttää prototyypin avulla, koska ominaisuuksien toteutus on jo tehty prototyypissä. Prototyypistä saatiin selville, että ulkoasua pitää muuttaa paljon, etenkin käytettävyydeltään ulkoasu ei ole hyvä mobiililaitteille, johtuen monesta vierityspalkista. Tämä ei myöskään ole käytännöllistä selaimissa. Prototyyppi sopii kuitenkin toiminnallisuudeltaan tuotantoversioksi.

Prototyyppiä on suhteellisen helppo muokata. Tietokantaan lisättäessä uusia kortteja, pakkaeditori näyttää ne automaattisesti. Lisätyn kortin tiedot ja kuvat näytetään, mikäli ne ovat olemassa. Korttien tauluista haetaan SQLite-komennoilla kaikki tiedot ja kuvat. Tätä ominaisuutta voidaan myös hyödyntää tuotantoversiota kehittäessä. Tuotantoversio käyttää Postgres-tietokantaa, joten tietokannan tuki pitää muuttaa prototyypin toteutuksessa, mikäli sitä jatko kehitetään yhteensopivaksi peliin. Tietokantaa vaihtaessa SQLitestä toiseen (Postgres-tietokantaan / PostgreSQL), pitää ohjelmoida pakkaeditori vastaamaan tätä. Muokattavaa on tällöin tietokannan kyselyissä. Suurin osa SQLite-kyselyistä toimii myös Postgres-kyselyissä kun vain osa koodista muutetaan, esimerkiksi haku SQLite-tietokannasta toimii alla olevalla tavalla (ensimmäisellä rivillä tietokannan nimi, toisella tietokannasta haku):

```
$db = new PDO("sqlite:new_permiaDB.sqlite");

$result = $db->query("SELECT title, name, address FROM authors");
```

Yllä olevan koodin voi myös toteuttaa toisellakin tavalla:

```
$dbhandle = sqlite_open('sqlitedb');

$query = sqlite_query($dbhandle, 'SELECT title, name, address FROM authors');
```

Koodin voi muuttaa Postgres-tietokantaa tukemaan seuraavalla koodin muunnoksella, jossa ”*sqlite_open*” korvataan ”*pg_pconnect*” sanalla ja ”*sqlite_query*” korvataan puolestaan ”*pg_query*”:llä :

```
$conn = pg_pconnect("dbname=new_permiaDB");
```

$result = pg_query($conn, "SELECT title, name, address FROM authors");

Tällöin SQLiten vaihto Postgres-tietokantaan sujuu helposti. Prototyypin SQLite-komentoja ja kyselyitä voidaan käyttää sellaisenaan myös tuotantoversiota tehdessä, mikäli tietokanta rakenne pysyy suunnitelman mukaisena.

4.3.2 Yhteenveto

Pakkaeditorin prototyypistä on tarkoitus tehdä uusi versio, jossa mahdollisesti käytetään prototyypin osia kehityksessä. Keräilykorttipelin tietokanta muuttui SQLitestä Postgres-tietokantaan ja tämä vaatii muutokset myös pakkaeditorin osalta tuleviin versioihin. Tuotantoversioon pitää tehdä käyttäjän tunnistus, jonka avulla pelaajat voivat rakentaa omia pakkojaan ja tallentaa ne tietoihinsa. Prototyypin ulkoasua pitää uudistaa vastaamaan itse keräilykorttipelin ulkoasua. Pakkaeditorissa käytetään Ajax-tiedonsiirtotekniikkaa, joka toteutetaan myöhemmässä versiossa jQueryn Ajax toteutuksella yhtenäisyyden vuoksi. Tämä ei kuitenkaan tuo nopeutusta tiedonsiirtoon. Pakkaeditorin lopullista versiota käytetään niin tietokoneilla kuin mobiililaitteilla. Tämän vuoksi käyttöliittymän pitäisi olla skaalautuva, jotta käyttäjä pystyisi käyttämään pakkaeditoria sujuvasti. Nykyisen version (prototyyppi) käyttöliittymä on staattinen ja suunniteltu isoimmille näytöille.

Alkuperäiseen suunnitelmaan verrattuna pakkaeditorin prototyypin toiminnallisuudet onnistuivat hyvin. Ainoaksi toiminnalliseksi puutteeksi prototyypissä koettiin korttien koon muuttaminen. Lisäksi vain oleelliset tiedot korteista näytetään, ja osa korttien tiedoista puuttuu. Esimerkiksi erikoishyökkäykset puuttuvat. Tämä saadaan helposti lisättyä ja silloin puuttuvat tiedot näytetään. Testattaessa pakkaeditori toimi jokaisessa viidessä yleisimmässä selaimessa. Testattaessa tehtiin jokaisen pakkaeditorin toiminto: pakan lisäys, pakan poisto, kortin haku, kortin lisäys, kortin poisto.

5 Pohdinta ja johtopäätökset

HTML5-tekniikkassa ilmeni vähän ongelmia selainten kesken. Eniten niitä ilmeni Internet Explorerissa, jossa HTML5:n tuki oli muita selaimia huonompaa. Yleisiä, kaikkiin selaimiin liittyviä ongelmia ilmeni vähän ja näitä esiintyi pelin varhaisessa versiossa. Yksi näistä yleisistä ongelmista oli satunnaislukugeneraattori, jossa käytettiin *Sort()*-funktiota. HTML5:en d*rag&drop*-ominaisuutta ei tue vielä kovin moni selain, joten tämän joutuu tekemään Javascriptillä. Selaimet kuitenkin parantavat HTML5-tekniikkaan joten piakkoin tämänkin voi tehdä puhtaasti HTML5:lla.

WebSocket-tiedonsiirtotekniikka on paljon tehokkaampi kuin muut tekniikat. Pian kaikki selaimet myös tukevat tätä tekniikkaa samalla kun HTML5:en määrittely paranee. Aikaisemmin tekniikka oli monessa selaimessa kytketty pois päältä, mutta kun määrittely parantui, moni selain päivitti tukeaan.

Pakkaeditorin prototyyppi onnistui hyvin myös eikä ongelmia tullut vastaan. Prototyyppi toimi viidellä yleisimmällä selaimella. Jatkotutkimusaiheita olisi HTML5:en ja Javascriptin toiminta myös muissa selaimissa, koska eroavaisuuksia on olemassa tälläkin hetkellä (Internet Explorerissa on suurimmat eroavaisuudet viidestä yleisimmistä selaimesta). Tämä työ keskittyi vain keräilykorttipelin kannalta oleellisiin tekniikoihin. Tutkimusta voisi laajentaa muihin HTML5 osa-alueisiin, sekä tutkia toimivatko muut osa-alueet selaimissa samalla tavalla ja onko näiden välillä eroavaisuuksia. Lisäksi ääntä tutkittiin vain selainten koodekkien tukien perusteella eikä ollenkaan käytännön tasolla. Tutkimusta laajentaminen eri äänien käytännön toteutuksiin selaimissa ja mobiililaitteisiin olisi aiheellista.

6 Yhteenveto

HTML5-tekniikka on kehittynyt nopeasti ja moni selain tukee tekniikkaa. Osa selaimista on kuitenkin jäljessä tekniikan tukemisessa, kuten Internet Explorer. Tämä vaikeuttaa HTML5-tekniikan käyttöä esimerkiksi verkkopeleissä, mikäli peliä pitäisi pystyä pelaamaan jokaisessa selaimessa. Selaimet kuitenkin parantavat HTML5:n tukea jatkuvasti. Työssä koottiin keräilykorttipelin kannalta oleelliset HTML5-tekniikan osa-alueet, joita peli tulee käyttämään. Tämän jälkeen tarkistettiin selainten tuki kyseisille osa-alueille kirjallisuudesta. Mikäli yhteensopivuus ongelmia löytyi, tutkittiin mikä olisi paras mahdollinen vaihtoehto ongelman ratkaisemiseksi. Yleisimmät selaimet eivät tue kaikki samaa ääni koodekkia. Esimerkiksi Internet Explorer 9 ei tue Wave-koodekkia kun muut selaimet tukevat. Näin ollen pelissä pitää käyttää kahta koodekkia, mikäli halutaan saada äänet pelissä toimimaan viidessä yleisimmässä selaimessa. Äänien tukea ei tutkittu työssä tarkemmin (vain koodekkien tuki selvitettiin selainten osalta).

Työssä tutkittiin kirjallisuudesta selainten tiedonsiirtotekniikoiden tukia ja nopeuksia. WebSocket havaittiin nopeimmaksi tekniikaksi, mutta selainten tuki sen osalta oli vaihtelevaa; esimerkiksi Internet Explorer ei tue ollenkaan kyseistä tekniikkaa. Tämän vuoksi vaihtoehtoisia tekniikoita selvitettiin aluksi kirjallisuuden perusteella, joista valittiin pari yleistä tekniikka, sekä ominaisuuksiltaan paras tekniikka, Socket.io. Yleiset tekniikat olivat XmlHttpRequest sekä Ajax. Yleisien tiedonsiirtotekniikoiden nopeuksia vertailtiin. Tämän jälkeen testattiin niiden toimivuus, sekä verrattiin nopeutta Socket.io-tekniikkaan. Kirjallisuudesta, sekä testeistä kävi ilmi, että Socket.io käyttää WebSocket-tekniikkaa mikäli mahdollista. Muulloin tekniikka käyttää seuraavaksi parhainta tiedonsiirtotekniikkaa. XmlHttpRequestiin ja Ajaxiin vertailtaessa Socket.io-tekniikka oli paljon nopeampi WebSocketin käytön vuoksi. Socket.io siis soveltuu hyvin keräilykorttipeliin, sekä on paljon monipuolisempi. Se ei myöskään rajoitu pelkkään yhteen tekniikkaan ja monipuolisempi kuin muut tekniikat kuten Faye ja Comet. Socket.io-tekniikan hyviä puolia on sen nopeus ja vähäinen kuormitus. Tämän lisäksi tekniikka mahdollistaa useamman tiedonsiirtotekniikan käytön, jolloin selaimet voivat automaattisesti päättää mitä tekniikka käyttävät. Tällöin

esimerkiksi Chrome-selain käyttää WebSocket-tekniikka, kun taas Opera käyttää Flashsockettia (johtuen virheestä Socket.io-tekniikassa).

Keräilykorttipelissä ilmeni virhe satunnaislukugeneraattorissa, jonka vuoksi kortit eivät jakautuneet tasaisesti ja tiettyä korttia tuli useammin kuin muita. Kyseistä funktiota testattiin ja huomattiin, että funktio ei toimi tasajakauman mukaisesti missään selaimessa, vaikka selainten välillä oli eroja ja joissakin selaimissa satunnaisuus toimi paremmin. Satunnaislukugeneraattorin funktiolle etsittiin korvaava funktio. Kyseinen funktio testattiin samalla tavalla kuin alkuperäinen funktio. Tulokset vahvistivat, että korvaava funktio oli parempi kuin alkuperäinen, joten funktio muutettiin. Animaation osalta tutkittiin onko se optimoitu tarpeeksi hyvin ja miten saadaan hyvää animaatiota aikaan. Tämä tehtiin kirjallisuuden avulla, josta vertailtiin saatuja ohjelmointi vinkkejä keräilykorttipelin koodiin.

Lopuksi toteutettiin keräilykorttipeliin prototyyppi pakkaeditorista, jossa pelaajat voivat tehdä omia pakkojaan halutuista korteista. Pakkaeditoriin käytettiin *drag&drop*-ominaisuutta korttien siirtelyyn, sekä korttien ja pakkojen poistamiseen. SQLite-tietokantaa ja Ajax-kyselyillä asiakasohjelmalle saatiin toimivuutta ja sisältö päivitettiin käyttäjän toimintojen mukaan. Pakkaeditori saatiin toteutettua suurimmilta osin suunnitelman mukaan. Yksi toteuttamaton ominaisuus oli pakoissa olevien korttien koon muuttaminen. Ulkonäöltään pakkaeditori oli hieman erilainen itse pelistä ja skaalautuvuus pienemmissä näytöissä puuttui, joiden toteutus jää tuotantoversioon.

Työssä tutkittiin HTML5:n tukea viidellä yleisimmällä selaimella. Pienempien selainten tuki tähän uuteen tekniikkaan jäi tutkimuksen ulkopuolelle. HTML5-tekniikka soveltuu hyvin pelien tekoon selaimissa, sekä korvaa muita tähän tekniikkaan tarkoitettuja tekniikkoja. Flash-tekniikka on ollut suosittu internet pelien tekemiseen, mutta HTML5 tuo paljon uusia ominaisuuksia, jotka tekevät tekniikasta suosittua. HTML5 soveltuu useammalle laitteelle paremmin kuin Flash ilman mitään erikseen asennettavia lisäosia. Kuormituksen testaaminen tiedonsiirtotekniikoiden saralla jäi tutkimuksen rajatun alueen ulkopuolelle. Tässä työssä kuormitusta tutkittiin vain kirjallisuuden avulla, ja käytännön testauksia ei toteutettu.

Tiedonsiirtotekniikoiden nopeuksia tutkittiin kirjallisuudesta, sekä testattiin käytännössä. Tutkimus teki HTML5:n osalta vertailun, joka liittyi keräilykorttipelin ominaisuuksiin kuten ääneen, videoon, kuvaan ja tiedonsiirtoon. Tutkimuksien laajentaminen HTML5:n muihin osa-alueisiin olisi hyvä tulevaisuuden tutkimuskohde.

Viitteet

[1] Vaughan-Nichols, S.J. 2010. Will HTML 5 Restandardice the Web?, ISSN: 0018-9162

[2] HTML5. [verkkodokumentti]. Päivitetty: 02.2012, [viitattu 09.02.2012] Saatavissa: http://dev.w3.org/html5/spec/Overview.html

[3] Dennis j. Bouvier, 1995. Version and standards of HTML, ACM SIGAPP Applied Computing Review - Special issue on the Web ISSN: 1559-6915

[4] RFC 1866- Hypertext Markup Language – 2.0. [verkkodokumentti]. Päivitetty: 1995, [viitattu 09.02.2012] Saatavissa: http://tools.ietf.org/html/rfc1866

[5] Peter Lubbers. 2012. Pro HTML5 Programming, Overview of HTML5. ISBN: 978-1-4302-3865-2

[6] Chuck Musciano, Bill Kennedy. 1998. HTML: The Definitive Guide, Thrid Edition, ISBN: 1-56592-492-4

[7] HTML5. [verkkodokumentti]. Päivitetty: 22.01.2008, [viitattu 09.02.2012] Saatavissa: http://www.w3.org/TR/2008/WD-html5-20080122/

[8] 7 W3C Technical Report Development Process. [verkkodokumentti]. Päivitetty: 14.10.2005, [viitattu: 24.5.2012] Saatavissa: http://www.w3.org/2005/10/Process-20051014/tr.html

[9] 8.7 Drag and drop -HTML Standard, [verkkodokumentti]. Päivitetty: 25.09.2012 [viitattu 25.09.2012] Saatavissa: http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html

[10] HTML5 differences from HTML4. [verkkodokumentti].
Päivitetty: 2011, [viitattu 26.01.2012]
Saatavissa: http://dev.w3.org/html5/html4-differences/

[11] Mark Murphy. 2011. Beginning android 3, HTML5.
ISBN: 978-1-4302-3298-8

[12] HTML5 differences from HTML4 , [verkkodokumentti]. Päivitetty: 03.2012
[viitattu: 14.06.2012]
Saatavissa: http://www.w3.org/TR/html5-diff/

[13] HTML5 Tutorial – Input Type : Email, URL, Phone, [verkkodokumentti].
Päivitetty: (ei tietoa) [viitattu 29.06.2012]
Saatavissa: http://www.html5tutorial.info/html5-contact.php

[14] Carl A. Gutwin, Michael Lippold, T. C. Nicholas Graham
Real-Time Groupware in the Browser:
Testing the Performance of Web-Based Networking
ISBN: 978-1-4503-0556-3

[15] Andrew Binstock, 2011, IE9 Unlocks HTML5,

[16] Pinto E. 2010. A graphics library for delivering 3D contents on Web browsers,
ISBN: 978-1-4244-7607-7

[17] Internet Explorer 9 Guide for Developers. [verkkodokumentti].
Päivitetty 14.03.2011, [viitattu 20.11.2011].
Saatavissa: http://msdn.microsoft.com/en-us/ie/hh410106#_HTML5

[18] HTML5. [verkkodokumentti]. Päivitetty 20.10.2011, [viitattu 20.11.2011].
Saatavissa: https://developer.mozilla.org/en/HTML/HTML5 .

[19] Opera: Canvas element support in Opera Presto 2.2. [verkkodokumentti].
Päivitetty (ei tietoa), [viitattu 20.11.2011].
Saatavissa: http://www.opera.com/docs/specs/presto22/

[20] Safari HTML5 Audio and Video Guide: Audio and Video HTML [verkkodokumentti]. Päivitetty 05.07.2011, [viitattu 20.11.2011]. Saatavissa:

http://developer.apple.com/library/safari/#documentation/AudioVideo/Conceptual/Using_HTML5_Audio_Video/AudioandVideoTagBasics/AudioandVideoTagBasics.html

[21] Everything you need to know about HTML5 video and audio [verkkodokumentti]. Päivitetty 26.01.2011, [viitattu 23.11.2011]

Saatavissa: http://dev.opera.com/articles/view/everything-you-need-to-know-about-html5-video-and-audio/

[22] HTML5 & CSS3 Support, Web Design Tools & Support – FindMeByIP – CSS3 & HTML5 Browser Support. [verkkodokumentti]. Päivitetty (ei tietoa), [viitattu 27.11.2011]
Saatavissa: http://fmbip.com/litmus

[23] DOM Storage – MDN. [verkkodokumentti]. Päivitetty 24.10.2011, [viitattu 24.11.2011]
Saatavissa: https://developer.mozilla.org/en/DOM/Storage#sessionStorage

[24] Drag and Drop – MDN. [verkkodokumentti]. Päivitetty 14.03.2011, [viitattu 27.11.2011]
Saatavissa: https://developer.mozilla.org/En/DragDrop/Drag_and_Drop

[25] Marco Casario, Peter Elst, Charles Brown, Nathalie Wormser, Cyril Hanquez, 2011, HTML5 Solutions: Essential Techniques for HTML5 Developers. ISBN: 978-1-4302-3387-9

[26] WebSockets – MDN. [verkkodokumentti] Päivitetty 30.10.2011, [viitattu: 02.12.2011]
Saatavissa: https://developer.mozilla.org/en/WebSockets

[27] [hybi] Experiment comparing Upgrade and CONNECT handshakes. [verkkodokumentti]. Päivitetty: 2010, [viitattu 15.02.2012]

Saatavissa: http://www.ietf.org/mail-archive/web/hybi/current/msg04744.html

[28] HTML5. [verkkodokumentti]. Päivitetty 28.11.2011, [viitattu 02.12.2011]

Saatavissa: http://msdn.microsoft.com/library/hh673546.aspx

[29] Introducing web sockets. [verkkodokumentti]. Päivitetty 22.11.2010, [viitattu 02.12.2011]

Saatavissa: http://dev.opera.com/articles/view/introducing-web-sockets/

[30] HTML5 – Safari Technology Overview – Apple Developer.
Päivitetty: ei tietoa, [viitattu 02.12.2011]
Saatavissa: http://developer.apple.com/technologies/safari/html5.html

[31] Michael Moncus, 2000. JavaScript Trainer.

[32] JavaScript Overview – MDN [verkkodokumentti].
Päivitetty 21.10.2011, [viitattu 21.11.2011]
Saatavissa:
https://developer.mozilla.org/en/JavaScript/Guide/JavaScript_Overview

[33] Anderson, 0. Fortuna, E. Ceze, L. Eggers, S. 2011. Checked Load:
Architectural support for JavaScript type-checking on mobile processors.
ISBN: 978-1-4244-9432-3

[34] JavaScript – MDN [verkkodokumentti]. Päivitetty 24.10.2011,
[viitattu 26.11.2011].
Saatavissa: https://developer.mozilla.org/en/JavaScript

[35] Gabarro, S. 2007. Web Application Design and Implementation:Apache 2, PHP5, MySQL, JavaScript, and Linux/UNIX
ISBN : 9780470083963 s.171 -183.

[36] The New JavaScript Engine in Internet Explorer9 – IEBlog – Site Home – MSDN Blogs. [verkkodokumentti]. Päivitetty: 18.03.2010,

[viitattu 16.02.2012]

Saatavissa:
http://blogs.msdn.com/b/ie/archive/2010/03/18/the-new-javascript-engine-in-internet-explorer-9.aspx

[37] Jacco van Ossenbruggen, Lynda Hardman, Lloyd Rutledge, Anton Eliëns. 1997 SIGWEB Newsletter Volume 6 Issue 3, Style sheet languages for hypertext

[38] Cascading Style Sheets, level 1. [verkkodokumentti]. Päivitetty: 2008, [viitattu: 10.06.2012]
Saatavissa: http://www.w3.org/TR/CSS1/

[39] Richard Leggett, Weyert de Boer, Scott Janousek. 2007. Foundation Flash Application for Mobile Devices, Macromedia and adobe flash: An overview.
ISBN: 978-1-4302-0308-7

[40] Debra Smarkusky, Sharon Toman, 2009, An interdisciplinary approach in applying fundamental concepts
ISBN: 978-1-60558-765-3

[41] Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5 (Adobe Featured Blogs), [verkkodokumentti]. Päivitetty: 2011 [viitattu: 25.06.2012]
Saatavissa: http://blogs.adobe.com/conversations/2011/11/flash-focus.html

[42] PHP – What is PHP? - Manual. [verkkodokumentti].
Päivitetty: 08.06.2012, [viitattu: 10.06.2012]
Saatavissa: http://php.net/manual/en/intro-whatis.php

[43] PHP – What can PHP do? - Manual. [verkkodokumentti].
Päivitetty: 08.06.2012, [viitattu: 10.06.2012]
Saatavissa: http://www.php.net/manual/en/intro-whatcando.php

[44] RFC 3875 - The Common Gateway Interface (CGI) Version 1.1 [verkkodokumentti]. Päivitetty: 2004, [viitattu: 10.06.2012]
Saatavissa: http://tools.ietf.org/html/rfc3875

[45] [Chapter 1] The Common Gateway Interface (CGI). [verkkodokumentti].
Päivitetty: 1996, [viitattu: 13.06.2012]
Saatavissa: http://oreilly.com/openbook/cgi/ch01_01.html

[46] CGI: The Common Gateway Interface. [verkkodokumentti].
Päivitetty: (ei tietoa), [viitattu: 13.06.2012]
Saatavissa: http://webdesign.about.com/od/cgi/a/aa021599.htm

[47] Socket.IO: the cross-browser WebSocket for realtime apps.

[verkkodokumentti]. Päivitetty (ei tietoa), [viitattu 06.12.2011].
Saatavissa: socket.io/#browser-support

[48] LearnBoost/socket.io-spec – GitHub [verkkodokumentti].
Päivitetty 23.05.2011, [viitattu 06.12.2011].
Saatavissa: https://github.com/LearnBoost/socket.io-spec

[49] S. Tilkov, S. Vinoski. 2010. Internet Computing, IEEE. Node.js: Using JavaScript to Build High-Performance Network Programs.
ISSN: 1089-7801 s. 80-83

[50] jWebSocket Cross-Browser-Compatibility [verkkodokumentti].

Päivitetty: 23.06.2010 [viitattu 12.10.2012]

Saatavissa: http://jwebsocket.org/jws_for_all.htm

[51] Bijin Chen, Zhiqi Xu. 2011. Multimedia Technology (ICMT),
2011 International Conference on. A framework for browser-based Multiplayer Online Games using WebGL and WebSocket
ISBN: 978-1-61284-771-9, s. 471 - 474

[52] The XMLHttpRequest Object, [verkkodokumentti].

Päivitetty: (ei tietoa) [Viitattu 1.10.2012]

Saatavissa: http://www.w3schools.com/xml/xml_http.asp

[53] XMLHttpRequest Articles Mozilla Hacks – the Web developer blog.
[verkkodokumentti]. Päivitetty: 06.05.2009, [viitattu 20.12.2011]
Saatavissa:
http://hacks.mozilla.org/category/xmlhttprequest/by/comments/as/complete/

[54] Ajax Tutorial, [verkkodokumentti].

Päivitetty: (ei tietoa) [viitattu: 1.10.2012]

Saatavissa:

http://code.google.com/intl/fi-FI/edu/ajax/tutorials/ajax-tutorial.html

[55] Faye: Simple pub/sub messaging for the web. [verkkodokumentti].

Päivitetty (ei tietoa), [viitattu 07.12.2011].
Saatavissa: http://faye.jcoglan.com/

[56] Comet Daily - Comet Maturity Guide, [verkkodokumentti].
Päivitetty: 2009 [viitattu 1.10.2012]
Saatavissa: http://cometdaily.com/maturity.html

[57] jWebSocket – The Open Source Java WebSocket Server. [verkkodokumentti].
Päivitetty: 18.11.2010, [viitattu 28.12.2011]
Saatavissa: http://jwebsocket.org/index.htm?page=browsers.htm

[58] AJAX Browser Support [verkkodokumentti].
Päivitetty (ei tietoa), [viitattu 20.12.2011]
Saatavissa: http://www.tutorialspoint.com/ajax/ajax_browser_support.htm

[59] Browser Support, [verkkodokumentti].
Päivitetty: 06.04.2011 [viitattu 1.10.2012]
Saatavissa:
https://groups.google.com/forum/?fromgroups#!topic/faye-users/YiffXm23Rd4

[60] JavaScript getTime() Method. [verkkodokumentti].
Päivitettty: (ei tiedossa), [viitattu 09.01.2012]
Saatavissa: http://www.w3schools.com/jsref/jsref_gettime.asp

[61] John Resig – Accuracy of JavaScript Time. [verkkodokumentti].
Päivitetty: 12.12.2008, [viitattu 09.01.2012]
Saatavissa: http://ejohn.org/blog/accuracy-of-javascript-time/

[62] System (Java Platform SE 6). [verkkodokumentti].
Päivitetty: (ei tietoa), [viitattu 21.01.2012]
Saatavissa: http://docs.oracle.com/javase/6/docs/api/java/lang/System.html

[63] Inside the Hotspot VM: Clocks, Timers and Scheduling Events – Part I – Windows (David Holmes' Weblog). [verkkodokumentti].

Päivitetty: 02.10.2006, [viitattu 21.01.2012]

Saatavissa:
http://blogs.oracle.com/dholmes/entry/inside_the_hotspot_vm_clocks

[64] Node.js and socket.io – problems in Opera Browser – Stack Overflow

[verkkodokumentti]. Päivitetty: 2011, [viitattu: 9.3.2012]

Saatavissa: http://stackoverflow.com/questions/7858173/node-js-and-socket-io-problems-in-opera-browser

[65] Opera 11 and issues with jsonp and xhr polling – Socket.IO | Google ryhmät

[verkkodokumentti]. Päivitetty: 2011, [viitattu: 9.3.2012]

Saatavissa: http://groups.google.com/group/socket_io/browse_thread/thread/014862015978539a

[66] The problem with transport in the Opera browser – Socket.IO | Google-ryhmät

[verkkodokumentti]. Päivitetty: 10.01.2012, [viitattu: 9.3.2012]

Saatavissa: http://groups.google.com/group/socket_io/browse_thread/thread/bbc4922132cd4808#

[67] javascript – socket.send() doesn't work on Opera 11.10 – Stack Overflow

[verkkodokumentti]. Päivitetty: 2011, [viitattu: 9.3.2012]

Saatavissa: http://stackoverflow.com/questions/6239927/socket-send-doesnt-work-on-opera-11-10

[68] Issue #9: tornadio2 doesn't work with Opera (v11.52)? - MrJoes/tornadio2 - GitHub. [verkkodokumentti]. Päivitetty: 2011, [viitattu: 9.3.2012]

Saatavissa: https://github.com/MrJoes/tornadio2/issues/9

[69] JavaScript sort() Method. [verkkodokumentti]. Päivitetty: (ei tietoa),

[viitattu: 9.3.2012]

Saatavissa: http://www.w3schools.com/jsref/jsref_sort.asp

[70] sort – MDN. [verkkodokumentti]. Päivitetty: 06.05.2011, [viitattu: 9.3.2012]

Saatavissa: https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/sort

[71] Array.sort() should not be used to shuffle an array | Lambda .

[verkkodokumentti]. Päivitetty: 16.11.2009, [viitattu: 4.3.2012]

Saatavissa: http://sroucheray.org/blog/2009/11/array-sort-should-not-be-used-to-shuffle-an-array/

[72] 2.4.6 HTML5 Canvas Start and Stop an Animation. [verkkodokumentti].

Päivitetty: 28.11.2011, [viitattu 21.02.2012]

Saatavissa: http://www.html5canvastutorials.com/advanced/html5-canvas-start-and-stop-an-animation/

[73] requestAnimationFrame for smart animating – Paul Irish.

[verkkodokumentti]. Päivitetty: 22.02.2011, [viitattu 22.02.2012]

Saatavissa: http://paulirish.com/2011/requestanimationframe-for-smart-animating/

[74] Erik Möller – requestAnimationFrame for smart(er) animating.

[verkkodokumentti]. Päivitetty: 20.12.2011, [viitattu 21.02.2012]

Saatavissa: http://my.opera.com/emoller/blog/2011/12/20/requestanimationframe-for-smart-er-animating

[75] HTML5 Rocks – Improving HTML5 Canvas Performance.
[verkkodokumentti]. Päivitetty: 16.05.2011, [viitattu 21.02.2012]

Saatavissa: http://www.html5rocks.com/en/tutorials/canvas/performance/

[76] Core HTML5 Canvas: Graphivs, Animation, and Game Development>

5.5 Double Buffering >5.5 Double Buffering – Pg. 375: Safari Books Online

[verkkodokumentti]. Päivitetty: (ei tietoa), [viitattu: 7.3.2012]

Saatavissa: http://my.safaribooksonline.com/book/illustration-and-graphics/9780132761635/chapter-5dot-animation/375

[77] jQuery Blog >> jQuery 1.7 Released. [verkkodokumentti]
Päivitetty: 03.11.2011, [viitattu 26.02.2012]
Saatavissa: http://blog.jquery.com/2011/11/03/jquery-1-7-released/

[78] jQuery Blog >> jQuery 1.7 Beta 1 Release. [verkkodokumentti]
Päivitetty: 28.09.2011, [viitattu 26.02.2012]
Saatavissa: http://blog.jquery.com/2011/09/28/jquery-1-7-beta-1-released/

[79] .bind() - jQuery API. [verkkodokumentti]. Päivitetty: (ei tietoa),
[viitattu: 26.02.2012]
Saatavissa: http://api.jquery.com/bind/

[80] jQuery Equivalent of Prototype Function.bind (HTML, CSS and JavaScript
forum at JavaRanch. [verkkodokumentti]. Päivitetty: 2001, [viitattu 6.3.2012]
Saatavissa: http://www.coderanch.com/t/477363/HTML-CSS-JavaScript/jQuery-Equivalent-Prototype-Function-bind