Petri Partanen

HTML5 monialustaisessa pelikehityksessä: kaksiulotteiset minipelit Phaser-pelimoottorilla

Tietotekniikan pro gradu -tutkielma

15. toukokuuta 2015

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Petri Partanen

Yhteystiedot: petri.m.partanen@student.jyu.fi

Ohjaajat: Ville Isomöttönen ja Jukka Varsaluoma

Työn nimi: HTML5 monialustaisessa pelikehityksessä: kaksiulotteiset minipelit Phaser-pelimoottorilla

Title in English: Multi-Platform Game Development with HTML5: two-dimensional minigames with the Phaser game engine

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 102+5

Tiivistelmä: Mobiililaitteiden yleistyessä videopelien pelaamiseen on tarve pelien helpommalle kehitykselle ja levitykselle kasvanut. HTML5 on avoin web-standardi, joka mahdollistaa monialustaisen multimediasisällön näyttämisen suoraan Internet-selaimissa. HTML5:n avulla voikin olla mahdollista luoda pelejä, jotka eivät kärsi natiiviohjelmien tavoin yhteensopivuus- tai asennusongelmista. On kuitenkin oletettavaa, ettei HTML5 ja siihen sidoksissa olevat tekniikat tai sitä tukevat Internet-selaimet ole vielä täysin valmiita kaikkiin web-teknologian asettamiin haasteisiin, mutta sitä voitaneen kaikesta huolimatta jo käyttää täysin pelattavien ajanvietepelien toteuttamiseen. Tämä tutkielma tutkii monialustaisen pelikehityksen tekniikoita, web-pohjaisten pelien tekoa HTML5:llä ja avoimen lähdekoodin HTML5-pelimoottoreita sekä työkaluja. Tutkielmassa toteutetaan kaksi HTML5-peliä avointen ohjelmistokirjastojen avulla. Näiden toteutettujen pelien kehitystä, suorituskykyä ja yhteensopivuutta eri laitteiden välillä testataan ja arvioidaan. Tutkimuksen tulosten mukaan mobiililaitteet kärsivät vielä osin yhteensopivuus- ja suorituskykyongelmista, mutta kannettavat ja pöytätietokoneet vaikuttivat jo toimivilta pelialustoilta tutkielmassa toteutetuille kaksiulotteisille HTML5-peleille.

Avainsanat: alustariippumattomuus, HTML5, JavaScript, monialustainen, pelikehitys, pelimoottori, pelit, Phaser, Pixi.js, Sisunmaan Sankarit, WebGL

Abstract: As mobile devices are becoming more and more popular for the casual gaming, there is an increasing need for achieving easier distribution and development for the games on these platforms. HTML5 is an open web standard that should provide platform independent multimedia content directly on any modern web browser. Whereas native applications may often suffer from issues with the portability and installation difficulties, the games taking advantage of HTML5 may have some advantages over them. However, due to it's immaturity this new technology is bound to have problems and an urge for improvement, but it may already have what it takes to create perfectly playable web games. This thesis studies the multi-platform game development, the benefits and pitfalls of the modern browser technologies and the tools for the web-based games. The resulting knowledge will be used to create two HTML5 games with the aid of libraries from the open community. The results of the study show that the mobile devices still suffer from compability and performance issues, but at least the laptop and desktop computers seem to be capable of running the created two-dimensional HTML5 games.

Keywords: cross-platform, game development, game engine, games, HTML5, JavaScript, multi-platform, Phaser, Pixi.js, Sisunmaan Sankarit, WebGL

Termiluettelo

A.I.	Tekoäly (*Artificial Intelligence*).
Ajax	JavaScript-kirjasto asynkronisten HTTP-pyyntöjen tekemiseksi (*Asynchronous JavaScript And XML*).
API	Ohjelmointirajapinta (*Application Programming Interface*).
Asynkroninen	Ajallisesti riippumaton kommunikointitapa osapuolten välille.
Bittikarttakuva	Pikseleistä koostettu digitaalinen kuva (engl. *bitmap image*).
CPU	Tietokoneen suoritin (*Central Processing Unit*).
CSS	WWW-sivujen tyyliohjeistus (*Cascading Style Sheets*).
DOM	Rakenteinen puumalli HTML-dokumentista (*Document Object Model*).
FPS	Ensimmäisen persoonan ammuntapeli (*First Person Shooter*).
Full HD	1920x1080 pikselin tarkkuuteen kykenevä täysteräväpiirto.
GPU	Grafiikkaprosessori (*Graphics Processing Unit*).
GUI	Graafinen käyttöliittymä (*Graphical User Interface*).
HTML	WWW-sivujen merkintäkieli (*HyperText Markup Language*).
HTTP	WWW-sivujen siirtoprotokolla (*HyperText Transfer Protocol*).
IDE	Ohjelmointiympäristö (*Integrated Development Environment*).
Isometrinen	Kuvakulma jossa kamera kuvaa pelimaailman viistosti sekä ylä- että sivusuunnassa.
JavaScript	Web-selaimissa toimiva dynaaminen ohjelmointikieli.
JPEG	Häviöllistä pakkausta käyttävä bittikarttakuva (*Joint Photographic Experts Group*).
JSON	Avoin kuvauskieli jäsennetyn tiedon välittämiseen (*JavaScript Object Notation*).
Metodi	Olion sisältämä funktio, joka voi muuttaa olion sisältämiä tietoja.
Minipeli	Pieni videopeli suuremman ja monipuolisemman pelikokonaisuuden sisällä (engl. *minigame*).
MP3	MPEG-1 -standardiin perustuva äänenpakkausmenetelmä.

NFC	Radiotaajuinen etätunnistustekniikka hyvin lyhyen kantaman tiedonsiirrolle (*Near Field Communication*).
OGG	Avoimen standardin äänenpakkausmenetelmä.
Partikkeliefekti	Tekniikka, jolla voidaan luoda ja hallita suuria määriä yksittäisiä objekteja esim. kipinöiden mallintamiseksi.
Pelinkehitysalusta	Graafinen työkalu pelien luontiin, joka sisältää pelimoottorin sekä muut tarvittavat ohjelmistokirjastot helppoon ja nopeaan pelikehitykseen.
PNG	Häviötön bittikarttagrafiikan tallennusformaatti, joka tukee läpinäkyvyyttä (*Portable Network Graphics*).
SDK	Ohjelmistokehityspaketti (*Software Development Kit*).
Shareware	Tietyin ehdoin maksuttomasti käytettävissä ja levitettävissä oleva sovellus.
Shell	Käyttöjärjestelmän komentotulkki.
Sprite	Videopeleissä käytettävä läpinäkyviä alueita sisältävä kuva.
Spritesheet	Kuvakollaasi, joka sisältää monta yksittäistä sprite-kuvaa.
SVG	Vektorikuvien kuvauskieli (*Scalable Vector Graphics*).
Tilemap	Pienistä ruuduista koostettu suurempi graafinen kokonaisuus, jonka avulla voidaan säästää muistia sekä prosessointiaikaa.
Tween	Peliobjektin ominaisuuksien arvojen muuntaminen matemaattisesti interpoloimalla.
WebCL	OpenCL-rajapinta JavaScriptillä toteutettavalle rinnakkaislaskennalle.
WebGL	JavaScriptin laitteistokiihdytetty OpenGL-rajapinta selainsovelluksille (*Web Graphics Library*).
WWW	Internet-verkon hypertekstijärjestelmä (*World Wide Web*).
XML	Kuvauskieli jäsennetyn tiedon välittämiseen (*Extensible Markup Language*).

Kuviot

Taulukot

Sisältö

1 Johdanto

Pelikehityksen yksi suurista haasteista tänä päivänä on erilaisten laitteistojen ja käyttöjärjestelmien laaja kirjo. Jotta peleillä olisi mahdollista saavuttaa mahdollisimman suuri yleisö, on kehittäjien yleensä huomioitava monet eri käyttöjärjestelmät ja laitteet. Kehittäminen eri julkaisualustoille vaatii kuitenkin lisätyötä, sillä järjestelmät ovat harvoin suoraan yhteensopivia toistensa kanssa (Bouras, Papazois ja Stasinos 2014, s. 3–5). Esimerkiksi mobiililaitteet – erityisesti älypuhelimet – ovat viime vuosina yleistyneet nopeasti pienten ajanvietepelien pelaamiseen (Curran ja George 2012, s. 31–32). Mobiilikäyttäjät ovat kuitenkin jakautuneet eri laitteistoarkkitehtuurien välille, minkä takia pelit ja monet muut ohjelmat on usein sovitettava erikseen jokaista järjestelmäjulkaisua varten (Corral ym. 2011).

Pelintekijän näkökulmasta katsottuna olisi ihanteellista mikäli yksi ja sama pelitoteutus tukisi suoraan kaikkia laitteita, eikä erillistä sovitustyötä olisi tarvetta tehdä. Erilaisia alustariippumattomia kehitystyökaluja on jo olemassa (esim. *Unity*), mutta ne eivät vielä itsessään välttämättä ratkaise kaikkia ongelmia (Gawley, Barr ja Barr 2012, s. 188–189). Useimmille mobiililaitteille on omat ohjelmistokauppansa (nk. *app store*), josta käyttäjät voivat etsiä sekä asentaa kaikenlaisia sovelluksia, kuten pelejä. Tämä on osaltaan helpottanut merkittävästi ohjelmien markkinointia ja käyttöä, mutta pelien julkaisu sovelluskauppojen kautta vaatii kuitenkin usein saatujen tuottojen jakamista näiden palveluntarjoajien kanssa (Juntunen, Jalonen ja Luukkainen 2013). Olisikin siten myös toivottavaa, että pelejä olisi mahdollista markkinoida ja pelata suoraan esimerkiksi kehittäjien itsensä hallitsemien sivustojen kautta. Käyttäjä taas hyötyisi jos pelejä ei tarvitsisi lainkaan asentaa pelaamista varten, vaan pelit olisivat välittömästi saatavilla.

HTML5 on WWW-sivustojen merkintäkielen uusin standardi, joka mahdollistaa multimediasisällön näyttämisen suoraan Internet-selaimissa. Se on siten myös avannut mahdollisuuden luoda pelejä suoraan selaimiin ilman tarvetta erillisille selainliitännäisille (Curran ja George 2012, s. 39). Ottaen huomioon tekniikan varhaisuuden on oletettavaa, että se on pelikehityksen kannalta kuitenkin vielä suhteellisen keskeneräinen. HTML5:llä saavutettavat hyödyt voivat joka tapauksessa olla jo niin merkittävät, että se kannattaa ottaa ainakin yhdeksi vakavaksi vaihtoehdoksi pelinkehitykseen.

Jyväskylän yliopiston monitieteisen tutkimusyksikön Agora Centerin tutkimushankkeessa vuosina 2014–2016 kehitetään yhteistyössä Lapsiasiavaltuutetun toimiston kanssa lasten oikeuksista kertova pelikokonaisuus, jonka vaatimuksiin kuuluvat pelin helppo saavutettavuus sekä toimivuus mahdollisimman monilla eri laitteilla. Tämä pelikokonaisuus käsittää useamman pelityypiltään erilaisen *minipelin*, jotka edustavat perinteistä 2D-grafiikkaa ja toiminnallisuutta. Selainpohjaisen ratkaisun katsottiin olevan varteenotettava toteutustekniikka, ja tämän tutkielman päätavoitteena on selvittää HTML5:n soveltuvuutta nykymuodossaan kahden tällaisen monialustaisen pelin toteutuksessa. Tavoitteena on erityisesti saavuttaa saumaton yhteensopivuus pöytätietokoneiden ja mobiililaitteiden välillä.

Koska pelinkehitystä ei nykypäivänä ole enää mielekästä aloittaa täysin ns. juuritasolta, rajoittaudutaan tässä tutkimuksessa hyödyntämään erityisesti jo olemassa olevia avoimen lähdekoodin ohjelmistokirjastoja, kuten pelimoottoreita. Tutkielman tavoitteena ei siis ole kehittää uutta HTML5-pelimoottoria tai testata vain yksittäisten ominaisuuksien soveltuvuutta peleihin, vaan kartoittaa ja arvioida koko tekniikan varteenotettavuutta pienten pelien kehityksen näkökulmasta yllä mainitun projektin tarpeisiin.

Tutkielma on luonteeltaan konstruktiivinen. Tutkielman alussa luodaan kirjallisuuskatsaus monialustaisiin peliteknologioihin, selainpeleihin ja niiden markkinoihin sekä tehdään selvitystä HTML5:stä ja sen nykytilasta. Tämän jälkeen esitellään kahden HTML5-pelin suunnittelu ja toteutus. Lopuksi näitä kehitettyjä pelejä testataan erityyppisillä laitteilla ja niiden suorituskykyä sekä yhteensopivuutta arvioidaan.

Tutkielma koostuu kahdeksasta luvusta. Luvussa 2 tutustutaan nykypäivän pelikehityksen haasteisiin ja pelien sekä pelaamisen muutoksiin. Tämän lisäksi pohditaan monialustaisen pelikehityksen tuomia ongelmia, hyötyjä ja vaikutuksia pelien julkaisun kannalta. Luvussa 3 perehdytään monialustaisen pelikehityksen mahdollistaviin tekniikoihin. Luvussa 4 esitellään HTML5, siihen sidoksissa olevat tekniikat sekä tehdään katsaus yleisesti saatavilla oleviin avoimen lähdekoodin HTML5-pelimoottoreihin ja niihin liittyviin työkaluihin. Luvussa 5 käydään läpi tutkimuksen toteutus. Luvussa 6 esitellään kahden HTML5-pelisovelluksen suunnittelua, yleisiä vaatimuksia ja toteutusta. Luvussa 7 suoritetaan toteutettujen pelien arviointia sekä tarkastellaan tuloksia pelien testauksesta. Lopuksi luvussa 8 annetaan johtopäätökset tutkielmassa läpikäydyistä asioista.

2 Pelikehityksen muutokset ja haasteet

Videopelaaminen ja pelien kehittäminen ovat tekniikan kehityksen myötä muuttuneet merkittävästi viime vuosikymmeninä. Siinä missä 1980-luvun alkupuolella pelejä tehtailtiin vielä melkeinpä ”autotalleissa”, dominoivat suuret pelijulkaisut puolestaan lähes yksinoikeudella 1990-luvun pelimarkkinoita. Tuolloin nk. *shareware*-kulttuuri olikin aktiivisimmillaan, ja monille peleille se saattoi olla jopa ainoa mahdollinen levityskeino. 2000-luvun vaihteen digitaalisen lataamisen yleistyminen muutti kuitenkin niin pelien levittämistä, kuin itse pelaamistakin. Menestyvän pelin ei tarvinnut enää välttämättä olla graafisen innovaation edelläkävijä, sillä yksinkertaiset ajanvietepelit ovat nousseet suosituimmiksi kuin koskaan (Curran ja George 2012, s. 31–32). Nykypäivän pelinkehittäjällä voi olla edessään potentiaalinen miljoonayleisö, mutta pelilaitteiden suuri valikoima ja eri julkaisutavat monimutkaistavat tilannetta huomattavasti.

Tässä luvussa kartoitetaan yleisesti nykypelaamista, luodaan katsaus pelimarkkinoiden suuntauksiin sekä pohditaan pelinkehityksen haasteita monialustavaatimusten pohjalta. Varsinaisiin monialustajulkaisun mahdollistaviin tekniikoihin tutustutaan myöhemmin luvussa 3.

2.1 Ajanvietepelaamisen herääminen

Ajanvietepelien tyylilajin voinee osittain päätellä sen nimestä, mutta määritelmä itsessään ei ole täysin yksikäsitteinen. Ajanvietepelien tyypillisiksi piirteiksi luokitellaan usein mm. lyhyet pelisprintit, helppo opittavuus ja nopea pelissä edistyminen (Mäyrä 2015, s. 323–324). Tämän vastakohtana voi siis helposti nähdä viime vuosikymmenien niin sanotun perinteisen PC-pelin, jonka pelaaminen vaatii usein paljon oppimista ja myös aikaa. Toisaalta taas ajanvietepeleihin luetaan usein kuitenkin sellaiset pelaamisen todelliset klassikot, kuten *Pong* (1972) ja *Pacman* (1980) (Kultima 2015). Juul (2010) pitääkin nykypäivän ajanvietepelejä eräänlaisena videopelaamisen paluuna alkuperäisille juurilleen, sillä hänen mukaansa tyypillinen nk. ”hardcore”-pelaaminen edustaa vain pientä osuutta koko videopelaamisesta.

Internetin yleistyminen 2000-luvun vaihteessa vauhditti merkittävästi videopelaamisen yleistymistä ja tuolloin nähtiin myös ensimmäiset varsinaiset suoraan selaimessa pelattavat pe-

lit (Mäyrä 2015, s. 322). Varsinkin pienille ajanvietepeleille tämä merkitsi aivan uutta aikakautta, sillä erityisesti ilmaiset ja helposti saavutettavissa olevat pelit osoittautuivat suosituiksi myös niiden ihmisten keskuudessa, jotka eivät olleet aiemmin videopelejä pelanneet. Ajanvietepelit ovatkin osaltaan luoneet pelimarkkinoille kokonaan uusia pelaajia ja pelaamisesta on tullut hiljalleen yhä tavanomaisempaa (Kultima 2015, s. 5). Tätä muutosta voidaan myös pitää eräänlaisena pelaamisen käännekohtana, sillä videopelit miellettiin ennen vain lähinnä juuri hardcore-pelaajien harrastukseksi (Juul 2010, s. 8–10).

Vuonna 2013 jo pelkästään selaimessa pelattavat ajanvietepelit kattoivat pelimarkkinoista noin 10 % (Newzoo 2015). Esimerkiksi *Facebook* on julkaissut oman rajapintansa pelien kehittämisen mahdollistamiseksi ja sen kautta pelejä pelaavat kuukausittain jo sadat miljoonat pelaajat (Mäyrä 2015). Sekä Juul (2010) että Mäyrä (2015) pitävät sosiaalisen median vaikutusta tärkeänä ajanvietepelien yleistymiselle. Syynä tähän voi olla pelien näkyvyys ja helppo saavutettavuus sivustojen kautta. On kuitenkin myös pelejä, jotka itsessään sisältävät jonkin sosiaalisen elementin ja tuovat näin peleihin täysin uudenlaisen moninpelillisen vuorovaikutuksen (Juul 2010, s. 150–151). Pelien sosiaalisuus voikin olla yksi syy monien ajanvietepelien suosiolle (Di Loreto ja Gouaïch 2010, s. 3). Käytetyn tekniikan kannalta tämä joka tapauksessa vaatii pelien sitomista entistä tiukemmin verkkoon.

Yleisesti ajanvietepelien suosio tarkoittaa myös sitä, että vastaavasti pelinkehittäjien on mahdollista menestyä ja saavuttaa huomattava pelaajamäärä ilman merkittävää rahallista panosta. Viime vuosikymmeninä ”perinteisten” pelien kehityskustannukset ovat kasvaneet valtavasti ja hittipelin vaatima budjetti voikin helposti nousta jo miljooniin dollareihin (Juul 2010, s. 148). Ajanvietepelien kehitys taas ei välttämättä vaadi uusimpien tekniikoiden implementointia tai valtavan sisällöllisen laajuuden luomista, vaan jo pelin koukuttavuus itsessään voi ratkaista, kuinka suosittu pelistä voi lopulta tulla (Evans 2015, s. 4). Ajanvietepelien yksi tärkeimmistä vaatimuksista on niiden saavutettavuus, sillä pelaajat eivät ole enää välttämättä valmiita tekemään lisätyötä pelien asentamiseksi (Kultima 2015, s. 2).

2.2 Mobiilipelien aikakausi

Ajanvietepelien lisäksi yksi pelimarkkinoiden mullistuksista on ollut älypuhelinten ja muiden mobiililaitteiden huima kehitys 2010-luvulle siirryttäessä. Laitteiden jatkuva läsnäolo on tuonut pelaamiselle osaltaan kokonaan uuden merkityksen, ja pelien helppo asennettavuus sovelluskauppojen kautta on puolestaan luonut pelienkehittäjille markkinointimahdollisuuksia, jollaisia ei ollut aiemmin olemassakaan. Tämä onkin mahdollistanut myös pienten pelienkehittäjien menestymisen täysin uudenlaisten ansaintamallien ansiosta (Feijoo ym. 2012, s. 216). Laitekohtaisista sovelluskaupoista käyttäjät voivat etsiä, ladata ja ostaa sovelluksia monille eri laitteille (Curran ym. 2015, s. 262).

Sekä pelit että itse pelaaminen ovat viime vuosina yleistyneet mobiililaitteilla merkittävästi. Vaikka ilmiö on suhteellisen uusi, on sen vaikutus ollut huomattava, sillä mobiilipelien arvioidaan vuonna 2015 vievän kaikkiaan noin kolmasosan koko maailman pelimarkkinoista (Gaudiosi 2015). Analyysien mukaan vuonna 2013 älypuhelimet kattoivat jo markkinoista noin 17 % ja tablet-laitteet 6 % (Newzoo 2015). Myös Suomessa mobiilipelaaminen on kasvanut viime vuosina selvästi, sillä vuonna 2013 jo lähes joka kolmas suomalainen pelasi aktiivisesti mobiilipelejä (Mäyrä ja Ermi 2014).

Mobiilipelien aikakauden voidaan Feijoo ym. (2012, s. 213–216) mukaan katsoa alunperin alkaneen ensimmäisen *iPhonen* julkaisun myötä vuonna 2007. Tähän on heidän mukaansa lisäksi vaikuttaneet merkittävästi myös vuonna 2008 avatut Applen ja Googlen sovelluskaupat, jotka tekivät pelien asentamisesta ja markkinoinnista mobiililaitteille huomattavasti aiempaa yksinkertaisempaa. Lisäksi osasyynä suosiolle voitaneen heidän mukaansa pitää myös kehittynyttä laiteteknologiaa, joka on mahdollistanut monimutkaisempien ja näyttävämpien pelien luomisen myös mobiililaitteille.

Huolimatta mobiilipelaamisen erittäin menestyksekkäästä alkutaipaleesta, ei ole silti edelleenkään syytä unohtaa muita pelialustoja. Esimerkiksi perinteisiä pöytätietokoneita käytetään yhä merkittävissä määrin pelaamiseen, eikä mobiilipelien ennusteta varsinaisesti valtaavan konsoli- tai PC-pelimarkkinoita (Gaudiosi 2015). Nykyinen trendi kuvastaneekin lähinnä sitä, että nykypäivän pelaaminen on siirtynyt myös olohuoneiden ulkopuolelle, mikä lienee erityisesti älypuhelinten ansiota. Suomessa tilanne on hieman päinvastainen, sillä muu

videopelaaminen on laskenut selvästi mobiilipelaamisen tieltä (Mäyrä ja Ermi 2014).

Käsitteenä mobiililaitteet kattavat hyvin laajan valikoiman erikokoisia ja suorituskykyisiä laitteita (Holzinger, Treitler ja Slany 2012, s. 179–182). Ruudun koko voi mobiililaitteilla olla käytännössä mitä vain kolmen ja kolmentoista tuuman väliltä. Mobiililaitteiden kategorisointi voikin tuottaa ongelmia, sillä esimerkiksi raja joidenkin kannettavien tietokoneiden ja tablettien välillä voi olla hyvin häilyvä. Mobiililaitteille on myös omat käyttöjärjestelmänsä, jotka eivät yleensä ole yhteensopivia muiden vastaavien järjestelmien kanssa. Pelinkehittäjän on kuitenkin huomioitava useampia laitejärjestelmiä, mikäli pelin halutaan saavuttavan vähintään puolet mobiilikäyttäjistä. Tämä tarkoittaa sitä, että pelin tulee saumattomasti toimia erikokoisilla ruuduilla, syötelaitteilla ja käyttöjärjestelmillä.

Mobiililaitteiden osalta käytettyjen käyttöjärjestelmien määrä on vuosien varrella selkeytynyt, sillä myytyjen uusien laitteiden käyttöjärjestelmät ovat tänä päivänä keskittyneet pääasiassa *Androidin* ja *iOS:n* välille (IDC 2015). Tilanne voi kuitenkin nopeasti muuttua, sillä mobiilimarkkinat elävät tunnetusti hyvin lyhyellä aikajänteellä. Charaf ym. (2014, s. 530–531) mukaan mobiililaitteille sovelluksiaan kehittävät kohdistavat ohjelmansa keskimäärin noin kolmelle eri laitejärjestelmälle. Vielä muutama vuosi sitten tilanne oli kehittäjien kannalta hieman haastavampi, sillä esimerkiksi *BlackBerry OS* ja *Symbian* olivat käyttäjämäärät huomioiden selvästi ohjelmien varteenotettavia julkaisualustoja (Heitkötter, Hanschke ja Majchrzak 2013). Symbian oli pitkään mobiilipelaamisen markkinajohtaja, mutta ei kyennyt kuitenkaan vastaamaan kilpailijoidensa kehitykseen. Lopulta Nokian puhelinliiketoiminnan siirtyminen Microsoftille vuonna 2014 merkitsi Symbian-puhelinten poistumista markkinoilta kokonaan. Lisäksi kaikista odotuksista huolimatta Microsoft ei ole onnistunut kasvattamaan *Windows Phonen* markkinaosuuttaan käytännössä juuri lainkaan (IDC 2015).

2.3 Ansaintamallit

Sovelluskaupat ja selaimessa pelattavat pelit ovat muuttaneet pelien ansaintalogiikkoja siten, etteivät pelaajat välttämättä enää maksa varsinaisesta pelistä, vaan itse pelin sisällöstä. Tästä on tullut hyvin yleinen tapa ja sitä käyttävät nykypäivänä erityisesti ajanvietepelit (Kultima 2015, s. 4). Esimerkiksi sosiaalisessa mediassa hyvin suosittu *FarmVille* on pelaajille käytän-

nössä ilmainen, mutta pelaajan on kuitenkin mahdollista ostaa lisätoiminnallisuuksia oikealla rahalla (Mäyrä 2015, s. 330-331). Tällaista maksumallia kutsutaan *mikromaksamiseksi* ja on täysin pelaajan vastuulla, haluaako hän hyödyntää tätä vai ei. Ansaintamalli näissä tapauksissa perustuu pelaajan haluun edetä pelissä nopeammin tai saada pelin sisällä haltuunsa jotain, mikä ilman oikean rahan käyttöä ei olisi välttämättä mahdollista (Evans 2015, s. 3).

Evans (2015) on selvittänyt mikromaksamiseen perustuvien pelien ideologiaa ja kutsuu niitä *"freemium"*-peleiksi. Hänen mukaansa keskeinen idea tällaisissa peleissä on, etteivät ne käytännössä pääty koskaan, mikä voi pelillisestä näkökulmasta katsottuna olla ongelmallista. Evansin mukaan myös pelien sosiaalisuus on tärkeässä asemassa freemium-peleissä, sillä ystävien ja muiden pelaajien mukanaolo voi tuoda peliin kilpailullisen elementin, joka saa pelaajat ostamaan lisää pelin sisältöä.

Osa peleistä noudattaa yhä perinteisempää ansaintamallia, jossa pelaaja edelleen ostaa pelin itselleen. Tällöin pelin hinta voi olla kuitenkin vain muutaman euron, mikä verrattuna perinteiseen "kymmenien eurojen hintaan" alentaa merkittävästi käyttäjän ostokynnystä. Tähän on mahdollista yhdistää myös kokeiluperiaate, jossa pelaaja voi pelata esimerkiksi 60 minuuttia ilmaiseksi, jonka jälkeen jatkamisesta on maksettava (Juul 2010, s. 77). Peleihin voidaan liittää myös mainoksia, joka on varsinkin Android-laitteilla hyvin yleinen menettely. Mainokset ovat joka tapauksessa ongelmallisia, sillä niitä voi olla vaikea sijoitella, eivätkä pelaajat pidä niistä (Marszałkowski 2011). Eri sovelluskauppojen välillä ansaintamallit voivatkin vaihdella, ja ansaintamalleja usein myös yhdistellään. Selkeästi parasta vaihtoehtoa on vaikea löytää, sillä siihen vaikuttavat hyvin monet asiat, esimerkiksi juuri pelin julkaisukanavat.

Mobiililaitteiden osalta sovelluskaupoista on tullut merkittävin tapa julkaista ohjelmia. Sovellusten määrä on kasvanut vuosi vuodelta kiihtyvällä vauhdilla, ja esimerkiksi Googlen sovelluskaupassa oli yli miljoona sovellusta jo vuonna 2013 (Curran ym. 2015). Sovelluskauppojen ongelmaksi onkin muodostumassa helposti ylitarjonta, sillä pelien erottuminen massasta voi osoittautua lähes mahdottomaksi. Toinen ongelma pelinkehittäjän kannalta on hinta: sekä Apple että Google ottavat pelien tuloista 30 %:n osuuden itselleen (Feijoo ym. 2012, s. 216). Tämä voi monien pelien kohdalla herättää kysymyksiä siitä, ovatko sovelluskauppojen antamat hyödyt todella sen arvoisia.

Sovelluskaupat rajoittavat osaltaan pelien toteutusratkaisuja, sillä sekä Apple että Google tarkastavat sovelluskauppoihin lisätyt sovellukset ja asettavat omat ehtonsa ohjelman hyväksynnälle (Feijoo ym. 2012, s. 216). Esimerkiksi Applen sovelluskaupan hyväksymisehdot listaavat lähes kaksisataa erillistä kohtaa, joita sovelluksen tulee noudattaa. Hyväksymisehtojen mukaan sovellus ei saa sisältää esimerkiksi väkivaltaa, etnistä syrjintää, seksuaalista sisältöä, ylimääräisiä mainoksia tai *iTunes*-ohjelmaa muistuttavaa ulkonäköä (Apple 2015). Myös toteutukseen käytettyjä teknisiä ratkaisuja on rajoitettu monilta osin. Googlen asettamat sovelluskaupan ehdot ovat samankaltaisia, sillä App Storen tavoin mm. seksuaalista tai rasistista sisältöä ei sallita (Google 2015).

Yleisesti katsottuna Apple on suodattanut kauppaansa lisätyt sovellukset tarkemmin kuin Google (Holzer ja Ondrus 2011, s. 25). Kehittäjälle sovelluskaupan ehtojen monimutkaisuus voivat olla ikävä yllätys, sillä Ohrt ja Turau (2012, s. 73) toteavat, että mikäli Apple jostain syystä hylkää ohjelman sovelluskaupastaan, ei kyseistä ohjelmaa yksinkertaisesti voi enää levittää esimerkiksi iOS-laitteille lainkaan. Ohjelmien julkaisu sovelluskauppojen kautta voi myös kestää oman aikansa, sillä hyväksymisprosessi voi kokonaisuudessaan osoittautua kehittäjälle vaikeaksi (Curran ym. 2015, s. 264).

Vaihtoehdot pelinkehittäjille ovat varsinkin mobiililaitteiden tapauksessa vähissä, sillä pelin sijoittaminen omalle sivustolle ladattavaksi ei ole välttämättä enää nykypäivän pelaajalle vaihtoehtona. Pelaajat ovat tottuneet jo sovelluskauppojen yksinkertaisuuteen, eikä pelin erillinen asentaminen satunnaiselta sivustolta ole tietoturvankaan kannalta houkuttelevaa. Toisaalta taas pöytätietokoneiden osalta sovelluskaupat eivät ole vielä täysin onnistuneet lyömään itseään läpi, joten pelien julkaisu ja markkinat hakevat vielä parasta ratkaisumallia.

2.4 Pelinkehittäjän haasteet

Edellä esitetyt pelaamista käsittelevät luvut tekevät hyvin tiiviin silmäyksen siihen monimuotoisuuteen, mitä kaikkea nykypäivän pelaamiseen kuuluu. Pelaaminen itsessään on yleistynyt vuosi vuodelta valtavaa vauhtia, eikä pelaaminen ole enää nk. hardcore-pelaajien harrastus, sillä pelit kulkevat jo esimerkiksi älypuhelimiemme mukana kaikkialle. Voisi myös sanoa, että pelien kehittäminen puolestaan on palannut hiljalleen 1970–1980 -lukujen kulta-

vuosiin, aikaan jolloin pienetkin pelinkehittäjät pystyivät menestymään yksinkertaisesti vain hyvällä idealla. Monien ajanvietepelien suosio on jo osaltaan todistanut, että pelkästään riittävän koukuttavalla peli-idealla voidaan saavuttaa miljoonia pelaajia. Yleisesti katsottuna peleillä on tänä päivänä luultavasti mahdollista saavuttaa suurempi kohderyhmä kuin koskaan aiemmin.

Kaikella tällä on kääntöpuolensa, sillä nykypäivän pelinkehittäjä joutuu kohtaamaan hyvin kirjavan valikoiman eri laitteita ja käyttöjärjestelmiä, joista pelin potentiaaliset julkaisukohteet on valittava. Keskeisenä ongelma on, etteivät valtaosa näistä laitteista ole suoraan yhteensopivia toistensa kanssa (Heitkötter, Hanschke ja Majchrzak 2013, s. 120). Esimerkiksi perustyökaluilla pelin kehittäminen yhdelle mobiililaitteelle sulkee käytännössä mahdollisuuden julkaista sama peli muille laitteille. Vielä epätodennäköisemmin kyseinen toteutus saadaan toimimaan helposti myös pöytätietokoneilla. Curran ym. (2015, s. 265) epäilevät, että laitteiden yhteensopivuus voi vielä monimutkaistua entisestään uusien innovaatioiden myötä.

Sitoutuminen tietylle laitealustalle ei ole pelinkehittäjälle toivottavaa, sillä markkinoiden monimuotoisuus on jakanut pelaajat laajalle valikoimalle eri laitteita ja järjestelmiä. Pelinkehittäjille mielekkäintä olisikin varmasti käyttää laitealustoihin liittyvien yhteensopivuuksien ratkomiseen käytetty työmäärä mieluummin vaikkapa pelin sisällön tuottamiseen. Helpoin ratkaisu pelitoteutukselle olisi sellainen, joka mahdollistaisi pelin julkaisun samanaikaisesti kaikilla mahdollisilla järjestelmillä, eikä vaatisi laitekohtaista asiantuntemusta.

Kehitys- ja julkaisutavan valintaan vaikuttaa lisäksi pelien markkinointi, sillä sovelluskaupat ottavat myynneistä ison siivun itselleen, eikä potentiaalisestikaan hyvän pelin näkyvyys ja menestys ole niiden kautta välttämättä mikään itsestäänselvyys. Koska pelien kehityskustannukset ovat pienentyneet, on myös kilpailu pelaajista kiristynyt ja sovelluskauppoja hallitsevat usein suuret peliyhtiöt (Moreira, Filho ja Ramalho 2014). Curran ja George (2012, s. 265) toteavat, että menestyminen sovelluskaupoissa voi äärimmillään rajoittua vain TOP-listalla olevien pelien kesken. Pelin julkaiseminen samanaikaisesti monen eri sovelluskaupan kautta voi tuoda myös lisäkustannuksia, sillä niiden päivittäminen ja ylläpito vaikeutuvat tällöin huomattavasti (Bouras, Papazois ja Stasinos 2014).

Monialustaisuudella tarkoitetaan sellaisten alustariippumattomien ohjelmien kehitystä, jotka toimivat useilla eri laitteilla ja käyttöjärjestelmillä (Heitkötter, Hanschke ja Majchrzak 2013, s. 122). Corral ym. (2011, s. 183) toteavat mobiilijärjestelmiä valmistavien tahojen hyötyvän kuluttajien sitoutumisesta laitekeskeisiksi, eivätkä siksi välttämättä halua olla varsinaisesti edistämässä monialustaisen ohjelmistokehityksen yleistymistä. Gawley, Barr ja Barr (2012, s. 189) mukaan Apple rajoitti vuonna 2010 mobiilisovelluskauppaansa hyväksyttävien ohjelmien toteutusvaihtoehtoja siten, ettei esimerkiksi *Adobe Flash* -toteutuksia hyväksytty lainkaan. Vaikka tätä rajoitusta myöhemmin hieman helpotettiin, herättää tapaus heidän mukaansa kuitenkin kysymyksen siitä, onko mobiiliohjelmien tulevaisuus todella sidottu näihin sovelluskauppoihin, vai tuleeko jokin muu keino ohjelmien levittämiseen olemaan ennen pitkää parempi ratkaisu. Curran ym. (2015, s. 265) ovat toisaalta sitä mieltä, että ainakin mobiililaitteiden tapauksessa kaikki sovellukset tulevat sijaitsemaan jatkossakin yhä pääasiassa sovelluskaupoissa.

3 Monialustainen pelikehitys

Edellisessä luvussa käytiin läpi pelaamisen nykytilaa ja todettiin, että kohdistamalla pelit yksinomaan vain tietyille laitteille tai käyttöjärjestelmille, jätetään mahdollisesti suuri määrä potentiaalisia pelaajia kokonaan pelin saavuttamattomiin. Lisäksi perinteinen laitekeskeinen kehitystapa ei välttämättä sovi nykypäivän verkkopelaamisen vaatimuksiin. Pelinkehittäjien onkin resurssiensa puitteissa pyrittävä laajentamaan pelin kohdealustavaihtoehdot kattamaan useita eri järjestelmiä. Pelin saaminen toimimaan useilla eri alustoilla tarkoittaa pelin kehittämistä erityisillä työkaluilla, jotka mahdollistavat ohjelman helpon ajamisen/kääntämisen usealle eri laitteistolle sekä käyttöjärjestelmälle. Tässä luvussa käydään läpi eri laitetyyppien välisiä monialustatekniikoita, joista osa soveltuu juuri pelien kehitykseen.

3.1 Monialustaisuuden perusteet

Monialustainen sovelluskehitys on mahdollista saavuttaa tekniikoilla, jotka hyödyntävät esimerkiksi ohjelman lähdekoodin helppoa muuntamista toiselle järjestelmälle tai toimivat suoraan sellaisinaan eri laitteilla. Niin pelien kuin muidenkin ohjelmien kehittäjien näkökulmasta katsottuna ei laitteistojen mukanaan tuomilla eroilla ole usein ohjelman kannalta välttämättä merkitystä. Esimerkiksi kannettavat tietokoneet ovat toiminnaltaan hyvin identtisiä, vaikka niissä olisikin täysin eri käyttöjärjestelmä. Monialustatekniikoiden tulisikin pyrkiä hämärtämään rajat eri laitteiden välillä ja yhtenäistämään käyttäjän sekä sovelluksen välistä vuorovaikutusta käytetystä laitteesta riippumatta (Achilleos ja Kapitsaki 2014, s. 305).

Peleihin monialustatekniikat eivät ole aina suoraan soveltuvia, sillä videopelit asettavat osaltaan omat haasteensa ohjelmistokehitykselle. Pelit vaativat usein esimerkiksi mahdollisuuden vapaaseen grafiikan piirtoon, hyvän tuen syötelaitteille, monipuolisen tavan tuottaa ääntä sekä riittävän suorituskyvyn. Täydellisesti toimivaa monialustaratkaisua ei peleille ole vielä olemassa ja esimerkiksi pelin julkaisu asettaa oman rajansa käytettävälle tekniikalle. Luvussa 2.3 todettiin mm. sovelluskauppojen asettavan tietyt sääntönsä toteutustekniikoille, vaikkakaan ne eivät sellaisenaan sulje pois kaikkia monialustatekniikoita. Ohrt ja Turau (2012, s. 73) toteavat kuitenkin, että kehittäjälle turvallisinta on valita laitekohtainen kehitystyökalu,

jotta ohjelma suuremmalla todennäköisyydellä hyväksyttäisiin sovelluskauppaan.

Monialustainen ohjelmistokehitys on itsessään hyvin ajankohtainen aihe, sillä sen tarve on laitteiden monipuolistuessa kasvanut vuosi vuodelta. Monialustatekniikoista on siten olemassa myös laajasti lähdekirjallisuutta. Näyttää kuitenkin siltä, että valtaosa viime vuosien materiaalista on keskittynyt tarkastelemaan nimenomaisesti samankaltaisten laitteiden välistä monialustakehitystä. Erityisesti mobiililaitteet ovat monialustatutkimuksen keskeisimpiä mielenkiinnon kohteita. Sen sijaan puhtaasti erilaisten laitetyyppien välisten monialustatekniikoiden tutkimus näyttääkin nyt jäävän – ainakin osittain – mobiililaitteiden jalkoihin. Tästä johtuen aiheen käsittely myös tässä tutkielmassa on vahvasti mobiililaitteisiin painottunut.

3.2 Natiivisovellukset

Perinteinen tapa toteuttaa ohjelmistoja ovat niin sanotut *natiivisovellukset*, joita esimerkiksi sovelluskaupoista ladatut ohjelmat pääasiassa ovat (Heitkötter, Hanschke ja Majchrzak 2013, s. 122). Natiivisovelluksilla tarkoitetaan ohjelmiston kohdistamista yleensä vain tietylle käyttöjärjestelmälle tai laitteistolle, joka on käytännössä ohjelmakoodin kääntämistä toisin kuin tulkattavien kielten tapauksessa (Charland ja Leroux 2011). Tällä tavalla tiettyä alustaa varten kehitetty natiivisovellus ei sellaisenaan toimi suoraan muilla järjestelmillä. Esimerkiksi puhtaasti Windowsille kehitettyä natiivisovellusta ei voida pääsääntöisesti käyttää OS X -käyttöjärjestelmässä. Natiivisovelluksia pidetäänkin näin puhtaiden monialustasovellusten vastakohtana ja ne toimivat tässä tutkielmassa monialustaisuus-käsitteen perustana.

Natiivisovellus kehitetään usein juuri kyseiselle käyttöjärjestelmälle tarkoitetulla *SDK*:lla, tietyillä ohjelmointikielillä sekä muilla tarvittavilla työkaluilla (Bouras, Papazois ja Stasinos 2014). Varsinkin PC-koneet tukevat hyvin laajaa valikoimaa eri ohjelmointikieliä eikä natiivisovelluksen luominen ole varsinaisesti sidottu mihinkään kehitystyökaluun. Mobiililaitteilla osalta tilanne on huomattavasti rajoittuneempi, sillä esimerkiksi Android-laitteille natiivisovellukset kehitetään pääsääntöisesti *Javalla*, iOS-laitteille *Objective-C* tai *Swift* -kielillä ja Windows Phonelle *.NET*-työkaluilla (Charland ja Leroux 2011, s. 51).

Jotta natiiviohjelma on mahdollista saada toimimaan myös muilla järjestelmillä, on kehittäjien yleensä tehtävä sovitustyötä (engl. *porting*), joka voi käytännössä tarkoittaa esimerkiksi

jonkin lähdekoodin osan kokonaan uudelleenkirjoittamista (Corral ym. 2011, s. 181). Natiiviohjelmien yhtenä etuna on, että niillä on suora yhteys käytettyyn käyttöjärjestelmään ja laitteistoon (Charland ja Leroux 2011). Ne voivat näin esimerkiksi antaa graafisille vaatimuksille ja tarvittaville syötelaitteille paremman tuen. Nämä ohjelmat voivat myös olla suorituskyvyltään monialustatekniikoilla toteutettuja versioita tehokkaampia. Käyttäjän tulee joka tapauksessa tavalla tai toisella asentaa natiiviohjelma laitteelleen, ennen kuin sitä on mahdollista käyttää (Bouras, Papazois ja Stasinos 2014). Myös tietoturvan hallinta voi olla natiiviohjelman tapauksessa käyttäjän kannalta kriittinen kysymys, sillä natiivisovelluksilla on usein järjestelmissä laajat luku- ja suoritusoikeudet.

3.3 Monialustaratkaisut

Hartmann, Stead ja DeGani (2011) jakavat mobiililaitteiden monialustaisen ohjelmistokehityksen neljään eri päätyyppiin: monialustainen kääntäminen, virtuaalikoneet, web-sovellukset (puhtaat ja hybridit) sekä mobiili-widgetit. Heitkötter, Hanschke ja Majchrzak (2013) puolestaan tekevät jaon kolmeen osaan: virtuaalikoneet, puhtaat web-sovellukset ja hybridiratkaisut. Monialustaista kääntämistä he eivät pidä vielä toistaiseksi toimivana, sillä varteenotettavia tekniikoita ei heidän mukaansa ole. Tämän tutkielman tavoitteena on tarkastella monialustaratkaisuja, jotka toimivat myös mobiilialustojen ulkopuolella ja soveltuvat erityisesti pelien kehitykseen. Näin ollen edellä mainituista vaihtoehdoista voidaan jättää suoraan huomioimatta mobiili-widgetit.

Monialustaratkaisut jaetaan tässä tutkielmassa seuraaviin päätyyppeihin: monialustainen kääntäminen, virtuaalikoneet, puhtaat web-sovellukset ja hybridiratkaisut. Hybridiratkaisuiksi luetellaan myös sellaiset virtuaalikoneisiin perustuvat tekniikat, joiden sovelluskehys toimii Internet-selaimen sisällä eikä itsenäisenä sovelluksena. Edellä lueteltujen tekniikoiden lisäksi on olemassa myös muita monialustaratkaisuja, joiden avulla esimerkiksi laitekohtainen lähdekoodi on mahdollista muuntaa toiselle järjestelmälle (Sommer ja Krusche 2013, s. 365). Yksi lähdekoodin muuntotyökaluista on mm. *J2ObjC*, jolla Java-kielinen lähdekoodi voidaan muuntaa Objective-C -kielelle. Tällaiset monialustaratkaisut sivuutetaan tässä tutkielmassa niiden vähäisen suosion vuoksi.

3.3.1 Monialustainen kääntäminen

Viime vuosina ohjelmistokirjastot sekä työkalut ovat monipuolistuneet ja vähentäneet monialustaiseen kääntämiseen käytettävää työtä. Nämä tekniikat mahdollistavat ohjelman kääntämisen suoraan eri käyttöjärjestelmille, jolloin ohjelma tarvitsee varsinaisesti kirjoittaa vain kerran. Käytännössä tämä tarkoittaa sitä, että ohjelmistokirjastot tarjoavat laitteistoriippumattoman *API*:n, jonka kautta ohjelma voi kommunikoida käyttöjärjestelmän kanssa (Hartmann, Stead ja DeGani 2011, s. 3).

Kääntämisen jälkeen ohjelma toimii kuten natiivisovellus ja jakaa näin natiivisovellusten sekä hyvät että huonot puolet. Esimerkiksi ohjelman päivittäminen vaatii edelleen kehittäjältään ohjelman kääntämistä erikseen jokaiselle laitealustalle. Lisäksi ohjelma on myös asennettava, jotta sitä voidaan käyttää. Yksi monialustaisen kääntämisen mahdollistava ohjelmistotyökalu on *Qt*, joka tukee yleisimpiä pöytä- ja mobiilijärjestelmiä. Qt-pohjaisten sovellusten ensisijaisena kehityskielenä on *C++* (The Qt Company 2015). Vastaava hieman enemmän graafisten sovellusten, kuten pelien kehitykseen suunnattu monialustaista kääntämistä tukeva kehitystyökalu on *Marmalade* (Marmalade 2015).

3.3.2 Virtuaalikoneet

Virtuaalikoneratkaisut[1] pitävät pääsääntöisesti sisällään natiiviohjelman, joka tulkkaa ja suorittaa sille syötettyä varsinaista sovelluskoodia. Tällä tavoin sama sovellustoteutus saadaan toimimaan eri järjestelmillä, sillä riittää että vain virtuaalikone on laiteriippuvainen. Virtuaaliratkaisu tarjoaa monialustaisen kääntämisen tavoin API:n, jonka kautta ohjelma kommunikoi käyttöjärjestelmän kanssa. Ohjelman näkökulmasta virtuaalikone on käytännössä se laitteisto, jolla ohjelmaa ajetaan. Virtuaalitoteutukset ovat monialustaista kääntämistä huomattavasti ylläpidettävämpiä, mutta niiden suorituskyky ei tulkkauksen takia yllä välttämättä natiivisovellusten tasolle. (Hartmann, Stead ja DeGani 2011, s. 3.)

Java-ohjelmointikielen *Java Runtime Environment* on yksi esimerkki virtuaalikoneesta, jolla yksi lähdekoodi on mahdollista saada toimimaan monilla eri laitteilla. Javan yksi tunnetuista mainoslauseista onkin: *"write once, run everywhere"*. Java ei ole kuitenkaan onnistunut

1. voidaan nimittää myös *itsenäisiksi sovelluskehyksiksi*

varsinaisesti saavuttamaan tätä tavoitetta, sillä laitteistoriippumattomien sovellusten kehittäminen sen avulla ei ole ollut aivan niin helppoa, kuin olisi voinut toivoa (Corral ym. 2011, s. 182). Kilpailevat tekniikat, kuten monialustainen kääntäminen ovatkin vähentäneet vuosien saatossa Javan suosiota. Grupp (2012, s. 5) huomauttaa, että erityisesti ohjelmistoyhtiö *Sunin* ja Microsoftin välinen lakiriita 2000-luvun vaihteessa hankaloitti Javan yleistymistä, sillä Microsoft ei enää oletuksena sisällyttänyt Javan virtuaalikonetta käyttöjärjestelmiinsä.

3.3.3 Puhtaat web-sovellukset

Puhtaat web-sovellukset ovat Internet-selaimessa toimivia ohjelmia, joissa itse selain toimii varsinaisena ohjelmistoalustana (Anttonen ym. 2011). Toiminnallisuus puhtaissa web-sovelluksissa toteutetaan HTML:n, *CSS*:n ja *JavaScriptin* avulla. HTML-tekniikalla toimivia sovelluksia ei tarvitse asentaa, vaan ne toimivat suoraan web-sivustoilta kaikilla selaimilla ilman tarvetta erillisille lisäosille. Myös sovellusten ulkonäkö ja toiminnallisuus ovat laitteesta riippumatta identtisiä, mutta voivat erota esimerkiksi ruudun koon mukaan. Ongelmana on kuitenkin rajoittunut pääsy moniin laitteistokohtaisiin toiminnallisuuksiin, kuten esimerkiksi kameroihin ja GPS-laitteisiin (Heitkötter, Hanschke ja Majchrzak 2013, s. 122).

Puhtaiden web-sovellusten suorituskyky ei yllä aivan natiivisovellusten tasolle, sillä selain toimii itsessään sovelluksen virtuaalikoneena, eikä tekniikka ole välttämättä yhtä optimoitua, kuin esimerkiksi pidemmän kehityshistorian omaavassa Java-kielessä (Ding ym. 2012, s. 165). Viime vuosina laiteteknologia ja tekniikan kehittyminen ovat tehneet erot suorituskyvyssä joka tapauksessa selvästi pienemmiksi (Holzinger, Treitler ja Slany 2012, s. 183). Tekniikan ehdottomana etuna on sen standardointi sekä Internet-selainten yleisyys. Puhtaat web-sovellukset ovat tämän tutkielman keskeinen tekniikka ja niihin tutustutaan tarkemmin luvussa 4.

3.3.4 Hybridiratkaisut

Hybridiratkaisut ovat natiivisovellusten ja puhtaiden web-sovellusten risteyksiä. Hybriditekniikat syntyivät alun perin täyttämään pääasiassa HTML-tekniikassa olevia puutteita, joka ei pystynyt tarjoamaan tarvittavia laitekohtaisia rajapintoja (Heitkötter, Hanschke ja Majchrzak

2013, s. 123). Hybridiratkaisuiksi luetellaan tässä tutkielmassa tekniikat, joiden sovelluskehys toimii suoraan Internet-selaimen sisällä ja sellaiset itsenäiset sovellukset, jotka on toteutettu puhtaiden web-sovellusten tekniikoilla. Jälkimmäisessä tapauksessa varsinainen sovellus luodaan HTML:n, JavaScriptin ja CSS:n avulla. Tämän lisäksi hybridialusta tarjoaa myös oman API:n, jonka kautta on mahdollista laajentaa sovelluksen toiminnallisuutta puhtaiden web-sovellusten ulkopuolelle. Hybridisovellus on natiivisovelluksen tapaan asennettava käytetylle laitteelle ja esimerkiksi mobiililaitteilla niitä on yleensä mahdollista julkaista sovelluskauppojen kautta (Charaf ym. 2014, s. 530).

Internet-selaimessa toimivia hybridisovelluksia voidaan toteuttaa monilla eri ohjelmointikielillä ja työkaluilla. Selainsovellusten hybridiratkaisut vaativat, että kyseisen tekniikan mahdollistava lisäosa on jo asennettuna selaimeen (Grupp 2012, s. 5). Tämänkaltaisista hybridiratkaisuista tunnetuin lienee Adobe Flash, jonka avulla pelejä on kehitetty selaimille jo 90-luvun lopulta (Mäyrä 2015, s. 322). Hyvin suuri osa selaimessa pelattavista peleistä on nimenomaan Flash-toteutuksia (Curran ja George 2012, s. 33). Huolimatta Flashin suhteellisen menestyksekkäästä historiasta, on sen tekniikka jo osittain vanhentunutta ja Adobe on siirtynyt kehittämään HTML-ratkaisuun perustuvaa *Adobe Edgeä*.

PhoneGap on avoimen lähdekoodin hybridisovellusten kehitystyökalu mobiililaitteille. PhoneGap on erillinen sovelluskehys, joka mahdollistaa HTML-tekniikan käyttämisen ja tarjoaa oman rajapintansa laajennetuille toiminnallisuuksille (PhoneGap 2015). PhoneGap ei toimi muilla kuin mobiililaitteilla ja se on sisällytetty tähän tutkielmaan lähinnä esimerkkinä hybridisovelluksista.

3.4 Monialustaiset pelinkehitystyökalut

Pääasiassa normaaleille sovelluksille tarkoitettujen monialustatekniikoiden lisäksi on olemassa puhtaasti pelinkehitykseen suunnattuja työkaluja. Monet näistä ohjelmistoista mahdollistavat pelien kehittämisen suoraan useille eri järjestelmille. Ratkaisut pitävät sisällään valmiita peleihin liittyviä ohjelmistokirjastoja ja muita mahdollisia apusovelluksia. Kaikkiin peleihin soveltuvaa universaalia työkalua ei ole olemassa, joten tässä luvussa esiteltävät kehitystyökalut edustavat hyvin erilaisia kehitysratkaisuja. Esiteltävät ratkaisut tukevat

joko monialustaista kääntämistä tai ovat hybridiohjelmistoja. Puhtaasti HTML-tekniikkaan perustuvia pelinkehitystyökaluja esitellään luvussa 4.8.

Pelimoottorit ovat ohjelmistokirjastojen kokonaisuuksia, joiden avulla pelinkehitys voidaan jakaa osiin ja joilla voidaan mahdollistaa saman toiminnallisuuden hyödyntämisen helposti useissa eri peleissä. Ne voivat sisältää esimerkiksi fysiikan mallinnusta, sekä tuen syötelaitteistolle ja grafiikan piirrolle. Pelimoottorin keskeinen käyttötarkoitus on helpottaa pelinkehitykseen liittyvää työtä ja kustannuksia, sekä vähentää tarvetta pelin kehittämiseen täysin alusta alkaen (Santelices ja Nussbaum 2001). Pelimoottorit ovat usein myös hyvin optimoituja ja kykenevät helposti parempaan suorituskykyyn, kuin pelinkehittäjien mahdolliset omat ratkaisut. Pelimoottorit tarjoavat oman API:n, jonka kautta sitä voidaan vaivattomasti hyödyntää. Pelimoottorit ovat yleensä kehitetty tietyn genren pelejä varten ja toiminnallisuuksiltaan ne voivat erota toisistaan merkittävästi (Gregory 2014).

Pelinkehitysalustoilla tarkoitetaan tässä tutkielmassa sellaisia pelien kehitykseen tarkoitettuja työkalukokonaisuuksia, joiden avulla pelin logiikkaa ja objekteja voidaan manipuloida suoraan mukana tulevan graafisen käyttöliittymän kautta. Pelinkehitysalustat vähentävät tarvetta lähdekoodin kirjoittamiselle ja pyrkivät sisällyttämään kaikki pelien toteuttamiseen tarvittavat ohjelmistokirjastot. Äärimmillään pelinkehitysalustoilla voidaan tehdä pelejä ilman ohjelmointia. Pelinkehitysalustat sisältävät oman pelimoottorinsa, jossa on usein esimerkiksi fysiikkaan ja tekoälyyn liittyviä ominaisuuksia.

Pelimoottorilla tai kehitysalustalla toteutettujen pelien loogista toiminnallisuutta on usein mahdollista ohjelmoida erilaisten *skriptikielten* avulla. Skriptikielet ovat pelimoottorin tulkkaamia ohjelmointikieliä, jotka on suunniteltu pelin sisäisten tapahtumien ohjelmointiin. Ne pohjautuvat usein perinteisiin ohjelmointikieliin, mutta voivat kuitenkin olla hyvin yksinkertaistettuja versioita esikuvistaan. Skriptikielten avulla myös vähemmän kokenut ohjelmoija voi helposti luoda peleihin ohjelmallista sisältöä. (Gregory 2014, s. 954–978.)

3.4.1 Cocos2d-X

Cocos2d-X on avoimen lähdekoodin ilmainen 2D-pelimoottori, joka tukee C++, *Lua* ja JavaScript -ohjelmointikieliä. Sillä toteutettuja pelejä voidaan kääntää monille eri käyttöjärjes-

telmille, kuten Android, iOS, OS X, Windows ja Windows Phone. Kehittäminen vaatii erilliset kääntämistyökalunsa jokaista järjestelmäjulkaisua varten. Cocos2d-X tarjoaa laajan valikoiman erilaisia graafisia ominaisuuksia sekä tuen äänille, fysiikan mallinnukselle ja verkkopelaamiselle. Sen rinnalle on olemassa myös laaja valikoima erillisiä apuohjelmia, jotka helpottavat kehitystyötä. Tarjolla on esimerkiksi pelinkehitysalustaa muistuttava *CocoStudio*, jonka avulla peleihin voidaan luoda mm. animaatioita. (Cocos2d-X 2015.)

3.4.2 GameMaker

GameMaker on monialustaista kääntämistä tukeva pelinkehitysalusta, jolla voidaan kehittää 2D-pelejä. Ohjelmasta on olemassa sekä maksuton että maksullinen versio. Ilmainen versio on ominaisuuksiltaan huomattavasti rajoittuneempi, eikä se tue esimerkiksi oletuksena kuin Windows-käyttöjärjestelmiä. Maksullisen version hinta on toiminnallisuuksien mukaan noin 100–800 dollaria. GameMakerin avulla on mahdollista luoda myös puhtaita web-sovelluksia, sillä pelitoteutukset voidaan kääntää HTML-yhteensopiviksi (ei ilmaisversiolla). (GameMaker 2015.)

3.4.3 Unity

Unity on pelinkehitysalusta, joka mahdollistaa pelien kehittämisen kaikille tunnetuimmille käyttöjärjestelmille ja myös pelikonsoleille (Unity 2015). Unity sisältää oman pelimoottorinsa ja se tukee sekä 2D- että 3D-grafiikkaa. Pelien kehittäminen sen avulla on nopeaa, sillä se tarjoaa monipuolisen valikoiman valmiita ohjelmistokirjastoja (Curran ja George 2012, s. 35). Unity on yksi suosituimmista pelinkehitysalustoista ja sen avulla on tehty myös hyvin nimekkäitä pelijulkaisuja, kuten *Cities: Skylines*, *Wasteland 2* ja *Angry Birds Epic*. Suosionsa ansiosta myös sen dokumentaatio on hyvin monipuolinen. Unityllä kehitetyt pelit käännetään jokaiselle järjestelmälle erikseen tai niitä ajetaan virtuaalikoneen päällä riippuen käyttöjärjestelmästä (Andersson ja Johansson 2014, s. 3).

Unityn avulla on myös mahdollista julkaista selainpohjaisia pelejä, jotka vaativat *Unity web player* -selainlaajennuksen asentamista käyttäjän koneelle. Kirjoitushetkellä web player ei kuitenkaan vielä tukenut esimerkiksi Linux-käyttöjärjestelmiä. Unityn kehitysalustasta on

olemassa maksuton ja maksullinen versio. Maksuton *Personal*-versio on ominaisuuksiltaan rajoittuneempi kuin *Professional*-versio, mutta mahdollistaa joka tapauksessa julkaisun kaikille tuetuille järjestelmille. Mikäli Unityllä kehitetyn pelin tuotot ylittävät 100 000 dollaria vuodessa, on kehittäjä velvoitettu ostamaan Professional-lisenssi. (Unity 2015.)

3.4.4 Unreal Engine

Unreal Engine on pelinkehitysalusta, joka on tunnettu erityisesti FPS-peleistään. Se tukee useimpia pöytä- ja mobiililaitteita sekä uusimpia pelikonsoleita. Unreal Enginen avulla on mahdollista luoda graafisesti erittäin näyttäviä 3D-pelejä. Sen avulla kehitetyistä peleistä maksetaan lisenssimaksuna 5 % osuus neljännesvuosittaisista tuotoista, jotka ylittävät 3 000 dollaria. Pelit kuten *Batman Arkham*, *Death Rally* ja *Gears of War* on kehitetty Unreal Enginen avulla. (Unreal Engine 2015.)

3.5 Tekniikoiden vertailu

Kaikki edellä esitellyistä tekniikoista eivät välttämättä sovellu täydellisesti kaikille järjestelmille, sillä myös laitekohtaiset erot vaikuttavat monialustatekniikoiden valintaan. Mobiililaitteilla on esimerkiksi omat rajoitteensa, joita pöytäkoneilla ei tarvitse välttämättä huomioida. Tällaisia rajoitteita ovat mm. mobiililaitteiden rajoittunut laskentateho ja virrankäytön rajoitteet (Corral ym. 2011, s. 183). Andersson ja Johansson (2014) ovat verranneet GameMakerin, Qt:n, PhoneGapin, Unityn ja natiivisovellusten eroja mobiililaitteilla. Tutkimuksessa huomioitiin erityisesti virrankäyttöön liittyvät erot, joiden mukaan Qt oli virran suhteen säästeliäin ja vei vain puolet samasta virtamäärästä, kuin mitä esimerkiksi Unity ja GameMaker. Ainoa hybriditekniikkaa edustava tekniikka PhoneGap oli suorituskyvyltään heikoin, Unityn ja GameMakerin ollessa nopeimpia.

Corral, Sillitti ja Succi (2012) ovat verranneet Android-mobiililaitteella PhoneGap-hybridisovellusten ja natiivisovellusten suorituskykyä. Tutkimuksessa PhoneGap-toteutukset todettiin natiivisovelluksia selvästi hitaammiksi. Heidän mukaansa onkin tärkeää hyväksyä, etteivät web-teknologioita hyödyntävät ohjelmat voi luultavasti koskaan saavuttaa täysin natiivisovellusten tehokkuutta.

Myös Heitkötter, Hanschke ja Majchrzak (2013) ovat vertailleet PhoneGap, *Titanium Mobile* ja natiivisovellusten eroja mobiililaitteilla. Titanium Mobile on virtuaalikonetekniikka mobiilisovellusten monialustaiselle kehittämiselle. Keskeinen johtopäätös tutkimuksessa on, että edellä olevat tekniikat ovat täysin toimivia monialustaratkaisuja mobiililaitteille ja haastavat natiivisovellukset kehityksen helppoudessa. Heidän mukaansa PhoneGap-toteutus on varteenotettavin, mikäli sovellusten ei välttämättä tarvitse saavuttaa kaikilla järjestelmillä mahdollisimman luontaista ulkonäköä.

Raivio (2013) on tutkielmassaan testannut kolmea monialustaista kehitystyökalua mobiilisovelluksen toteutukseen. *Appcelerator Titanium*, PhoneGap ja *Sencha Touch* todettiin kukin tutkimuksessa toimivaksi tekniikaksi. Suurimmiksi heikkouksi listattiin mm. testattujen työkalujen huono laitteistotuki ja hybriditekniikoiden suorituskyky. Myös Kasari (2012) on vertaillut mobiilisovelluksen kehitystä eri tekniikoilla. Kasarin tutkimuksessa sama sovellus toteutettiin natiivi-, hybridi- ja puhtaana web-sovelluksena. Toteutuksia verrattiin toisiinsa *MeeGo*-laitealustalla. Tutkimuksen mukaan natiivisovellus oli monilta osin web-toteutuksia parempi, mahdollistaen mm. helpomman käyttöliittymäkehityksen ja paremman suorituskyvyn. Kasari kuitenkin toteaa, että web-sovellusten toteutustekniikoilla on omat hyvät puolensa ja ne voivat osoittautua tulevaisuudessa hyvin suosituiksi.

Holzinger, Treitler ja Slany (2012) ovat tutkineet eri monialustasovellusten kehitystä mobiililaitteille ja verranneet tekniikoiden soveltuvuutta eri laitteistotyyppien välillä. Tutkimuksessa huomioitiin esimerkiksi näyttöruudun koko ja arvioitiin sovellusten käytettävyyttä eri ruuduilla. Heidän mukaansa mikään testatuista tekniikoista ei ollut muita merkittävästi parempi, sillä sovellusten lopputulos riippuu suurilta osin myös sovelluskehittäjien teknisestä osaamisesta. Joka tapauksessa sovelluskehitys eri laitteistolle vaatii merkittävästi testaamista huolimatta kehitystyökalun valinnasta.

Ylönen (2014) on tutkinut monialustaisten mobiilipelien kehitystyökaluja ja vertaillut kokeellisesti Cocos2d-X:n sekä Unityn eroja. Tutkimuksessa ei kumpaakaan ratkaisua havaittu merkittävästi toistaan paremmaksi. Tekniikat eroavat Ylösen mukaan lähinnä kehitystapojensa perusteella, ja Unity havaittiin tutkimuksessa lähestyttävämmäksi, mikäli kehittäjä ei hallitse erityisesti ohjelmointia.

Edellä olevan kirjallisuuskatsauksen perusteella sovellusten kannalta natiivitoteutus on suositeltavin, mikäli sovelluksen graafinen käyttöliittymä halutaan säilyttää laitteiston tyylin mukaisena. Peleihin tämä rajoitus ei välttämättä merkittävästi vaikuta, sillä varsinkin pelinaikainen grafiikka piirretään usein ilman järjestelmän käyttöliittymää. Natiivisovellukset ovat myös suorituskyvyltään väistämättä tehokkaimpia tapoja sovellusten toteutukseen, mutta nämä saavutetut edut kostautuvat joka tapauksessa monialustaisen kehitystyön merkittävänä vaikeutumisena.

3.6 Selainpohjaiset pelit

Englanninkielisessä kirjallisuudessa käsitteitä *browser game* ja *web game* käytetään hieman eri merkityksissä. Sekä Marszałkowski (2011) että Vanhatupa (2010) viittaavat browser game -nimityksellä yksinomaan sellaisiin moninpeleihin, joita pelataan selaimen kautta pitkiä aikoja ja jotka vaativat pelaamista varten erillisen käyttäjätilin. Esimerkiksi hyvin suositut pelit, kuten *Hattrick* ja *Travian* ovat tähän kategoriaan luokiteltavia selainpelejä (Vanhatupa 2010, s. 42). Web game -määritelmä taas on usein laajempikäsitteinen ja kattaa myös muut selaimessa pelattavat pelit (esim. Zhang 2012, s. 22). Tässä tutkielmassa ei ole tarpeen erotella erityyppisiä selaimessa pelattavia pelejä toisistaan, joten yleisesti kaikista Internet-selaimessa pelattavista peleistä käytetäänkin jatkossa nimitystä *selainpohjaiset pelit*.

Edellä olevissa luvuissa käsiteltiin yleisesti monialustaisen pelikehityksen mahdollistavia työkaluja. Yleinen ongelma joka kosketti kaikkia muita tekniikoita paitsi puhtaita web-sovelluksia, oli pelin saavutettavuus kaikilla laitteilla ilman erillistä asentamista. Osittaisena poikkeuksena tähän ovat selaimessa toimivat sovelluskehykset, jotka käyttäjän tarvitsee asentaa vain kerran, jonka jälkeen kyseistä lisäosaa hyödyntäviä pelejä on mahdollista pelata suoraan web-sivustoilta. Nämä selainten lisäosina toimivat hybridisovellukset eivät syystä tai toisesta ole kuitenkaan onnistuneet yleistymään kaikilla laitealustoilla, sillä joko niitä ei ole ylipäätään saatavilla kaikille mahdollisille laitteille tai käyttäjän on yhä itse asennettava ne. Yleisessä käytössä olevaa selaimen lisäosana toimivaa monialustaista peliratkaisua ei siis – ainakaan toistaiseksi – ole saatavilla.

Myös kaikki luvussa 3.4 esitellyt pelinkehitystyökalut kärsivät edellä mainitusta asentami-

seen liittyvästä ongelmasta. Kuten luvussa 2.3 mainittiin, jos peli vaatii asentamista, on se julkaistava laitekohtaisissa sovelluskaupoissa, mikäli peliä halutaan levittää esimerkiksi mobiililaitteille. Tämä tarkoittaa samalla sovelluskauppojen asettamien ehtojen ja mahdollisten maksujen hyväksymistä. Kehitystyökalujen sisältämät lisenssiehdot asettavat nekin mahdollisesti omat lisärajoituksensa pelin toteutukselle ja julkaisulle. Pelinkehitystyökalut saattavat kyllä osaltaan nopeuttaa pelinkehitystä, mutta näiden ohjelmistojen ilmaisversiot ovat ominaisuuksiltaan usein rajoitettuja, eivätkä siten välttämättä sovellu kaikkiin tilanteisiin.

Yksi uusista ja poikkeuksellisemmista tavoista pelata on *pilvipelaaminen*. Pilvipelaamisessa peliä ajetaan erillisellä palvelimella ja peli suoratoistetaan (engl. *streaming*) pelaajan laitteelle (Cai, Leung ja Chen 2013, s. 550–551). Pilvipelaamisen avulla myös mobiililaitteilla voi olla mahdollista pelata suorituskyvyltään hyvin vaativia pelejä. Se vaatii kuitenkin toimiakseen hyvin tehokkaan verkkoyhteyden ja mikäli mobiililaitteen suorituskyky ei riitä selainpeleihin, ei se luultavasti riitä myöskään tehokkaaseen suoratoistoon (Cai, Leung ja Chen 2013, s. 552). Tässä tutkielmassa pilvipelaamista ei käsitellä tarkemmin.

Ohjelmistoteknologian yleisenä suuntauksena on sovellusten siirtyminen paikallisten laiteasennusten sijaan verkkopalveluiksi (esim. Taivalsaari ym. 2011, s. 17). Internet-selaimesta on tullut yleinen ohjelmistoalusta, jonka avulla on mahdollista käyttää esimerkiksi *Google Driven* tapaisia toimisto-ohjelmistoja ilman että käyttäjän tarvitsee asentaa niitä laitteelleen (Taivalsaari ym. 2011). Myös kokonaisia käyttöjärjestelmiä, kuten *Chromium OS* on luotu selainteknologian ympärille (Anttonen ym. 2011, s. 804). Videopelit ovat osaltaan seuranneet tätä trendiä, sillä selaimissa pelattavat pelit ovat jo saavuttaneet suuren suosion erityisesti niiden helpon saavutettavuuden myötä (Vanhatupa 2011, s. 366).

Selainpelien uranuurtajia ovat olleet mm. hybridiratkaisut, sillä HTML-teknologian suorituskyky on pitkään ollut rajoittavana tekijänä HTML-pelien yleistymiselle. Tilanne on nyt kuitenkin merkittävästi muuttumassa, sillä selaintekniikat ovat kovaa vauhtia saavuttamassa pelien vaatiman tehokkuuden (Anttonen ym. 2011, s. 803). Pelin toteuttaminen puhtaana web-sovelluksena voi myös osaltaan ratkaista ongelman pelin saatavuuden ja monialustaisuuden kanssa, sillä HTML-pelien tulisi oletuksen mukaan toimia kaikilla laitteilla suoraan web-sivustoilta.

Pelien sijaitseminen yksinomaan verkossa tuo samalla mukanaan myös joitain muita hyötyjä. Käyttäjän ei tarvitse esimerkiksi päivittää selainpelejä manuaalisesti, vaan koska peli sijaitsee verkossa, latautuu sen uusin versio suoraan pelaamista aloitettaessa (Gawley, Barr ja Barr 2012, s. 202). Selainpeli voi olla myös eri web-lähteistä koostettu kokonaisuus (ns. *mashup*), jolloin mm. pelimoottorin päivitys voi myös kehittäjän näkökulmasta olla mahdollista toteuttaa automaattisesti (Anttonen ym. 2011, s. 806).

Selainpelin julkaisemiseksi riittää yksinkertaisimmillaan sivuston lisääminen käytetylle palvelimelle (Taivalsaari ym. 2011, s. 18). Juntunen, Jalonen ja Luukkainen (2013) toteavat, että tämä voi olla kustannuksellisesti kannattavaa, sillä sovelluskauppojen ottama 30 % provisio on huomattava. Heidän mukaansa nettisivustolla sijaitsevien HTML-sovellusten yhtenä etuna on myös niiden näkyvyys Internetin hakupalveluissa, mikä helpottaa merkittävästi niiden löytymistä.

HTML-tekniikka ei ole välttämättä täysin ongelmatonta, sillä esimerkiksi Facebook teki eräässä vaiheessa yrityksen kääntää sovelluksensa selainpohjaisiksi, mutta HTML-tekniikan suorituskyky osoittautui tuolloin liian huonoksi (Li ja Bao 2014, s. 252). Tekniikan ongelmaksi voi toistaiseksi myös koitua sen yhteensopivuusongelmat eri laitteistoilla. HTML-peleille ei ole myöskään olemassa vakiintunutta markkinointikanavaansa, mikä pakottaa pelinkehittäjät julkaisemaan pelinsä omalla palvelimella. Tämä voi tarkoittaa myös oman maksujärjestelmän ylläpitoa, mikä varsinkin pienten pelinkehittäjien tapauksessa voi olla käytännössä mahdotonta toteuttaa.

4 HTML5-standardi

HTML on merkintäkieli, joka määrittelee Internet-sivustojen esittämiseen liittyvien elementtien syntaksin. HTML5 on tämän HTML-standardin tuorein laajennos, jonka uudistuksia ovat mm. multimedian esitykseen tarkoitetut elementit, kuten `<audio>`, `<canvas>` ja `<video>`. HTML5:n kehitys alkoi virallisesti vuonna 2004, kun *Mozilla* sekä *Opera* aloittivat standardin valmistelemisen (W3C 2004). Noin kymmenen vuotta myöhemmin lokakuussa 2014, HTML5-standardista julkistettiin ensimmäinen virallinen suositeltu versio (engl. *recommendation*) HTML 5.0 (W3C 2014). Tarve standardille oli suuri, ja tekniikka otettiin Internetissä käyttöön laajasti jo vuosia ennen sen virallistamista, sillä selainvalmistajat sisällyttivät HTML5:n perusominaisuuksia selaimiinsa jo standardin kehitysvaiheessa. Standardia laajennetaan jatkossa uusilla ominaisuuksilla ja seuraava standardisuositus HTML 5.1 virallistettaneen vuonna 2016.

Vaikka HTML5 on itsessään varsinaisesti vain elementit määrittävä standardi, kattaa se yleisesti tunnettuna käsitteenä usein myös siihen sidoksissa olevat muut tekniikat, kuten *CSS3*:n, *JavaScriptin* ja *WebGL*:n (esim. Garaizar, Vadillo ja Lopez-de-Ipina 2012, s. 23). Tässä tutkielmassa termillä HTML5 tarkoitetaan nimenomaan tätä edellä mainittua kokonaisuutta, joka mahdollistaa selaimessa toimivien client-sovellusten ja erityisesti pelien toteuttamisen. Client-sovelluksilla tarkoitetaan tässä yhteydessä sellaisia sovelluksia, joita ajetaan asiakaskoneella eivätkä ne välttämättä tarvitse aktiivista yhteyttä ulkoisiin palvelimiin suorituksensa aikana.

HTML5-selainpeli voi olla puhtaasti client-pohjainen tai siihen voi liittyä palvelin (Vanhatupa 2011, s. 364). Palvelimen avulla on mahdollista luoda esimerkiksi moninpeliin liittyviä toiminnallisuuksia tai sillä voidaan suorittaa tarvittaessa raskaampaa laskentaa. Esimerkiksi Kuuskeri ja Mikkonen (2009) ovat selvittäneet, kuinka selainsovellusten toiminnallisuus tulisi tehokkaasti osittaa palvelimen ja asiakaskoneiden välillä. Tässä tutkielmassa rajoitutaan yksinkertaisuuden vuoksi tarkastelemaan puhtaasti client-sovelluksena toimivia pelejä. HTML5 mahdollistaa joka tapauksessa monipuolisen kommunikoinnin palvelimien kanssa, eikä varsinaisesti sulje pois mitään palvelinratkaisua.

4.1 JavaScript

JavaScript[1] on yleisin modernien Internet-selainten tukema client-sovellusten olio-ohjelmointikieli (Taivalsaari ym. 2011, s. 19). Nimestään huolimatta JavaScriptillä ei ole varsinaisesti kielen syntaksin yhtäläisyyksiä lukuun ottamatta mitään tekemistä Java-ohjelmointikielen kanssa (Mikkonen ja Taivalsaari 2007, s. 3). JavaScript kehitettiin alun perin yksinkertaisten toimintojen tekemiseksi suoraan käyttäjäkoneella ilman tarvetta jatkuvalle palvelimen väliselle kommunikaatiolle (Khan ym. 2014, s. 91). Kielen käyttö on nykypäivänä laajentunut kattamaan huomattavasti monimutkaisempia kokonaisuuksia, kuin mitä oli alun perin ajateltu ja tämä on aiheuttanut yleistä kritiikkiä kieltä kohtaan (esim. Rastogi ym. 2014). Kielen kehitys jatkuu yhä, ja seuraavat versiot tulevat korjaamaan keskeisimpiä ongelmia (Anttonen ym. 2011, s. 802–803).

JavaScript on dynaaminen ohjelmointikieli, jossa esimerkiksi muuttujien tyyppitarkistus tehdään vasta suorituksen aikana (Aho ym. 2012, s. 59). Taivalsaari ym. (2008) listaavatkin JavaScriptin hyviksi puoliksi erityisesti juuri dynaamisuuden, *C*-kieltä muistuttavan syntaksin ja yleisen potentiaalin hyvään suorituskykyyn. JavaScriptissä esimerkiksi funktioita voidaan muuttaa sovelluksen suorituksen aikana, mikä web-ohjelmoinnissa on usein hyvin tarpeellinen ominaisuus (Mikkonen ja Taivalsaari 2007, s. 3–4). Kielen dynaamisuus voi toisaalta tehdä virheiden jäljittämisestä vaikeaa, ja JavaScript-sovellukset vaativat yleensä huomattavasti testaamista (Taivalsaari ym. 2011, s. 19–20).

Selainsovellusten ohjelmointi ei ole varsinaisesti rajoittunut yksinomaan JavaScriptille, sillä on olemassa myös muita ohjelmointikieliä, joita on mahdollista käyttää selainsovellusten toteuttamiseen. Muiden ohjelmointikielten tapauksissa lähdekoodi vain käännetään ennen julkaisua puhtaaksi JavaScriptiksi. Tämä mahdollistaa esimerkiksi *Elmin* tapaisen funktionaalisen ohjelmointikielen käyttämisen. Myös JavaScriptin perustoiminnallisuutta laajentavien kielten, kuten *TypeScriptin* käyttö on mahdollista. TypeScript on Microsoftin kehittämä kielistandardi, joka lisää esimerkiksi muuttujien tyyppimääritelmät ja monipuolistaa kielen oliopohjaisuutta (Rastogi ym. 2014).

JavaScriptin avoimen luonteen vuoksi client-puolen lähdekoodi on täysin käyttäjän muokat-

1. kielistandardin virallinen nimitys on *ECMAScript*

tavissa. Curran ja George (2012, s. 32–33) toteavat, että ongelmaa voidaan yrittää kiertää käyttämällä erityisiä apuohjelmia, joilla lähdekoodista voidaan tehdä vähemmän luettavampi (engl. *obfuscation*). On kuitenkin epäselvää, onko kyseinen menetelmä todella toimiva, vai onko lähdekoodi mahdollista muuntaa aina takaisin täysin luettavaan muotoon. Lähdekoodin kommentoinnit ja muuttujien alkuperäiset nimet kyseisellä menetelmällä on joka tapauksessa mahdollista poistaa lopullisesti.

4.2 Grafiikka

Pelien kannalta yksi tärkeimmistä uudistuksista HTML5:ssä on `<canvas>`-elementti. Se mahdollistaa selaimissa bittikarttapohjaisen grafiikan, tekstin sekä animaatioiden luomisen. Elementti tarjoaa sovelluksille JavaScriptin kautta saavutettavan API:n, jolla siihen voidaan piirtää. Canvakseen liittyy erittäin oleellisena teknologiana myös *WebGL*, joka on JavaScriptille tarkoitettu *OpenGL*-rajapinta (Curran ja George 2012, s. 32–33). Käytännössä WebGL mahdollistaa GPU:n avulla laitteistokiihdytetyn kaksi- sekä kolmiulotteisen grafiikan renderöinnin suoraan canvas-elementissä. Peleille tämä ominaisuus on erityisen tärkeä, sillä laitteistokiihdytetyn grafiikan piirto on ensisijainen tekniikka pelien suorituskyvyn kannalta.

Taivalsaari ym. (2011) toteavat, että juuri WebGL tulee olemaan ratkaiseva tekijä selainpelien lopullisen läpimurron saavuttamiseksi. Curran ja George (2012, s. 34–35) pohtivat, että kolmiulotteiset WebGL-pelit tulevat vielä yleistymään Internetissä merkittävästi, mutta ovat kuitenkin sitä mieltä, että se tuskin tapahtuu ainakaan ennen vuotta 2022. Myös Nazarov ja Galletly (2013) suhtautuvat WebGL:ään varovaisen optimistisesti ja kirjoittavat, että vaikka WebGL ei ole vielä välttämättä valmis monimutkaisen ja dynaamisen 3D-grafiikan näyttämiseen, on sen etuna tekniikan takana olevat tahot, jotka ovat kovaa vauhtia tuomassa WebGL:ää kaikille selaimille. Grupp (2012) toteaakin, että WebGL:n todellinen voimavara on sen suuri OpenGL-kehittäjäyhteisö, joka tulee luomaan tekniikasta ennen pitkää erittäin suorituskykyistä.

Microsoft ei alkuun sisällyttänyt WebGL:n toiminnallisuutta Internet Exploreriin perustellen sitä turvallisuussyillä (Curran ja George 2012, s. 34). Ongelma ei ole vakuuttanut merkittävästi muita selainvalmistajia sillä Nazarov ja Galletly (2013) huomauttavat, että esimerkiksi

Mozilla on pitänyt tekemiään suojauskeinoja riittävinä ja on sallinut WebGL-tuen selaimissaan jo pitkään. Tietoturvaongelmat jotka koskettavat WebGL:ää ovat olemassa myös muissa vastaavissa tekniikoissa, kuten Adobe Flashissa ja Microsoftin *Silverlightissa*, eivätkä ne itsessään estä WebGL:n käyttöä (Grupp 2012). Joka tapauksessa Internet Explorerin versionumerosta 11 alkaen myös Microsoft on sisällyttänyt selaimeensa tuen WebGL:lle (Microsoft 2015). Tämä oletettavasti osoittaa sen, etteivät WebGL:n tietoturvaongelmat ole välttämättä enää itsessään este tekniikalle.

HTML5 tukee myös *SVG*-muodossa olevan vektoripohjaisen grafiikan piirtämistä `<svg>`-elementtiin. Vektorigrafiikan avulla voidaan luoda erinomaisesti skaalautuvia kuvia, jotka bittikarttoihin verrattuna voivat viedä huomattavasti vähemmän tallennustilaa (Li ja Bao 2014, s. 255). Vektoripohjaiset kuvat tallennetaan geometrisessä muodossa yksittäisten pikselien sijaan. Tämä mahdollistaa vektoripohjaisten kuvien helpon muokattavuuden ja liikuttelun, mikä tekeekin niistä erityisen hyviä animaatioiden toteuttamiseen (Garaizar, Vadillo ja Lopez-de-Ipina 2012).

4.3 Muut tekniikat

Edellisissä luvuissa esiteltyjen ominaisuuksien lisäksi HTML5:een liittyy myös muita pelinkehityksen kannalta hyödyllisiä tekniikoita, joihin ei tämän tutkielman puitteissa ole mahdollista luoda tarkempaa katsausta. Tässä luvussa tehdäänkin muutamiin oleellisimpiin tekniikoihin vain pikainen katsaus.

CSS3 on WWW-sivustojen tyyliohjeistuksen uusin standardi. Sen avulla on mahdollista manipuloida HTML-elementtejä sekä määritellä sivuston ulkoasua ja tyylejä. CSS3 soveltuu SVG:n tavoin esimerkiksi animaatioiden luomiseen ja molemmilla menetelmillä on omat hyvät ja huonot puolensa (Garaizar, Vadillo ja Lopez-de-Ipina 2012). Suoraa pelillistä hyötyä CSS3 ei välttämättä itsessään kuitenkaan tarjoa. Li ja Bao (2014, s. 255) ovat myös tutkimuksessaan todenneet, että elementtien piirtäminen CSS3:n avulla hidastuu objektimäärän kasvaessa, eikä se ole suorituskyvyltään canvas-elementin veroinen. Myös Puputti (2012) toteaa tutkimuksensa perusteella CSS3-animaatioiden olevan ainakin mobiililaitteilla hyvin hitaita.

WebCL on *OpenCL*-rajapinta, jolla voidaan suorittaa laitteistokiihdytettyä rinnakkaislaskentaa JavaScriptin kautta. Peleissä rinnakkaislaskentaa on mahdollista hyödyntää esimerkiksi tekoälyyn tai fysiikkaan liittyvässä laskennassa (Blewitt, Ushaw ja Morgan 2013). Jeon, Brutch ja Gibbs (2012) ovat verranneet WebCL-toteutusten suorituskykyä mobiililaitteilla normaaleihin JavaScript-toteutuksiin. Tulosten mukaan WebCL voi parantaa sovellusten suorituskykyä rinnakkaislaskennalle hyvin soveltuvissa toteutuksissa jopa 100-kertaisesti. Khan ym. (2014) mukaan WebCL ei kuitenkaan vielä saavuta aivan samaa hyötykerrointa, kuin esimerkiksi natiivit C-kieliset OpenCL-toteutukset. Heidän mukaansa WebCL vaatii vielä kehitystyötä, mutta teknologia on joka tapauksessa jo joissain tapauksissa vertailukelpoinen natiivitoteutusten kanssa.

WebSockets on protokolla käyttäjän ja palvelimen väliseen kommunikointiin. HTTP-protokollaa ei alun perin tarkoitettu reaaliaikaiseen, kahdensuuntaisen tiedon välitykseen, mutta sen tarve on viime vuosina ollut joka tapauksessa ilmeinen. Korvaavia menetelmiä, kuten *Ajax* on tämän vuoksi kehitetty paikkaamaan palvelimelle tapahtuvan kommunikoinnin tarvetta, mutta WebSocketsin avulla tiedon välitys voidaan toteuttaa aikaisempia ratkaisuja tehokkaammin. Tämä on mahdollista sillä WebSockets tuo mukanaan täysin uusia kommunikointirajapintoja, joita aikaisemmat tekniikat eivät voineet käyttää hyväkseen. WebSockets on osaltaan hyvin tärkeä tekniikka, sillä se mahdollistaa moninpelien tehokkaan toteuttamisen selaimilla. (Curran ja George 2012, s. 36–37.)

HTML5 tukee myös nk. *offline datan* tallennusta *Web Storagen* avulla. Se mahdollistaa aiempaa suuremman tietomäärän tallentamisen selaimeen myöhempää käyttöä varten, ja peleissä sitä voidaan hyödyntää esimerkiksi pelin tilan tallennukseen (engl. *save game*). Vanha HTML-ratkaisu tuki tiedon tallentamista ainoastaan sivukohtaisiin kekseihin (engl. *cookies*), joiden koko oli huomattavan rajoitettu ja ne lähetettiin palvelimelle aina jokaisen HTTP-pyynnön yhteydessä (Curran ja George 2012, s. 37). Nimensä mukaisesti offline data on käytettävissä myös silloin, kun selain ei ole yhdistettynä verkkoon. Tämä voi mahdollistaa sovellusten käyttämisen vaikka Internet-yhteyttä ei olisikaan saatavilla (Juntunen, Jalonen ja Luukkainen 2013, s. 1057).

4.4 Selaintuki

Yhteensopivuusongelmat eri selainten välillä ovat aina olleet HTML-tekniikan yksi suurimmista ongelmista, sillä huolimatta sen standardoinnista, selainvalmistajat ovat kehittäneet selaimiaan hyvin epäyhtenevästi (Anttonen ym. 2011, s. 801–802). Yksi syy tälle voi olla yksinkertaisesti kilpailuasetelma, joka on pakottanut selainvalmistajat luomaan kilpailijoitaan parempia menetelmiä. Toisaalta juuri kilpailuasetelma on ollut ratkaisevassa roolissa selainsovellusten suorituskyvyn merkittäville parannuksille viime vuosina (Charland ja Leroux 2011, s. 50). Gawley, Barr ja Barr (2012) pohtivat, että eri selainten välisiä eroavaisuuksia selittävät osaltaan myös HTML5-standardin muutokset ennen sen varsinaista virallistamista, mikä on johtanut selainvalmistajat tekemään joitain omia, yksilöllisiä ratkaisujaan. Heidän mukaansa myös selainten pohjautuminen eri selainmoottoreihin (esim. *WebKit* ja *Gecko*) on vaikeuttanut HTML5-tuen lopullista yhtenäistämistä.

HTML5 on hyvin laaja kokonaisuus joka käsittää monia laitekohtaisia rajapintoja. Monet näistä rajapinnoista saattavat olla käyttötarpeeltaan myös hyvin harvinaisia. HTML5 määrittelee esimerkiksi rajapinnat mahdollisen ilmanpaineen, lämpötilan, kosteuden ja *NFC*-tietojen lukemiseksi (Rodríguez ym. 2014). Näiden laitekohtaisten rajapintojen tuen implementointi on selainvalmistajien vastuulla ja yksittäiset ominaisuudet vaihtelevat selainkohtaisesti (Juntunen, Jalonen ja Luukkainen 2013, s. 1058).

Achilleos ja Kapitsaki (2014) ovat selvittäneet selainten HTML5-tukea mobiililaitteiden osalta. Heidän mukaansa useimmat selainvalmistajat ovat jo lisänneet selaimiinsa tärkeimmät standardin määrittämät toiminnallisuudet, kuten tuen WebSockets-tekniikalle. Toisaalta taas esimerkiksi tuki akunvarauksen tason ilmoittavalle rajapinnalle puuttui lähes kaikista selaimista. Tutkimuksessaan he käyttivät mm. *HTML5test*-sivustoa, joka listaa käytetyn selaimen tukemat HTML5-standardin yksittäiset ominaisuudet. Muiden kuin mobiililaitteiden selainten kehitykseen he eivät ottaneet kantaa.

Taulukossa 1 on listattu tietokoneiden ja pelikonsoleiden Internet-selainten tukemien yksittäisten HTML5-ominaisuuksien pisteytykset (HTML5test 2015), sekä yleisesti selainten hallussa olevat markkinaosuudet (W3Counter 2015). Listaus paljastaa, etteivät selainvalmistajat ole vielä toistaiseksi kyenneet implementoimaan kaikkia standardin sisältämiä toiminnalli-

suuksia. Selainten välillä on myös suuria eroja, eivätkä varsinkaan pelikonsoleiden selaimet tue kuin korkeintaan puolta kaikista määritellyistä ominaisuuksista. Näin ollen kaikkia HTML5-standardissa listattuja teknisiä toiminnallisuuksia ei voida vielä välttämättä pitää oletusarvoisesti itsestään selvinä. HTML5test (2015) -sivuston tallentamat historiatiedot selainten pisteytyksistä viittaavat kuitenkin siihen, että kaikki Internet-selaimet ovat hiljalleen lisäämässä HTML5-tukeaan uusien versioiden myötä.

Internet-selain	HTML5-tuen pisteytys	Markkinaosuus
Google Chrome 39	501 / 555	43,7 %
Internet Explorer 11	336 / 555	16,3 %
Safari 8.0	396 / 555	15,9 %
Mozilla Firefox 35	449 / 555	14,5 %
Opera 26	497 / 555	3,1 %
Nintendo Wii U	265 / 555	
Sony Playstation 4	328 / 555	
Xbox One	281 / 555	
Nintendo New 3DS	311 / 555	
Nvidia SHIELD	484 / 555	
Sony Playstation Vita	309 / 555	

Taulukko 1. Selainten HTML5-ominaisuuksien pisteytys eri laitteilla.

Pelien kannalta ääniin ja grafiikkaan liittyvät HTML5-ominaisuudet ovat kriittisimpiä. Curran ja George (2012) listaavat yhdeksi tärkeäksi ominaisuudeksi mm. tuen kokoruudun tilalle. Heidän mukaansa esimerkiksi 3D-selainpelit eivät voi saavuttaa täyttä pelikokemusta, jos kokoruudun tukea ei ole saatavilla. HTML5-standardi kyllä määrittelee rajapinnan koko näytön tuelle, mutta sen implementointi eri laitteilla voi olla vielä puutteellinen.

HTML5-standardi ei määrittele erityistä äänitiedostojen formaattia, sillä selainvalmistajat eivät standardia valmisteltaessa päässeet yksimielisyyteen yksittäisen formaatin valinnasta (Curran ja George 2012, s. 38). Eri selainvalmistajat ovatkin päätyneet eri ratkaisuihin, eivätkä selainten tukemat ääniformaatit ole täysin yhteneviä toistensa kanssa. Gawley, Barr ja Barr (2012, s. 196) toteavat, että yleisimmät selainten tukemat äänitiedostomuodot ovat *MP3*

ja *Ogg*. Heidän mukaansa tilanne on tekniikan kannalta hyvin valitettava, sillä selainsovellusten tulisi sisällyttää käytännössä molemmat näistä tiedostomuodoista, jotta sovellusten äänet toimisivat varmasti kaikilla selaimilla.

4.5 Suorituskyky

Jotta HTML5 voitaisiin ottaa täysipainoisesti käyttöön peleissä, on yksi tärkeimmistä kysymyksistä sen suorituskyky. Pelit ovat aina olleet tunnetusti erittäin vaativia laskentatehon, muistinkäytön ja graafisten ominaisuuksien suhteen. Käytetyllä ohjelmointikielellä on myös tässä suhteessa hyvin tärkeä rooli. Anttonen ym. (2011, s. 805) muistuttavat, että vielä vähän aikaa sitten selainten JavaScript-toteutukset olivat erittäin hitaita, mutta tilanne on muuttunut viime vuosina merkittävästi.

Peleissä tarvitaan usein mm. normaalia laskentaa, jonka tehokkuus on pelien kannalta hyvin oleellista. Khan ym. (2014) ovat tutkineet JavaScriptin suorituskykyä numeerisessa laskennassa ja tulokset ovat hyvin rohkaisevia. Tutkimuksen mukaan JavaScript-toteutukset olivat monissa mittauksissa kilpailukykyisiä natiivien C-kielisten toteutusten kanssa. Pahimmissa tapauksissa JavaScript-versio oli C-kielistä toteutusta kolme kertaa hitaampi, mutta tämän voitaneen heidän mukaansa olettaa johtuvan lähinnä Internet-selaimeen sisälletyn JavaScript-tulkin optimoinnin puutteista.

Edellä mainitussa tutkimuksessa JavaScriptin laskennan tehokkuutta oli mahdollista parantaa käyttämällä *asm.js*-ohjelmointikieltä, jolla voidaan ohjelmoida hyvin nopeita selainsovelluksia. Saavutettu nopeushyöty perustuu asm.js-kielen syntaksiin, joka pakottaa ohjelmatoteutukset sellaiseen muotoon, mikä on Internet-selaimelle normaalia JavaScript-toteutusta paremmin optimoitavissa (Asm.js 2015). Kieli ei ole JavaScript-kielen varsinainen laajennos, kuten TypeScript, vaan pikemminkin hyvin rajattu versio JavaScriptin sisältämistä laskennan perustoiminnallisuuksista. Se toimii näin ollen kaikilla selaimilla aivan kuten JavaScript ja sitä voitaneen hyödyntää peleissä erityisesti optimoitaessa suorituskyvylle kriittisiä ohjelmaosioita.

JavaScriptin nopeus ei HTML5-pelien kannalta ole ainoa suorituskykyyn vaikuttava tekijä, sillä myös grafiikan piirron tulee olla riittävän optimoitua, jotta pullonkauloja tehokkuu-

delle ei syntyisi (Anttonen ym. 2011, s. 805). Laitteistokiihdytyksen taso ei 3D-grafiikan tapauksessa ole Nazarov ja Galletly (2013) mukaan vielä täysin ongelmatonta ja WebGL vaatii paljon kehitystyötä, ennen kuin se on heidän mukaansa vertailukelpoinen perinteisten pöytäsovellusten kanssa. He myös mainitsevat, että WebGL:n suorituskyky heikkenee selvästi, mikäli käytössä on suuri määrä 3D-objekteja ja esimerkiksi fysikaalista laskentaa. Li ja Bao (2014) ovat tutkineet yleisesti HTML5:n piirtonopeutta ja verranneet canvas-piirtoa SVG-piirtoon. Tulosten mukaan canvaksen avulla toteutettu bittikarttagrafiikka oli selkeästi suorituskykyisempi objektimäärien kasvaessa.

Aho ym. (2012) ovat tutkineet WebGL ja WebCL -tekniikoiden tuomia hyötyjä mobiililaitteilla. Tutkimuksen mukaan näiden tekniikoiden laitteistokiihdytyksen mukanaan tuomat edut ovat selkeitä ja nopeuttavat sovelluksia, joissa tekniikoita on ylipäätään mahdollista hyödyntää. WebGL:n osittaisena ongelmana on, että se vaatii laskennan tekemistä GPU:lla, mikä mobiililaitteilla on usein heikkotehoinen (Curran ja George 2012, s. 34). Myöskään vanhempien pöytätietokoneiden näytönohjaimet eivät kykene tukemaan kaikkia WebGL:n yksittäisistä 3D-ominaisuuksista (Nazarov ja Galletly 2013, s. 24). Tämä voi osaltaan ainakin toistaiseksi rajoittaa WebGL:n mukanaan tuomia hyötyjä. Curran ja George (2012) kuitenkin muistuttavat, että jatkossa uudet laitteet kykenevät melko varmasti hyödyntämään WebGL:ää täysipainoisesti.

HTML5:n suorituskyky ei ole täysin yksiselitteinen, sillä siihen vaikuttavat niin käytetty laitteisto, Internet-selain, kuin itse sovelluksen toteutus. Varsinkin vanhemmat laitteet kärsivät WebGL-tuen puuttumisesta, mikä voi tarkoittaa merkittäviä suorituskykyongelmia, sillä grafiikan piirto toteutetaan tällöin CPU:lla GPU:n sijaan (Ding ym. 2012). CPU:n käyttö grafiikan piirtoon tarkoittaa myös, että tuo laskentateho on suoraan pois muista pelien tarvitsemista osista. JavaScript on joka tapauksessa jo itsessään saavuttamassa sellaisen tehokkuuden, joka riittänee moniin peleihin. 2D-piirron suorituskyky ei välttämättä sekään ole HTML5:llä enää suuri ongelma, vaan suorituskyvyn esteeksi nousevat pikemminkin laitteisto- ja selaintuen puuttuminen.

4.6 Sovelluskehitys

HTML5-sovellusten yleiset vahvuudet ja heikkoudet pätevät suoraan myös pelitoteutuksiin. Normaalit HTML5-sovellukset voivat kuitenkin erota pelitoteutuksista monin osin, sillä pelit käyttävät usein esimerkiksi vain yhtä HTML-elementtiä, eivätkä välttämättä sisällä merkittävästi CSS ja HTML -rakenteita. Seuraavassa esiteltyjen kirjallisuuslähteiden tuloksia ei siis voida aivan suoraan rinnastaa HTML5-pelien tuloksiin, mutta ne toimivat joka tapauksessa yleisenä viitteenä HTML5:n nykytilasta.

Gawley, Barr ja Barr (2012) ovat toteuttaneet jo olemassa olevan natiivisovelluksen uudestaan HTML5:n avulla ja verranneet näitä sovelluksia sekä niiden kehitysprosesseja toisiinsa. Tutkimuksen mukaan HTML5-sovellus toimi pääsääntöisesti esikuvansa mukaisesti, mutta ongelmia oli havaittavissa osittain mobiililaitteiden äänten sekä laitekohtaisten ominaisuuksien kanssa. HTML5-sovellus todettiin suorituskyvyltään heikommaksi, mutta sen levittäminen ja päivittäminen oli huomattavasti yksinkertaisempaa.

Puputti (2012) on tutkielmassaan kehittänyt HTML5-pohjaisen sovelluksen. Tutkimuksessa keskityttiin erityisesti sovelluksen käytettävyyteen ja suorituskykyyn mobiililaitteilla. Tekniikan ongelmiksi mainittiin mm. sovelluksen yhteensopivuus erikokoisille ruuduille ja mobiililaitteiden verkkokäytön rajoitukset. Sovelluksen toteutus oli itsessään onnistunut, eikä sen suorituskyvyssä ollut juuri moitittavaa.

Anttonen ym. (2011) ovat luoneet katsauksen HTML5-teknologian kehitykseen ja arvioineet sen tulevaisuudennäkymiä. Heidän mukaansa tekniikan mukanaan tuomat hyödyt mm. grafiikan osalta ovat erittäin lupaavia ja sovelluskehityksen yleinen tulevaisuus tulee heidän mukaansa olemaan vahvasti sidoksissa niihin. Juntunen, Jalonen ja Luukkainen (2013) ovat tutkineet HTML5:n mahdollisuuksia web-ohjelmistojen toteutukseen mobiililaitteille. Heidän mukaansa huolimatta suorituskyvyn puutteista natiiviohjelmiin verrattuna, tekniikan mahdollistama helppo saavutettavuus ja luontainen monialustaisuus voi olla ratkaiseva tekijä tekniikan väistämättömälle suosiolle.

4.7 Grafiikka- ja pelimoottorit

Tässä luvussa kartoitetaan pelinkehitykseen soveltuvia HTML5-ohjelmistokirjastoja. Tekniikat on rajattu pääosin avoimen lähdekoodin toteutuksiin. Luvussa keskitytään puhtaisiin HTML5-toteutuksiin, eikä hybridiratkaisuja käydä läpi. Luvussa esiteltävät moottorit ovat *HTML5 Game Engines* -sivustolta[2] poimittuja suosituimpia vaihtoehtoja.

Vaikka HTML5 on verrattain uusi tekniikka, on sitä hyödyntäviä ohjelmakirjastoja ehtinyt muodostua jo laaja määrä. Tämän tutkielman puitteissa ei ole mahdollista kartoittaa kaikkia mahdollisia pelimoottoreita ja kirjastoja. On lisäksi muistettava, että koska esimerkiksi avoimien ohjelmistokirjastojen päivitystahti on usein nopeaa, eivät tutkielmassa listatut ominaisuudet välttämättä pidä paikkaansa enää edes muutaman kuukauden aikajänteellä. Tarkoituksena onkin nostaa esille vain muutamia merkittävimpiä moottorivaihtoehtoja ja täsmentää, mitä ne kirjoitushetkellä tarjoavat ja mitä niillä on mahdollista tehdä.

Grafiikkamoottorilla (engl. *rendering engine*) tarkoitetaan tässä tutkielmassa ohjelmakirjastoja, jotka antavat pelien kehittäjille pääsyn graafisiin rajapintoihin, mutta eivät välttämättä tarjoa valmiita ratkaisuja muihin peleissä usein käytetyille toiminnallisuuksille, kuten fysiikan mallinnukseen. Grafiikkamoottoreita voidaan pitää joidenkin pelien tapauksissa täysin käyttökelpoisina pelimoottoreina, sillä kaikki pelit eivät tarvitse grafiikan piirtoa erikoisempia ominaisuuksia. Tarkkaa rajaa pelimoottoreiden ja grafiikkamoottoreiden ominaisuuksien välille on mahdotonta määritellä (Gregory 2014, s. 11). Tässä tutkielmassa tarkka määritelmä ei ole joka tapauksessa tarpeen.

Kaksiulotteiset pelimoottorit käyttävät usein hyväkseen *sprite*-kuvia, joiden avulla voidaan muodostaa mm. animaatioita. Spritet ovat kuvia, jotka sisältävät läpinäkyviä alueita, jolloin niitä voidaan piirtää toistensa sekä muiden kuvien päälle ilman värillistä kuvataustaa. Spritet ovat aina olleet kaksiulotteisten teksturoitujen pelien käytetyimpiä tekniikoita ja niitä hyödynnetään myös tässä tutkielmassa. (Gregory 2014, s. 544.)

Muita pelimoottoreiden hyödyllisiä ominaisuuksia ovat mm. animaatiot, *GUI*-elementit, *partikkeliefektit* ja äänikirjastot. Partikkeliefektit ovat savun, kipinöiden ja muiden vastaavien mallintamiseen tarkoitettu menetelmä. Yksittäiset partikkelit muodostavat kokonaisuutena

2. `http://html5gameengine.com`

efektin, jota voi olla vaikea saavuttaa uskottavasti muilla tavoin. Partikkeleita luodaan ja tuhotaan automaattisesti nk. *emitterillä*. Partikkelit ovat suuren määränsä takia yleensä graafisesti hyvin yksinkertaisia objekteja, mutta niitä voidaan animoida ja partikkeleiden liikettä voidaan mallintaa fysikaalisesti. (Gregory 2014.)

Tässä tutkielmassa tehdään hyvin lyhyt katsaus kolmiulotteiset pelit mahdollistaviin peli- ja grafiikkamoottoreihin. Varsinainen syvällisempi tarkastelu keskitetään 2D-pelimoottoreihin. Syynä tähän ovat pääasiassa tämän tutkielman rajatut resurssit, mutta myös valmiiden löydettävissä olevien HTML5 3D-pelimoottoreiden ilmeinen keskeneräisyys. Lienee selvää, että kolmiulotteisten pelimoottoreiden vaatimukset ovat selvästi kaksiulotteisia verrokkejaan suuremmat ja niiden kehitys yleiskäyttöisiksi vaatii vielä oman aikansa.

Vanhatupa (2011, s. 366) mainitsee, etteivät pelimoottorit olleet vielä tuolloin yleisesti käytössä kaikissa selainpeleissä. Tilanne lienee nyt kuitenkin jo toisenlainen, sillä vaikka selainpelin toteuttaminen alusta alkaen ei ole erityisen vaikeaa, ovat useimmat HTML5-pelimoottorit yksinkertaisia ja nopeita käyttää. Valmiita avoimen lähdekoodin ratkaisuja on yleisesti saatavilla yksinkertaisista grafiikkamoottoreista kokonaisiin pelinkehitysalustoihin, eikä perustoiminnallisuuden uudelleenkirjoittaminen ole pelinkehittäjälle saavutetun hyödyn kannalta välttämättä mielekästä.

Yogya ja Kosala (2014) ovat kartoittaneet mahdollisia avoimen lähdekoodin fysiikkamoottoreita käytettäväksi selainpohjaisissa HTML5-peleissä. Kolmen vertailun perusteella he ovat tulleet siihen tulokseen, että *Bullet.js* ja *Cannon.js* -fysiikkamoottorit ovat täysin kykeneviä erityyppisten pelien toteutuksiin. Näitä fysiikkamoottoreita voidaan helposti hyödyntää esimerkiksi luvussa 4.7.5 esiteltävän *Three.js*-grafiikkamoottorin kanssa. Vastaavia fysiikkatoteutuksia HTML5-pelimoottoreiden rinnalle ovat myös *Ammo.js*, *JigLibJS2.js* ja *PhysiJS.js* (Nazarov ja Galletly 2013, s. 22–23).

4.7.1 EaselJS

EaselJS on 2D-grafiikkamoottori, joka pyrkii tarjoamaan toiminnallisuuksia canvas-elementille piirrettävien objektien käsittelemiseksi. EaselJS ei siis ole varsinaisesti pelimoottori, mutta tarjoaa tarvittavan perustoiminnallisuuden tietyille syötelaitteille, tekstille ja grafiikan piir-

tämiselle. EaselJS on julkaistu MIT-lisenssillä. EaselJS:n rinnalle on olemassa muita ohjelmistokirjastoja, joiden avulla sen toiminnallisuutta voidaan laajentaa pelikehitystä varten. Tällaisia kirjastoja ovat mm. *PreloadJS*, *SoundJS* ja *TweenJS*, joista esimerkiksi SoundJS lisää tuen ääniä varten. EaselJS:n API on samankaltainen Flashin kanssa mikä voi helpottaa moottorin käyttöönottoa Flashiin tottuneiden kehittäjien osalta. (EaselJS 2015.)

4.7.2 MelonJS

MelonJS on MIT-lisenssillä kehitetty 2D-pelimoottori, joka tukee laajasti erilaisia toiminnallisuuksia. Se sisältää mm. fysiikkamoottorin, partikkeliefektejä sekä tuen äänille ja kosketusnäytöille. MelonJS pyrkii olemaan suhteellisen kevyt toteutus eikä yritä sisällyttää liian raskaita toiminnallisuuksia. Pelimoottorin dokumentaatio on saatavilla luokkadokumentaationa. (MelonJS 2015.)

4.7.3 Pixi.js

Pixi.js on avoimen MIT-lisenssin alla julkaistu HTML5-grafiikkamoottori, joka tukee 2D-objektien piirtoa sekä muita graafisia perustoiminnallisuuksia. Pixi soveltuu jo sellaisenaan pelikehitykseen, mutta ei kuitenkaan tarjoa varsinaisia pelillisiä ominaisuuksia. Tämä voi olla toivottua silloin, kun fysiikkamoottori ja muu toiminnallisuus on tarkoitus tuoda joka tapauksessa muista pelikirjastoista. Pixi tukee automaattisesti WebGL-kiihdytystä, mikäli laitteisto sen sallii. WebGL:n avulla on myös mahdollista käyttää erilaisia graafisia tehosteita jotka mahdollistavat esimerkiksi kuvan sumentamisen tai keinotekoisen pikselöinnin. Pixille on luokkadokumentaatio, jonka kautta voi tarvittaessa nähdä myös moottorin lähdekoodin. Pixi päivittyy lähes kuukauden välein ja sen kehitys on aktiivista. (Pixi.js 2015.)

4.7.4 Phaser

Phaser on Pixi.js:n laajennos joka tarjoaa lisätoiminnallisuuksia erityisesti pelejä varten. Kirjoitushetkellä Phaser tarjoaa mm. kolme erityyppistä fysiikkamoottoria, tilemap-kartat sekä tuen erityyppisille syötelaitteille. Phaserin dokumentaatio sisältää Pixin tavoin luokkadokumentaation, mutta Phaserille on myös hyvin laaja valikoima valmiita esimerkkejä ja koko-

naisia pelitoteutuksia, joita on mahdollista käyttää toteutuksen tukena. Phaser on avointa lähdekoodia ja julkaistu MIT-lisenssillä. Se tukee suoraan TypeScript-kielen käyttöä lähdekoodin kirjoittamiseksi. Phaserin lähdekoodin suurimmat versiopäivitykset ovat tapahtuneet keskimäärin noin 3–4 kuukauden välein. Kirjoitushetkellä pelimoottorista kehitetään versiota numero 3, jonka luvataan tuovan paljon uusia ominaisuuksia ja suorituskyvyn parannuksia. (Phaser 2015.)

4.7.5 Three.js

Three.js on MIT-lisenssillä kehitetty grafiikkamoottori 3D-piirtoa varten. Three pyrkii yksinkertaistamaan kolmiulotteisten objektien piirtämisen ja liikuttamisen. Threen avulla voidaan hallita kolmiulotteisia objekteja, valoja, materiaaleja ja kameroita. Moottori tukee WebGL-kiihdytystä ja SVG-vektorigrafiikkaa. Se ei sisällä juurikaan pelikehitykseen suunnattuja ominaisuuksia ja kokonaisten pelien kehittäminen sillä voi olla vielä työlästä. Threen versio päivittyy noin kolmen kuukauden välein. (Three.js 2015.)

4.7.6 Turbulenz

Turbulenz on täysipainoinen pelimoottori, joka tukee kaksi- ja kolmiulotteista grafiikkaa. Se sisältää hyvin laajan valikoiman niin fysiikkamoottorin, äänikirjaston kuin moninpelin toteutukseen tarvittavia toiminnallisuuksia. Turbulenz mahdollistaa tarvittaessa pelien julkaisemisen palveluidensa kautta ja ottaa pelin myynnistä sovelluskauppojen tapaan 30 % osuuden. Turbulenz oli aiemmin maksullinen, mutta on nyt julkaistu MIT-lisenssillä. Ohjelmistokirjasto päivittyy noin kaksi kertaa vuodessa. (Turbulenz 2015.)

4.8 Pelinkehitysalustat ja työkalut

Myös HTML5:lle on jo olemassa pelinkehitysalustoja, joihin tehdään tässä luvussa nopea katsaus. Pelinkehitysalustoilla on paikkansa myös HTML5-peleissä, sillä vuosien saatossa hyvin suosituksi osoittautunut Adobe Flash on menestynyt osittain juuri kehitystyökalujensa ansiosta (Curran ja George 2012, s. 33). Monet nykypäivän pelinkehittäjästä eivät ole välttämättä koskaan ohjelmoineet pelejä suoraan, vaan ovat aina käyttäneet graafisia työkaluja ja

yksinkertaisia kehitysalustan skriptikieliä pelien luomiseksi.

Pelinkehitys vaatii usein osakseen joitain erillisiä työkaluja, ja avoimen lähdekoodin pelimoottorit ovat luoneet rinnalleen joukon erilaisia apusovelluksia. Tässä luvussa esitellään muutamia työkaluja, joiden tuottamaa dataa voidaan hyödyntää suhteellisen helposti HTML5-pelimoottoreissa. Tavanomaisimpia apusovelluksia, kuten kuvan- tai äänenkäsittelyohjelmia ei ole syytä käydä tässä tutkielmassa läpi. Esiteltävät työkalut on poimittu edellisessä luvussa esiteltyjen pelimoottoreiden dokumentaatioista, joissa suositellaan juuri kyseisten työkalujen käyttöä kehityksen aikana.

4.8.1 Construct 2

Construct 2 on 2D-pelinkehitysalusta, jonka avulla on mahdollista luoda HTML5-pelejä täysin ilman lähdekoodin kirjoittamista. Kehitystyökalusta on saatavilla kolme eri versiota, joista kaksi on maksullisia. Construct 2 sisältää hyvin laajan määrän toiminnallisuuksia, kuten fysiikan, tekoälyn reitinetsinnän, erikoisefektejä, mahdollisuuden pelin tallentamiseen ja tuen moninpelien toteuttamiselle. Kehitystyökalun avulla pelin voi myös julkaista eri kanavien kautta (mm. sovelluskaupat) ja olemassa on myös mahdollisuus kääntää toteutus hybridisovellukseksi. Dokumentaatio on hyvin kattava ja sisältää myös paljon esimerkkejä. Kehitysalustan ilmaisversio on ominaisuuksiltaan rajoitettu ja sillä ei voi esimerkiksi toteuttaa kaupallisia julkaisuja. (Construct 2 2015.)

4.8.2 MightyFingers

MightyFingers on avoimen lähdekoodin HTML5-pelinkehitysalusta, joka pohjautuu luvussa 4.7.4 esiteltyyn Phaser-pelimoottoriin. Muiden pelinkehitysalustojen tavoin MightyFingersissä on graafinen työkaluympäristö, jolla esimerkiksi pelin objekteja voidaan helposti lisätä ja kontrolloida. MightyFingers pyrkii yhdistämään kaiken 2D-peleihin tarvittavan samaan työkaluun. Kehitysalusta itsessään toimii HTML5:llä, mikä tarkoittaa, ettei sitä tarvitse asentaa ja sitä voidaan käyttää kaikilla laitteilla. (MightyFingers 2015.)

4.8.3 Grunt

Grunt on automaatiotyökalu (engl. *task runner*), jota voidaan pelin julkaisun yhteydessä käyttää automatisoimaan esimerkiksi JavaScript-tiedostojen yhdistäminen, kommenttien poistaminen ja kansiorakenteen yksinkertaistaminen. Nämä toimenpiteet ovat hyvin tavallisia HTML-pelejä kehitettäessä, sillä tiedostojen määrän vähentäminen ja tiedostokoon pienentäminen ovat tärkeitä pelin verkon yli lataamisen nopeuttamiseksi. Normaalisti tämä työ tehdään yksittäisinä komentoina komentorivin kautta tai *shell*-skripteinä. Yksittäisten komentojen määrä voi kuitenkin kasvaa nopeasti suureksi, eikä niiden hallinta ole välttämättä enää helppoa. Gruntille on tarjolla hyvin laaja valikoima erilaisia apuohjelmia, jotka suorittavat edellä mainittuja yksittäisiä komentoja. Grunt vaatii toimiakseen *Node.js*-sovelluksen, jonka avulla se myös ylläpitää automaattisesti käytettyjen apuohjelmien asennusta. Grunt-työkalua käyttävät mm. Adobe, *jQuery* ja *WordPress*. (Grunt 2015.)

4.8.4 TexturePacker

TexturePacker on kuvatyökalu, jonka avulla peleissä käytettäviä kuvia voidaan pakata *spritesheeteiksi* ja optimoida samalla käytetty tiedostomuoto. Spritesheetit ovat kuvakollaasimaisia kokoelmia, jotka sisällyttävät useita kuvia yhteen kuvatiedostoon ja vähentävät näin tiedostomääriä. Selainpeleissä kuvatiedostojen koko on tärkeää verkon yli lataamisen vuoksi, sillä pelin avaaminen voi kuvien määrän kasvaessa hidastua merkittävästi. Käytännössä myös monet pelien kuvista on usein mahdollista tallentaa alkuperäistä huomattavasti pienempään formaattiin, säilyttäen tästä huolimatta kuva visuaalisesti muuttumattomana. (TexturePacker 2015.)

Spritesheetien käyttäminen mahdollistaa myös pelien *sprite*-kuvissa olevien läpinäkyvien alueiden lomittamisen, joka vähentää kuvatiedostojen vaatimaa kokoa entisestään. TexturePackerin avulla kuvista voidaan luoda myös nk. *atlas*-tiedosto, johon viittaukset spritesheetin yksittäisten kuvien tietoihin voidaan tallentaa. Tätä tiedostoa voidaan käyttää suoraan useissa pelimoottoreissa, jolloin kuvat voidaan määritellä helposti lähdekoodin ulkopuolelle. Kuvatiedostojen koko voi vaikuttaa suoraan myös pelin suorituskykyyn. TexturePackerin ilmaisversio on toiminnoiltaan osittain rajoitettu. (TexturePacker 2015.)

4.8.5 Tiled

Tiled on useiden pelimoottoreiden tukema avoimen lähdekoodin *tilemap*-editori, jolla on mahdollista luoda peleihin laajoja ja monipuolisia tiilipohjaisia karttoja. Tiled tukee spritesheetejä, joiden sisältämiä yksittäisiä kuvia voidaan asetella taulukkomaisesti rinnakkain, jolloin saadaan aikaan kaksiulotteinen ”tiilistä” koostettu suurempi kuvakokonaisuus. Tiled tallentaa lopulta kuvakokonaisuuden indeksoituna taulukkona, jonka numerot vastaavat annetun spritesheetin yksittäisiä kuvia (tiiliä). Tiled sopii erityisesti *Super Mario* -tyyppisten pelien karttojen luomiseen. Tiilikarttojen etuna on yksittäisten kuvien toistuvuus ja tallennusmenetelmä, joka vähentää merkittävästi pelin muistinkäyttöä ja vaadittua tallennustilaa. Tiilikarttoja on mahdollista myös helposti dynaamisesti muuttaa pelin aikana. (Tiled 2015.)

4.9 Pelien kehitys ja markkinointi

HTML5-pelikehityksen avuksi on olemassa runsaasti kaupallisia kirjajulkaisuja, joista monet kuvaavat mm. pelien toteutusta esimerkkien kautta. Myös valmiita pelitoteutuksia on Internetissä paljon, eivätkä HTML5-pelit itsessään ole mikään uusi käsite. Varsinaista tieteellistä tutkimusta HTML5-peleistä ei ole kuitenkaan merkittävästi saatavilla. Monialustatutkimuksen tapaan HTML5:stä kertova tieteellinen lähdekirjallisuus on myös usein hyvin mobiilipainotteista.

Weeks (2014) on tutkimuksessaan toteuttanut yksinkertaisen kaksiulotteisen pelin HTML5:llä ja kehittänyt sen ympärille myös oman pelimoottorinsa. Pelin grafiikan piirto on toteutettu suoraan canvakselle, eikä Weeks ole työssään havainnut erityisiä ongelmia. HTML5:tä pidettiin tutkimuksessa varsin toimivana vaihtoehtona toteutetulle pelille. Yksinkertaisten 2D-pelien toiminnallisuus ei HTML5:ssä ole välttämättä ongelma, sillä canvaksen piirto-ominaisuudet antavat Weeksin mukaan hyvin tehokkaan tavan piirtää varsinkin alkeellisia 2D-objekteja. Kolmiulotteisten pelien vaatimukset ovat taas kaksiulotteisiin verrattuna jo hyvin toisenlaiset.

Curran ja George (2012) ovat tehneet kattavan katsauksen HTML5:n soveltuvuudesta erityisesti kolmiulotteiseen mobiilipelikehitykseen. Tutkimuksessa esitellään HTML5 sekä siihen liittyvät teknologiat ja verrataan niitä yleisimpiin vaihtoehtoisiin peliteknologioihin, ku-

ten Adobe Flash 3D:een ja Unity 3D:een. Huolimatta WebGL:n positiivisista puolista, ei tekniikka ollut heidän mukaansa kuitenkaan vielä tuolloin saavuttanut näiden vaihtoehtoisten menetelmien tasoa kolmiulotteisten pelien toteutukseen. He kuitenkin huomauttavat, että kaksiulotteiset selaimessa pelattavat moninpelit ovat vasta tekemässä läpimurtoaan, eikä tekniikan voida olettaakaan kykenevän vielä vaativampiin kolmiulotteisiin peleihin.

HTML5:n ehdoton vahvuus on sen todellisessa monialustaisuudessa. Web-teknologioiden kehitys on lisäksi tuomassa sitä vakaasti kohti muiden tekniikoiden suorituskykyä. Torrente ym. (2014, s. 114) ovat sitä mieltä, että ennen pitkää HTML5:tä voidaan oletettavasti käyttää pelaamiseen kaikilla niillä laitteilla, joissa vain on pääsy Internetiin. Vanhatupa (2011, s. 367) vie ajatuksen tätäkin pidemmälle ja tekee olettaman, että kaikki videopelit siirtyvät ennen pitkää verkkoon, eikä niitä asenneta enää lainkaan paikallisille laitteille. Ennen kuin tämä on kuitenkaan mahdollista, on tekniikan ohella myös HTML5-pelien julkaisukanavien muodostuttava riittävän tehokkaiksi.

Luvussa 3.6 todettiin jo lyhyesti, kuinka HTML-pelien markkinointi ja maksumallit siirtyvät kehittäjien itsensä implementoitaviksi, mikäli käytössä on oma palvelin. Myös Gawley, Barr ja Barr (2012, s. 204) pohtivat, että HTML5-sovellusten levitys ja markkinointi on totuttuihin tapoihin verrattuna ratkaisevasti erilainen, sillä tarvetta perinteisille sovelluskaupoille ei enää ole. Heidän mukaansa kehittäjien tuleekin siirtyä täysin toisenlaiseen liiketoimintamalliin, mikäli HTML5-tekniikkaa halutaan käyttää maksullisten sovellusten luomiseen. Vanhatupa (2011, s. 367) toteaa, että yksi ratkaisu selainpelien rahoitusmallille voisi olla mm. kuukausimaksut.

Sovelluskauppojen toimintamallilla on osaltaan omat hyvät puolensa, kuten keskitetty maksujärjestelmä, luotettavuus ja markkinointikanavat. Luvussa 2.3 käytiin läpi mobiililaitteille suunnattuja sovelluskauppoja ja niiden toimintaa. Nämä sovelluskaupat eivät sellaisenaan sovellu puhtaasti web-tekniikoilla kehitetyille peleille, sillä esimerkiksi Apple voi kauppansa ehtojen mukaan kieltää web-sovelluksien julkaisun (Heitkötter, Hanschke ja Majchrzak 2013, s. 130). Laitevalmistajien omien sovelluskauppojen ehtojen ongelmana voivat olla myös pelin sisällön rajoitteet. Ratkaisuna tähän voisi olla puhtaasti HTML5-peleille osoitetut sovelluskaupat.

Yksinomaan HTML5-sovelluksia varten kehitettyjä sovelluskauppoja on jo ilmestynyt muutamia (esim. *OpenAppMkt*), mutta mikään niistä ei ole noussut erityisen suosituksi. Myös useimmat selainvalmistajat ovat luoneet omat sovelluskauppansam, mutta näiden ongelmana on julkaisujen rajoittuminen vain tietyille selaimille. *Mozilla Labs Apps* oli yksi kunnianhimoisempi Mozilla-yhtiön projekti, joka olisi mahdollistanut HTML5-sovellusten julkaisun ja myymisen mm. *PayPalin* avulla kaikille laitealustoille. Projekti kuitenkin haudattiin kaikessa hiljaisuudessa vuonna 2014.

On selvää, että HTML5 on vasta hiljalleen tekemässä todellista läpimurtoaan, eikä tekniikka ole suinkaan täydellistä. Torrente ym. (2014, s. 114) ovat sitä mieltä, että älypuhelimien ja tablettien HTML5-tuki on toistaiseksi jopa liian puutteellinen peleille. Monialustaisuuden ja avoimen standardin hyödyt ovat kehittäjien kannalta houkuttelevia, mutta yhteensopivuus- ja suorituskykyongelmat voivat silti haitata toteutuksen toimivuutta. Yksittäisten laitteiden hätäratkaisuna ongelmallisille HTML5-peleille voivat – ehkä hieman yllättäen – olla hybridiratkaisut, kuten *CocoonJS* ja *Intel XDK*.

CocoonJS on hybridityökalu, joka mahdollistaa puhtaiden HTML5-sovellusten muuntamisen laitekohtaiseksi ratkaisuksi, jotta se voitaisiin julkaista mm. sovelluskaupoissa. CocoonJS on jo monien HTML5-pelimoottoreiden valmiiksi tukema ratkaisu ja esimerkiksi Pixi.js-pelitoteutukset on mahdollista muuntaa sen avulla hybridisovelluksiksi. CocoonJS sisältää myös pelin julkaisuun ja sisäiseen maksamiseen liittyviä toiminnallisuuksia. (Ludei 2015.)

HTML5-pelien lopullisena tavoitteena voitaneen pitää niiden ajamista yksinomaan Internet-selaimilla, mutta mikään ei tietenkään estä väliaikaisten hybridiratkaisujen käyttöä. Varsinkin maksulliset HTML5-pelit voivat vielä toistaiseksi hyötyä perinteisistä sovelluskaupoista. Ilmaispelien kehittäjät saattavat jo nyt pystyä julkaisemaan pelejään onnistuneesti selaimille ja kehittämään suorituskykyisiä toteutuksia, jotka toimivat jo riittävän useilla laitteilla ongelmitta.

5 Tutkimuksen toteutus

Tämä tutkimus on luonteeltaan konstruktiivinen, jossa tarkoituksena on ratkaista käytännön ongelma aiempaan teoriatietoon pohjautuen. Järvinen ja Järvinen (2004, s. 109–110) kuvaavat konstruktiota kysymyksellä: voidaanko haluttu lopputulos saavuttaa annetuilla resursseilla? Vastaavasti tämän tutkielman keskeisenä tutkimuskysymyksenä on: voidaanko HTML5:n avulla toteuttaa monialustaisia 2D-pelejä, jotka toimisivat ongelmitta sekä mobiili- että pöytätietokoneilla? Ongelmilla tarkoitetaan tässä yhteydessä yhteensopivuus- sekä suorituskykyongelmia. Lisäksi tutkielmassa halutaan selvittää, mitä ovat HTML5:n mukanaan mahdollisesti tuomat hyödyt ja haitat pelikehityksen näkökulmasta. Tutkimusongelma pohjautuu luvussa 6 kuvattavan monialustaisen peliprojektin asettamiin vaatimuksiin.

Hevner ym. (2004, s. 82–90) esittävät suunnittelutieteen (engl. *design science*) mallin, jolla IT-teknisen artefaktin, eli uuden innovaation suunnittelu-, toteutus- ja arviointiprosessit käydään läpi. Esitelty malli on ongelmanratkaisuprosessi, joka alkaa artefaktin suunnittelulla ja siihen liittyvän liiketoimintaongelman esiintuomisella. Tämän jälkeen artefaktia arvioidaan ja tuloksena luodaan uutta tutkimuksellista tietoa. Lopuksi itse tutkimuksen luotettavuutta arvioidaan ja tulokset julkaistaan. Esitetty liiketoimintaongelma on keskeinen osa suunnittelutieteen tarkoitusta ja keskeinen syy sille, miksi tutkimus ylipäätään tehdään.

Tässä tutkielmassa edellä kuvattua mallia käytetään viitteellisenä ohjeena tutkimukselle. Samaa mallia käytetään yleisesti myös muualla pelitutkimukseen suuntautuvien tutkimusmetodien pohjana (esim. Olsson 2015). Koska suunnittelutieteellisen tutkimuksen on aina perustuttava aikaisemmalle tiedolle ja käsitteille (Hevner ym. 2004, s. 80), on tämän tutkimuksen teoreettinen pohja luotu narratiivisen kirjallisuuskatsauksen kautta. Tätä osuutta tutkimuksesta edustavat tutkielman tätä edeltäneet luvut. Teoreettista tietoa on tutkielman konstruktiossa hyödynnetty monialustaisen peliteknologian sekä työkalujen kartoituksessa ja valinnassa.

Kirjallisuuskatsauksen aineisto kerättiin pääosin Internetin sähköisistä tietokannoista. Katsauksessa keskityttiin erityisesti vuoden 2010 ja sen jälkeiseen aineistoon. Vanhempaa kirjallisuutta ei katsottu hyödylliseksi, sillä tutkittavana oleva teknologia kehittyy ja muuttuu

hyvin nopeasti. Maksullista tai muutoin saavuttamatonta materiaalia ei myöskään sisällytetty tutkimukseen tutkielman rajattujen resurssien vuoksi. Samasta syystä myös systemaattisen kirjallisuuskatsauksen katsottiin olevan liian raskas menetelmä tähän tutkielmaan.

Luvussa 6 on kuvattu kahden HTML5-pelin taustatiedot, vaatimukset, suunnittelu sekä toteutus. Toteutusta varten valittiin erikseen projektissa käytettävä HTML5-pelimoottori, jonka valintaprosessi on verrattavissa ns. perinteisen valmisosan hankintaan. Tätä hankintaprosessia kuvaavat mm. Järvinen ja Järvinen (2004, s. 111), vaikka eivät sille suurta tieteellistä arvoa varsinaisesti annakaan. Heidän mukaansa hankintaprosessi alkaa mahdollisten kandidaattien valinnalla ja vertailulla, jonka jälkeen tehdään hankintapäätös sekä hyväksymistestaus. Tutkielman pelimoottorin osalta tämä prosessi käydään läpi luvussa 6.4.

Näitä edellä mainittuja pelejä ja niiden kehitystä arvioidaan luvussa 7. Pelien arvioinnissa keskitytään suorituskyvyn ja luotettavuuden arviointiin, jotka esim. Hevner ym. (2004, s. 85) esittävät IT-artefaktin eräiksi keskeisiksi tarkastelun kohteiksi. Pelejä testataan koululaisten sekä kehityksessä mukana olleen tutkimusryhmän toimesta ja näissä testauksissa esiintulleita mahdollisia laadullisia sekä käytettävyyteen liittyviä ongelmia nostetaan esille. Myös pelien kehityksen aikana esiin tulleita ongelmia ja muita opittuja tietoja käydään luvussa yksityiskohtaisesti läpi.

Lopuksi pohditaan tutkimuksen luotettavuutta, keskeisiä tuloksia, niiden yleistyvyyttä ja konstruktion soveltuvuutta alkuperäisen ongelman ratkaisuun. Tutkimuksen tuloksissa annetaan viitteitä HTML5:n nykytilasta ja sen mahdollisuuksista monialustaisessa pelinkehityksessä. Tuloksia voivat hyödyntää pelikehittäjät sekä mahdollista jatkotutkimusta tekevät tahot. Kaikkia tutkimuksen tuottamia tuloksia ei voida sellaisenaan yleistää edustamaan koko HTML5-teknologiaa, sillä konstruktion puitteet antavat vastauksia vain tutkimusongelmalle keskeisen peliprojektin osalta. Tutkimuksen yhteenveto annetaan luvussa 8.

6 Pelien suunnittelu ja toteutus

Tässä luvussa kuvataan tutkielman konstruktio, jossa esitellään kehitettävien HTML5-pelien vaatimukset, suunnitelmat ja toteutus. Tutkielman pelit ovat osa Jyväskylän yliopiston Agora Centerin ja Lapsiasiavaltuutetun toimiston yhteistyöhanketta *"Lasten sivut sinne missä lapsetkin"*, jonka osana toteutettiin peliprojekti nimeltään *Sisunmaan Sankarit*. Peliprojektin tarkoituksena on pelillistää *Lasten sivujen*[1] *"Apua moneen ongelmaan"* (lyh. AMO) -osiota.

Alakouluikäisille lapsille suunnattu lasten oikeuksista kertova peli sisältää useita eri aihealueen minipelejä sekä yhden suuremman pelikokonaisuuden. Tähän tutkielmaan valitut kaksi minipeliä eroavat toiminnallisuuksiltaan ja edustavat siten hieman erilaista pelikehityksellistä näkökulmaa ja teknisiä vaatimuksia. Peliprojektin lopullisena tarkoituksena on olla osana uudistuvaa Lasten sivujen -sivustoa, jonka sisältönä ovat lasten oikeudet.

6.1 Sisunmaan Sankarit

Sisunmaan Sankarit on sarjakuvamaisista peleistä vaikutteita ottava lasten seikkailupeli. Pelin tarina keskittyy *Sisunmaahan*, jossa pelaaja voi vapaasti liikkua ja suorittaa löytämiään tehtäviä minipelien muodossa. Pääpeli on graafiselta ulkoasultaan yläviistosta kuvattu maailma, jossa pelaaja liikuttaa luomaansa pelihahmoa (nk. avatar). Peliä aloittaessaan pelaaja saa vapaasti valita pelin vaikeustason. Pääpelistä minipelin löydettyään pelaaja voi pelata sitä vapaasti niin monta kertaa kuin haluaa. Minipelit keskittyvät lasten oikeuksista kertoviin teemoihin.

Minipelin onnistuneesti suoritettuaan pelaaja saa palkintona esineen, jota hän voi käyttää pääpelissä. Tarpeeksi monta minipeliä läpäistyään pelaaja pääsee kokeilemaan loppuhaastetta, jolla itse pääpeli voidaan lopulta voittaa. Pelissä on myös lyhyitä dialogeja, joilla annetaan ohjeita pelaamiseen ja viedään pelin tarinaa eteenpäin.

1. `http://lastensivut.fi`

6.2 Pelisuunnitelma

Sisunmaan Sankarit -projektista tutkielmaan valitut kaksi minipeliä ovat nimeltään *Perhe* ja *Terveys*. Juuri nämä pelit on valittu, koska yhdessä ne koostuvat tekniikoiltaan hyvin tyypillisistä 2D-pelien piirteistä, kuten sprite-grafiikasta, animaatioista, äänistä, fysiikan mallinnuksesta, tekoälystä ja liikkuvasta taustasta. Perhe-peli on tiilikarttapohjainen fysikaalista laskentaa vaativa peli, jossa on liikkuva pelihahmo sekä liikkuva kamera. Lisäksi pelin grafiikka vaatii myös näkyvien objektien asettamista järjestykseen, jotta on mahdollista saada aikaiseksi syvyysvaikutelma yläviistosta kuvatusta pelikentästä. Terveys-pelissä on puolestaan paikallaan oleva pelialue, jossa pelimekaniikka on hyvin keskeisessä roolissa ja peli sisältää paljon animaatioita. Tässä luvussa pelien varsinaiset toiminnalliset vaatimukset esitetään pelisuunnitelmina.

Pelisuunnitelmien lisäksi toiminnallisiin vaatimuksiin voidaan lukea myös pelien mahdolliset myöhemmin laajennettavat ominaisuudet. Peleihin saatetaan esimerkiksi lisätä sosiaalisia elementtejä, kuten yksinkertainen keskustelupalkki ruudun reunaan. Keskustelupalkin avulla pelaaja voi lähettää ennalta määriteltyjä viestejä muille pelaajille ja nähdä ketkä muut ovat pelaamassa. Myös muiden pelaajien hahmojen näkymistä pääpelin aikana ei haluta sulkea toteutusvaihtoehdoilla pois.

6.2.1 Perhe-peli

Perhe-peli on yläviistosta kuvattu luolamainen satunnaisgeneroitu labyrinttipeli, jossa pelaajan tulee etsiä labyrinttiin eksyneet pelihahmot. Taustatarinan mukaan pelihahmot ovat riidelleet keskenään ja pelaaja auttaa selvittämään tilanteen kokoamalla nämä takaisin yhteen. Pelaaja on esitetty Perhe-pelissä samalla tavoin kuin pääpelissäkin. Luolastoon eksyneet hahmot (4 kpl) ovat mangusti-eläimiä, joista yksi seuraa pelaajaa heti peliä aloitettaessa. Luolastossa on myös satunnaisesti liikkuvia hiiriä, jotka pakenevat pelaajaa tämän tullessa liian lähelle. Pelaajan hahmo jättää luolastossa liikkuessaan jalanjälkiä, joita pelaajan on mahdollista käyttää hyväkseen kulkemiensa reittien muistamiseksi.

Peli alkaa suoraan luolaston sisältä, ja heti pelin alussa pelaajalle näytetään info-dialogi, jossa esitetään lyhyt tarina ja neuvotaan etsimään luolastoon eksynyt mangustiperhe takaisin

yhteen. Pelin näkökenttä on rajattu pelaajan ympärille ja se pienenee hiljalleen ajan kuluessa. Näkökenttä esittää pelaajan taskulamppua, johon on mahdollista löytää lisää paristoja luolastosta. Paristojen tai mangustihahmojen löytäminen suurentaa näkökenttää väliaikaisesti. Pelissä on aikaraja, joka on sidottu suoraan näkökentän laajuuteen. Pelin jäljellä oleva aika näytetään pelaajalle tekstinä.

Kun pelaaja löytää labyrinttiin eksyneen hahmon, näytetään onnistumisen merkiksi esimerkiksi partikkeliefekti, jonka jälkeen löydetty hahmo alkaa seurata pelaajaa. Mikäli löydetty hahmo on viimeinen luolastosta etsittävä, peli päättyy ja pelaajalle näytetään onnistumisesta kertova dialogi. Vastaavasti jos pelaajalta loppuu aika, näytetään dialogilla pelaajalle viesti epäonnistumisesta ja peli päättyy. Pelaaja ei voi läpäistä luolaston seiniä, mutta voi joka tapauksessa nähdä kaikki hahmot ja pelin esineet seinien läpi jos ne ovat näkökentässä. Pelaajan askelten äänet ja labyrintin luolamainen taustamusiikki kuuluvat koko pelin ajan.

Peliä ohjataan tietokoneilla joko nuolinäppäimillä tai hiirellä. Mobiililaitteilla pelin ohjaus tapahtuu joko kosketusnäytön painalluksella (engl. *tap*) tai pyyhkäisyllä (engl. *swipe*). Painallus saa pelin hahmon liikkumaan painallusta kohti kunnes painallus päättyy. Pyyhkäisemällä hahmoa liikutetaan pyyhkäisyn suuntaan, ja erityisen nopealla pyyhkäisyllä on mahdollista saada hahmo liikkumaan hetken normaalia nopeammin. Pelaajan liikuttamista kosketusnäytöillä pyyhkäisyn avulla voitaneen pitää Perhe-pelin ensisijaisena pelitapana, sillä Browne ja Anand (2012) esittävät tutkimuksensa perusteella, että syötteiden muuntaminen suoraan vastaaviksi pelimaailman fyysisiksi ominaisuuksiksi voi olla luontevin tapa pelata.

Labyrintistä on tarkoitus pystyä luomaan erikokoisia kenttiä pelin mahdollisen vaikeustason mukaan. Pelin kehitystä on tarkoitus jatkossa mahdollisesti laajentaa erilaisilla luolastosta löytyvillä bonuksilla, kuten lähimmän mangustin olinpaikan ilmoittavalla nuolella tai katoavilla seinillä. Myös ovien sekä muiden vastaavien animoitujen objektien lisääminen pelimaailmaan myöhemmin on oltava mahdollista.

6.2.2 Terveys-peli

Terveys-peli on tyypiltään *Tamagotchia* muistuttava virtuaalilemmikki, jossa pelaajan on tarkoitus hoivata pelin lemmikkiä ja jonka parissa hän voi viettää hetken aikaa. Virtuaalilem-

mikki on pieni kananpoikanen, joka on kipeänä. Pelillä on tarkoitus viestiä, että kaikki ovat joskus sairaana ja kaipaavat hoivaamista. Pelissä ei ole eri vaikeustasoja, eikä peliä voi hävitä. Peliä on tarkoitus pelata useita eri kertoja pääpelin pelaamisen aikana. Jokaisen pelikerran jälkeen lemmikki siirtyy lopulta nukkumaan ja herää vasta, kun pelaaja palaa takaisin pelattuaan esimerkiksi hetken aikaa muita minipelejä. Kun peliä on pelattu riittävän monta kertaa, lemmikki paranee lopulta ja pelaaja saa pelistä suorituspalkinnon.

Peli on hidastempoinen ja siinä on hyvin rauhallinen taustamusiikki koko pelaamisen ajan. Pelin ohjeistus näytetään dialogilla minipeliä ensimmäisen kerran aloitettaessa ja se toimii samalla myös osana pääpelin keskeistä ohjeistusta. Virtuaalilemmikki on aina ruudun keskellä ja sen ympärillä on erilaisia esineitä, joita pelaaja voi käyttää lemmikkiin ”raahaamalla” niitä tämän päälle. Raahaaminen tapahtuu joko hiirellä tai kosketusnäytöllä, kun pelaaja siirtää esinettä pyyhkäisemällä (hiirellä painike pohjassa). Lemmikki viestii tarpeistaan animaatioiden ja äänien avulla. Mikäli pelaaja ei osaa käyttää lemmikin tilaa vastaavaa oikeaa esinettä, annetaan pelaajalle vihje puhekuplan kautta tai kyseistä esinettä korostamalla.

Kaikkia lemmikin eri animaatioita, tiloja ja niiden välisiä siirtymiä ei ole tämän tutkielman puitteissa tarvetta listata yksitellen läpi. Tässä suunnitelmassa annetaankin niistä vain tiivistetty kuvaus. Lemmikin eri tiloja ovat mm. sairas, nälkäinen, kyllästynyt, uninen ja kuumeinen. Animaatiot koostuvat siipien, pään, jalkojen, nokan ja silmien liikuttelusta. Lemmikki voi olla esimerkiksi nälkäinen, jolloin se räpyttää siipiään, aukoo suutaan ja pitää ääntä. Vastaavasti kuumeisena lemmikki on hyvin passiivinen ja pitää esimerkiksi silmiään puoliksi kiinni.

Eri esineet, joita pelaaja voi lemmikille antaa, ovat mm. kuumemittari, tyyny, juomalasi, lelu sekä ruokaa. Lemmikin tilasiirtymät tapahtuvat esineitä käytettäessä. Esimerkiksi uniselle lemmikille tyynyn raahaaminen asettaa sen animaation kautta nukkumaan. Lemmikki voi myös reagoida esineeseen jo raahauksen alkuvaiheessa, mm. ruokaesineen liikuttaminen saa nälkäisen lemmikin räpyttämään siipiään kiivaammin. Jos pelaajan pudottama esine ei kelpaa lemmikille, siirtyy esine automaattisesti takaisin lähtöpaikkaansa ja lemmikki esittää kieltäytymistä osoittavan animaation tai äänen.

6.3 Pelien tekniset vaatimukset

Koska pelit on tarkoitettu lapsille, ei ole mielekästä olettaa, että pelaaja asentaisi ne laitteelleen ennen pelaamista. Lapsilla ei myöskään ole välttämättä edes tarvittavia käyttäjäoikeuksia sovellusten asentamiseksi. Lisäksi lasten käytössä olevat laitteet voivat vaihdella pöytätietokoneista älypuhelimiin. Pelin tulisikin toimia suoraan kaikilla laitteilla mahdollisimman helposti ja tukea myös niiden kaikkia syötelaitteita.

Pelin saatavuuden mahdollistamiseksi luvussa 3 esitetyt asentamista vaativat perinteisemmät monialustaratkaisut eivät käytännössä soveltuneet pelin toteuttamiseen. Lisäksi koska peliä tullaan markkinoimaan pääasiassa web-sivuston kautta, koettiin selainpelitoteutus kaikkein luontevimmaksi. Hybridiratkaisut, kuten Adobe Flash suljettiin pois vaihtoehdoista, sillä ne eivät pääasiassa toimi mobiililaitteilla. Joustavimmaksi toteutusvaihtoehdoksi katsottiin HTML5, sillä se ei mitenkään sido pelin markkinointia ja julkaisua. Tekniikka tiedettiin keskeneräiseksi, mutta sen avulla aikaisemmin toteutetut peliprototyypit todettiin joka tapauksessa pääosin hyvin toimiviksi ja nopeiksi kehittää.

HTML5-tekniikan valinnan myötä peleille asetettiin tekniset vaatimukset ja rajoitteet, jotka pelin ohjelmistokirjastoja valittaessa ja pelejä toteutettaessa on otettava huomioon:

- Pelien tulee toimia kannettavilla ja pöytätietokoneilla.
- Pelien tulee toimia tableteilla ja uusimmilla älypuhelimilla.
- Pelien tulee toimia kaikilla yleisimmillä käyttöjärjestelmillä.
- Pelien tulee toimia uusimpien Google Chrome, Mozilla Firefox, Internet Explorer ja Safari -selainten kanssa.
- Pelien kehitys tulee tapahtua ilmaisilla työkaluilla ja avoimilla lisensseillä.
- Pelejä tulee pystyä pelaamaan kosketusnäytöillä, näppäimistöllä tai hiirellä.
- Pelien suorituskyvyn tulee olla riittävä, eikä se saa haitata pelaamista.
- Pelejä ei tarvitse asentaa niiden pelaamiseksi.
- Pelien minimiresoluutio on 960x540.

Tuettavat laitejärjestelmät tulee rajoittua yleisesti käytössä oleviin laitteisiin, eikä esimerkiksi tukea yli viisi vuotta vanhoille laitteille katsota täysin välttämättömäksi pelien vaatimusten täyttymiselle. Edellä olevien teknisten vaatimusten toteutuminen kuvataan luvussa 7.3.

6.4 Työkalujen valinta

Peleille asetetut toiminnalliset ja tekniset vaatimukset huomioiden kartoitettiin mahdollisia HTML5-pelimoottoreita ja tarvittavia työkaluja. Vertailtavat 2D-pelimoottorit rajattiin luvussa 4.7 esiteltyihin vaihtoehtoihin. Pelinkehitysalustoja ei otettu vertailuun mukaan, sillä kehitystyö haluttiin tehdä puhtaasti lähdekoodia muokkaamalla, eikä toteutusta haluttu rajoittaa tietylle sovellustyökalulle.

Taulukossa 2 on listattu vaihtoehtoiset pelimoottorit ominaisuuksineen. Kaikki vertailluista kehityskirjastoista on julkaistu MIT-lisenssillä ja ovat siten ilmaisia käyttää. Valinnassa painotettiin tarvittavien ominaisuuksien mahdollisimman helppoa käyttöönottoa. Lisäksi ohjelmistokirjaston tuli olla riittävän aktiivisesti ylläpidetty ja hyvin dokumentoitu. Pelimoottoreiden päivitysten ajankohdat löytyivät *github*-palvelusta[2], jossa kaikkien listattujen moottoreiden lähdekoodit on julkaistu.

MelonJS:n WebGL-tuki todettiin toistaiseksi puutteelliseksi, sillä se on suorituskyvyltään dokumentaationsa mukaan canvas-versiota huonompi. EaselJS:n ominaisuuksia olisi listattujen lisäksi ollut mahdollista laajentaa luvussa 4.7.1 esitettyjen lisäkirjastojen avulla ja myös fysiikkamoottorin käyttö on sen yhteydessä mahdollista. Tämä olisi muiden listattujen vaihtoehtojen rinnalla ollut kuitenkin huomattavan joustamaton ratkaisu. Turbulenzin kehitys taas näyttää pysähtyneen, sillä kattavista ominaisuuksistaan huolimatta sen viimeisin versio oli jo yli vuoden vanha.

Phaser katsottiin ominaisuuksien ja vertailun perusteella soveltuvan hyvin projektin vaatimuksiin. Käyttöön otettiin aluksi versio 2.2.0, joka kehityksen aikana päivitettiin viimeisimpään vakaaseen versioon 2.3.0. Pelimoottorin suorituskykyä testattiin aluksi peliprojektin erään minipelin prototyypin avulla, jonka toteutus onnistui vaatimusten mukaisesti. Tämän lisäksi kartoitettiin pelimoottorin ruudunpäivitysnopeuksia eri resoluutioita käytettäessä (kuvattu luvussa 6.6.2). Aputyökaluiksi kehitykseen valittiin Grunt ja TexturePacker, jotka ovat täysin yhteensopivia pelimoottorin kanssa, ja esimerkiksi Grunt-sovellusta käytetään Phaserin lähdekoodin julkaisussa.

2. `http://github.com`

	EaselJS 0.8.0	MelonJS 2.0.2	Phaser 2.3.0	Pixi.js 3.0.2	Turbulenz 0.28.0
A.I. toiminnallisuus					
Animaatiot	×	×	×		×
Fysiikan mallinnus	**	×	×	**	×
Kosketusnäytöt		×	×	×	×
Mobiililaitetuki		×	×	×	×
Partikkeliefektit		×	×		×
Resurssien lataus	**	×	×	×	×
Sprite-tuki	×	×	×	×	×
Tekstin piirto	×	×	×	×	×
Tiilikartat		×	×		
Tween	**	×	×		
WebGL	×	×*	×	×	×
Äänet	**	×	×		×
Versio julkaistu	12/2014	12/2014	3/2015	4/2015	3/2014

*puutteellinen, **helposti laajennettavissa

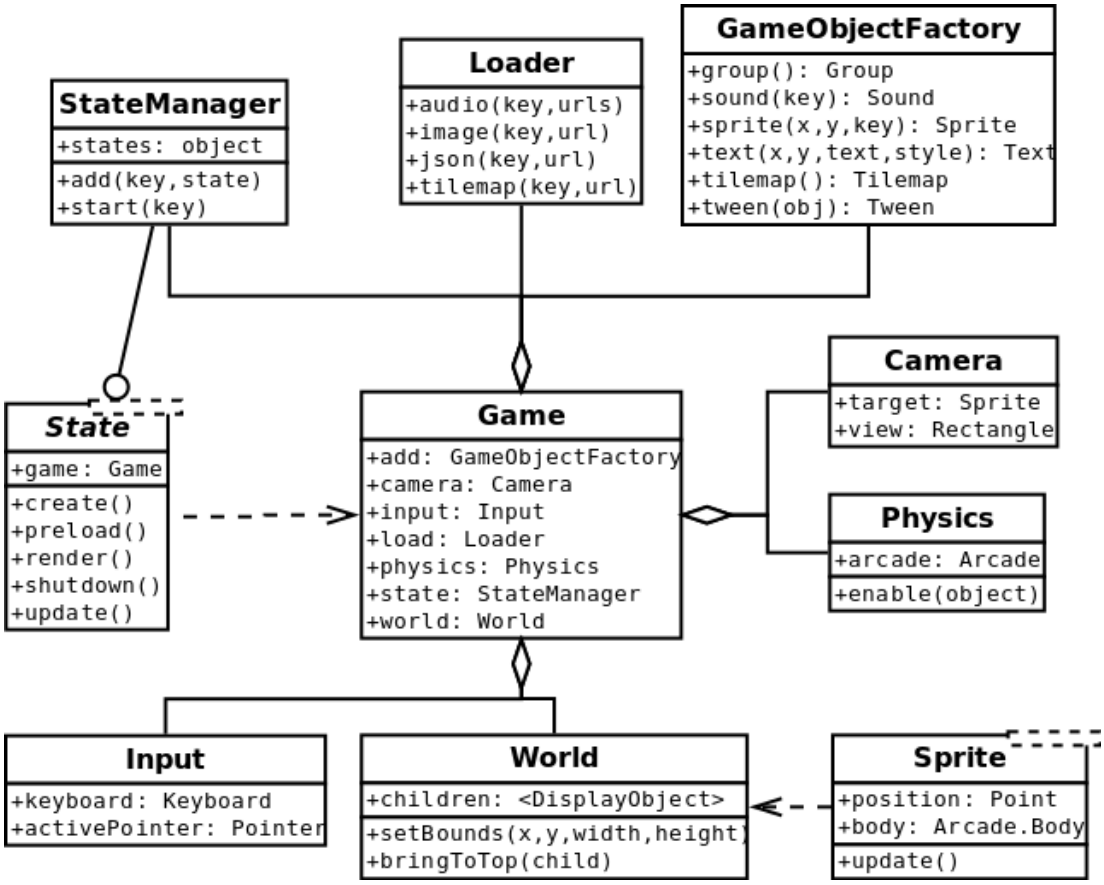
Taulukko 2. Pelimoottoreiden ominaisuuksien vertailu.

6.5 Pelimoottorin rakenne

JavaScriptissä ei ole monien muiden ohjelmointikielten tapaisia luokkia, vaan *olioita*, jotka koostuvat avain-arvo -tietopareista. Näitä olioita voidaan kutsua myös *objekteiksi*, mutta tässä tutkielmassa käytetään yksiselitteisyyden vuoksi termiä olio. Objekti-termillä viitataan jatkossa itse pelimaailmassa sijaitseviin näkyviin objekteihin. Olioiden tietoparien arvot voivat olla *metodeja*, toisia olioita, taulukoita tai yksittäisiä arvomuuttujia (esim. numeroita). Metodit ovat olion sisäisiä funktioita, jotka käsittelevät olion sisältämiä arvoja. Olioita voidaan periyttää muista olioista, jolloin niiden vakioarvot kopioidaan periytettyyn olioon. (Flanagan 2011, s. 115–140).

Valittu pelimoottori määrittää omalta osaltaan pelien teknisen rakenteen, eikä siihen ole tarvetta merkittävästi puuttua. Kuviossa 1 on esitetty Phaserin keskeinen rakenne merkittävimpien olioiden, metodien ja muuttujien osalta. Phaserissa pelit määritellään `State`-olioilla,

joiden välillä voidaan suorittaa siirtymiä siten, että tilaa vaihdettaessa edellisen tilan kaikki peliobjektit tuhotaan. Tällä tavoin kehittäjän ei tarvitse erikseen tuhota luotuja objekteja esimerkiksi pelikentän vaihtuessa, vaan pelimoottori tuhoaa ne automaattisesti. Jokainen peli on itsessään yksittäinen tila, joka luo käynnistettäessä omat peliin liittyvät objektinsa.



Kuvio 1. Pelimoottorin yksinkertaistettu rakenne luokkakaaviona.

`State`-olioille on määritelty vakiometodit, joita pelimoottori kutsuu tarpeen mukaan. Peliä ladattaessa pelimoottori kutsuu `State.preload()`-metodia, jossa määritellään `Loader`-olion kautta ladattavaksi pelin resurssitiedostot. Kun resurssitiedostot on ladattu, pelimoottori kutsuu metodia `State.create()`, jossa pelin on tarkoitus luoda pelimaailmaan kaikki pelin alkutilanteessa tarvitsemansa objektit. Tämän jälkeen peli käynnistyy, ja pelin aikana pelimoottori kutsuu sekä yksittäisten luotujen objektien että itse `State`:n `update()`-metodeja, joissa pelin suorituksen aikaista logiikkaa voidaan toteuttaa. Pelin päättyessä pelimoottori kutsuu `State.shutdown()`-metodia, jossa voidaan tarvittaessa esimerkiksi manuaalisesti tuhota pelin objekteja.

Pelin näkyvät objektit, kuten pelihahmot ovat pääsääntöisesti Phaserin `Sprite`-olioita, joille määritellään näytettävä kuva. `Sprite`-olioita voidaan luoda ja liikutella vapaasti pelin aikana. Näihin olioihin voidaan lisätä pelin toiminnallisuutta joko suoraan periyttämällä siitä uusi olio tai sisällyttämällä se pelissä määriteltyyn erilliseen olioon. Phaserissa periyttäminen on tehokas keino, sillä tällöin oliota voidaan suoraan käyttää Phaserin omissa apufunktioissa normaaleiden `Sprite`-olioiden tapaan.

Pelimoottorin keskeisin osa on `Game`-olio, johon lähes jokaisella Phaserin määrittelemällä oliolla on suora pääsy. `Game`:n kautta päästään käsiksi kaikkiin tarpeellisiin pelimoottorin sisältämiin dynaamisiin olioihin ja niiden metodeihin. Itse peli käynnistetään sivulatauksen yhteydessä luomalla uusi `Game`-olio ja määrittämällä sille ensimmäinen käynnistettävä `State`-tila. `StateManager`-olio pitää yllä tietoa kaikista pelin eri tiloista ja mahdollistaa siirtymät niiden välillä.

Pelimaailman eri objektit ja resurssit, kuten pelihahmojen spritet ja äänet, lisätään pelimaailmaan `GameObjectFactory`-olion kautta. Pelin sisältämiä objekteja voidaan vapaasti luoda ja tuhota pelin aikana. Pelimaailma kuvataan pelaajalle `Camera`-oliolla, joka määrittelee mikä alue pelimaailmasta piirretään ruudulle. Pelimaailma itsessään on `World`-olio, joka pitää sisällään tiedot kaikista piirrettävistä peliobjekteista ja mm. pelimaailman koosta.

Fysiikkamoottorin käyttöönotto on Phaserissa nopeaa. Halutut peliobjektit lisätään valitun fysiikkamoottorin listaukseen ja fysiikan mallinnus toimii objekteille tämän jälkeen pääosin automaattisesti. Perhe-pelissä käytettiin `Arcade`-fysiikkamoottoria, joka mallintaa peruskinematiikkaa ja törmäyksiä. Törmäysten tarkastelu tulee kehittäjän määritellä itse pelitilan `State.update()`-metodissa niiden objektien välille, joiden törmäyksiä halutaan mallintaa. Tämän tarkoituksena on vähentää fysiikkamoottorin tarvetta laskea kaikkien mahdollisten pelimaailman välisten objektin välisiä törmäyksiä. Törmäysten tarkastelu pelissä suoritetaan `Arcade.collide()`-metodilla.

Phaser tarjoaa joillekin olioille valmiita herätteitä (`Signal`), joita pelimoottori aktivoi, kun kyseiseen herätteeseen liitetty ehto toteutuu. Herätteitä on sisällytetty kaikkiin keskeisimpiin peliobjekteihin. Esimerkiksi `Sprite`-olioiden sisältämä `Events.onOutOfBounds`-heräte aktivoidaan, kun `Sprite` liikkuu pelimaailman ulkopuolelle. Omien herätteiden lisää-

miseksi olioihin ei ole olemassa valmiita metodeja, joten herätteiden käyttö on rajoittunut pelimoottorin valmiisiin vaihtoehtoihin.

Tween on nimitys, jota käytetään pelimoottoreissa objektien ominaisuuksien pehmeälle muuntamiselle interpoloinnin avulla. Tweenille määritetään haluttujen ominaisuuksien, kuten objektin koon tai sijainnin loppuarvo ja muunnoksen haluttu kestoaika. Pelimoottori tekee tweenien sisältämät muunnokset pelin aikana, jolloin haluttu lopputulos saavutetaan pehmeästi. Tweenien avulla voidaan luoda animaatioita tai objektien uskottavaa liikettä ilman fysiikkamoottoria. Phaserissa `Tween`-olioita voidaan lisätä käytännössä kaikkiin olioihin `GameObjectFactory.tween()`-metodilla, jolloin muunnettavien ominaisuuksien ei tarvitse olla nimenomaan näkyvien olioiden parametrejä.

6.6 Yleinen toteutus

Suurin osa pelien vaatimista toiminnallisuuksista on mahdollista toteuttaa suhteellisen suoraviivaisesti Phaserin kautta. Tekoälyä varten Phaser ei kuitenkaan tarjoa mitään valmiita ratkaisuja, joten se on luotu peleihin itse. Satunnaisarvoja peliin voidaan luoda Phaserin `RandomDataGenerator`-oliolla. Myös sekalaisia matemaattisia apufunktioita on saatavilla `Math`-oliosta (esim. erilaiset lukujen pyöritykset).

Pelien sisältämät dialogit toteutettiin HTML-elementteinä ja aseteltiin CSS:n avulla pelin canvas-elementin päälle. Tällä tavoin dialogit teksteineen saatiin skaalautumaan paremmin erikokoisille ruuduille ja teksti oli pienilläkin ruuduilla yhä luettavissa. Dialogien toteuttaminen myös pelin sisäisinä elementteinä olisi ollut mahdollista, mutta se katsottiin työmäärään nähden liian vaivalloiseksi.

6.6.1 Tiedostot

Pelin keskeiset tiedostot koostuvat HTML, CSS, JavaScript, JSON, kuva- ja äänitiedostoista. Jotta peli olisi selainten saavutettavissa, tulee pelillä olla HTML-tiedosto, jossa pelin canvas-elementti on määriteltynä. Canvas voidaan upottaa sivun muiden elementtien sekaan, jolloin se saadaan integroitua sivuston yleiseen ulkoasuun. CSS-tiedostot määrittelevät sivun elementtien ulkoasun, mutta yksinkertaisimmissa toteutuksissa niitä ei välttämättä tarvita. Ke-

hityksen aikana pelin sisältämät HTML-sivut säilytettiinkin toteutuksessa mahdollisimman minimaalisina.

Phaser osaa lukea esimerkiksi TexturePackerin ja Tiledin tuottamaa JSON-dataa, jonka määrittely on löydettävissä pelimoottorin dokumentaatiosta. Tällä tavoin ei ole tarpeellista lisätä tiedostojen nimiä tai muita yksityiskohtia suoraan lähdekoodiin, vaan tietoja voidaan hallita keskitetysti JSON-tiedostojen kautta. JSON-tiedostossa jokaiselle resurssille määritellään yksilöivä nimi (nk. *key*), jolla resurssiin viitataan lähdekoodissa. Kaikki tutkielmassa toteutettujen pelien ääni- ja kuvatiedostot määriteltiin tällä tavoin.

Pelin kuvatiedostot koostuvat PNG-tiedostoista, joiden avulla voidaan määritellä kuviin tarvittaessa läpinäkyvyyttä. Myös JPEG-kuvien käyttö olisi osittain mahdollista, mutta PNG-kuvat soveltuivat paremmin pelien sarjakuvamaiseen ulkonäköön ja rajalliseen värien määrään. TexturePacker-sovellusta käytettiin yhdistämään minipelikohtaiset kuvatiedostot yksittäisiksi tiedostoiksi. Äänitiedostot peleihin lisättiin sekä MP3- että OGG-tiedostoina yhteensopivuuden takaamiseksi.

JavaScript-tiedostot on kehityksen aikana eroteltu pienempiin osioihin helpommin hallittaviksi. Grunt-apuohjelman avulla ne lopulta yhdistetään ja *minimoidaan* (engl. *minification*) julkaisua varten. Minimisoinnilla tarkoitetaan lähdekoodin tiivistämistä poistamalla siitä kaikki tarpeettomat kirjaimet ja lyhentämällä mahdollisesti muuttujien nimiä (Souders 2008). Tällöin lähdekoodi vie vähemmän tilaa ja selain pystyy prosessoimaan sen nopeammin. Myös sivulataukset nopeutuvat, kun tiedostot ovat pienempiä ja niitä on vähemmän. Minimisoinnin voi tehdä myös monille muille tiedostotyypeille, kuten CSS-tiedostoille.

Grunt-apuohjelmalla tehdään minimisoinnin ohella myös muita automatisoituja tehtäviä, kuten kuvatiedostojen optimointi, äänitiedostojen muuntaminen MP3-muodosta OGG-muotoon, pelin lähettäminen palvelimelle sekä lähdekoodia analysoivan *JSHint*-sovelluksen raportin luonti. Äänitiedostojen automaattinen luonti poistaa näin tarpeen hallita ja säilyttää kehityksen aikana OGG-tiedostoja MP3-tiedostojen ohella. JSHint on staattinen lähdekoodin analysointisovellus, joka auttaa löytämään selkeitä virheitä lähdekoodista (Gong ym. 2015). Sitä käytetään pelitoteutuksissa parantamaan lähdekoodin tasoa sekä välttämään joitain dynaamisen kielen aiheuttamia ongelmia.

6.6.2 Resoluutio

Koska kyseessä on bittikarttagrafiikkaan perustuva peli, on pelille määriteltävä ns. *natiiviresoluutio*, jolla pelin grafiikka piirretään. Tavanomaisten videopelien tavoin suurempi resoluutio vaatii käytetyltä laitteistolta parempaa graafista suorituskykyä, joten valinta on tehtävä erityisesti kohdennettua minimilaitteistoa silmällä pitäen. Resoluution vaihtaminen lennosta pelin aikana on käytössä olevalla pelimoottorilla mahdollista, mutta vaikeuttaa graafisten elementtien suunnittelua, sillä bittikartat tulee tällöin skaalata ja ankkuroida erikseen ruudun resoluution mukaan. Varsinainen peli-ikkuna itsessään on mahdollista skaalata esimerkiksi kokoruudun tilaan, mutta siinä tapauksessa sen resoluutio ei muutu.

Toteutettuihin peleihin haluttiin valita yksi kiinteä resoluutio, joka ei muuttuisi missään vaiheessa. Natiiviresoluution valitsemiseksi pelin suorituskykyä mitattiin eri laitekokoonpanoilla. Testaus suoritettiin Phaser-pelimoottorilla luomalla sen avulla yksinkertainen graafinen esitys, jossa oli muutamia liikkuvia kuvia. Toteutuksen ruudunpäivitysnopeutta seurattiin pelimoottorin `Time.fps`-muuttujan arvon avulla. Tarkoituksena oli kartoittaa, onko mahdollista käyttää suurempaa resoluutiota, kuin mitä teknisissä vaatimuksissa luvussa 6.3 on määritelty. Testattujen resoluutioiden kuvasuhteet valittiin osin toisistaan poikkeaviksi, jotta samalla voitaisiin arvioida kuvasuhteiden eroja eri ruuduilla.

Testauksen tulokset taulukossa 3 osoittavat, että mobiililaitteiden suorituskyky putoaa resoluutiota nostettaessa hyvin nopeasti. Pöytätietokoneet taas kykenevät *Full HD* -tasoiseen tarkkuuteen ruudunpäivityksen nopeuden kärsimättä. Testauksessa myös kokeiltiin skaalata eri resoluutioita kokoruudun tilaan, mutta sillä ei ollut mitään vaikutusta ruudunpäivitysnopeuteen. Li ja Bao (2014) kirjoittavat, että Internet-selainten rajoittama korkein ruudunpäivitysnopeus on oletuksena 60 ruutua sekunnissa. Mittausten mukaan mm. Internet Explorer 11 on tästä poikkeus, sillä sen ruudunpäivitysnopeuden yläraja vaikuttaa olevan pikemminkin 67 ruutua sekunnissa.

Testien perusteella on kannattavinta valita peliin pienin mahdollinen resoluutio, joka soveltuu pelin graafiseen ulkoasuun. Tutkielman pelien lopullisen resoluution valintaan vaikutti myös kuvasuhde, sillä pelisuunnitelman mukaan peleihin haluttiin toteuttaa myöhemmin myös erillinen sosiaalista sisältöä käsittävä palkki kuvaruudun reunaan. Keskustelupalkin li-

	960x720 (4:3)	**1280x720** (16:9)	**1366x768** (16:9)	**1920x1200** (16:10)	**WebGL**
Acer Iconia W500, Win 7					
Chrome 42	57	52	47	41	kyllä
Internet Explorer 10	60	60	60	60	ei
Ainol Novo Hero, Android 4.3.1					
Firefox 37	40	36	34	26	kyllä
Vakioselain	38	31	30	29	kyllä
iPad 1st gen., iOS 5.1.1					
Safari 5.1	67	64	58	25	ei
iPad 4th gen., iOS 8.3					
Safari 8.0	60	60	59	33	kyllä
LG L4, Android 4.1.2					
Chrome 43	20	18	15	5	ei
Vakioselain	64	57	55	36	ei
LG L7, Android 4.4.4					
Firefox 36	18	16	15	*	kyllä
Vakioselain	*	*	*	*	—
iMac 27'', OS X 10.10					
Safari 8.0	40	41	39	37	kyllä
PC, Xubuntu 15.04					
Chrome 43	60	60	60	60	kyllä
Firefox 37	60	60	60	45	kyllä
PC, Windows 7					
Chrome 42	60	60	60	60	kyllä
Firefox 31	23	18	15	7	kyllä
Internet Explorer 11	67	67	65	64	kyllä

*ei toiminut

Taulukko 3. Ruudunpäivitysnopeus (ruutua/sekunti) eri laitteilla ja resoluutioilla.

säämistä kokeiltiin erillisenä HTML-elementtinä pelin canvas-elementin rinnalle, ja käytännön testien perusteella laajakuvasuhteet 16:9 ja 16:10 olisivat olleet liian leveitä erillisen palkin lisäämiseksi. Näin ollen kuvasuhteeksi valittiin 4:3 ja pelien varsinaiseksi resoluutioksi 960x720.

6.6.3 Käyttäjäsyöte

Pelin syötteet haetaan `Input`-olion kautta, joka sisältää tarvittavat metodit näppäimistön, hiiren ja muiden syötelaitteiden lukemiseksi. Syötteet käsitellään pelin `State.update()`-metodissa, mutta niihin voidaan sitoa myös herätteitä. Molemmissa tutkielmaan toteutetuissa peleissä syötteet käsiteltiin suoraan ilman herätteitä. Pelimoottori ei erottele ilman erityistä tarvetta kosketusnäyttöjen ja hiirten kursoreita toisistaan, ts. täysin sama kursoreihin liittyvä syötelogiikka toimii ilman erityisiä muutoksia myös suoraan tableteilla ja pöytätietokoneilla.

Käyttäjäsyötteen lukeminen pelimoottorin luontaisella päivitysnopeudella voi olla tarpeetonta, sillä 25–30 kertaa sekunnissa on usein täysin riittävä syötteen päivitysnopeus. Gregory (2014, s. 340–341) toteaa, ettei pelin kaikkia osia, kuten tekoälyä tarvitse välttämättä päivittää kuin kerran tai kaksi sekunnissa. Phaser pyrkii joka tapauksessa renderöimään pelin grafiikan ja päivittämään kaiken muun toiminnallisuuden 60 kertaa sekunnissa.

Pelin logiikalle tarkoitetun `State.update()`-metodin kutsunopeuteen on pelimoottorissa mahdollista vaikuttaa `Time.suggestedFps`-muuttujan avulla. Tällöin pelin logiikan päivitystä voidaan hidastaa esim. 30 ruutuun sekunnissa. Pelaaja ei luultavasti havaitse pelissä mitään eroa vaikka pelin logiikan ja syötteiden päivitysnopeus olisikin hitaampaa, mutta pelin suorituskykyä se voi joka tapauksessa parantaa. Tästä syystä tutkielmassa toteutetuissa peleissä `update()`-metodin päivitystiheys laskettiin 25 ruutuun sekunnissa.

Phaser tukee monieleohjausta, mikäli laitteisto itsessään sen sallii. Tutkielman pelien toteutus ei kuitenkaan vaatinut useamman kursorin hallintaa, joten aktiivisten kursorien maksimimääräksi asetettiin vain yksi. Tämä vaikutti suoraan pelattavuuteen mobiililaitteilla, sillä näin esimerkiksi ruudun laidan koskettaminen vahingossa pelaamisen aikana ei rekisteröitynyt peliin. Monieleohjausta käytettäessä olisi lisäksi otettava huomioon pelin käyttäminen niillä laitteilla, joissa ei ole lainkaan kosketusnäyttöä.

6.6.4 Efektit

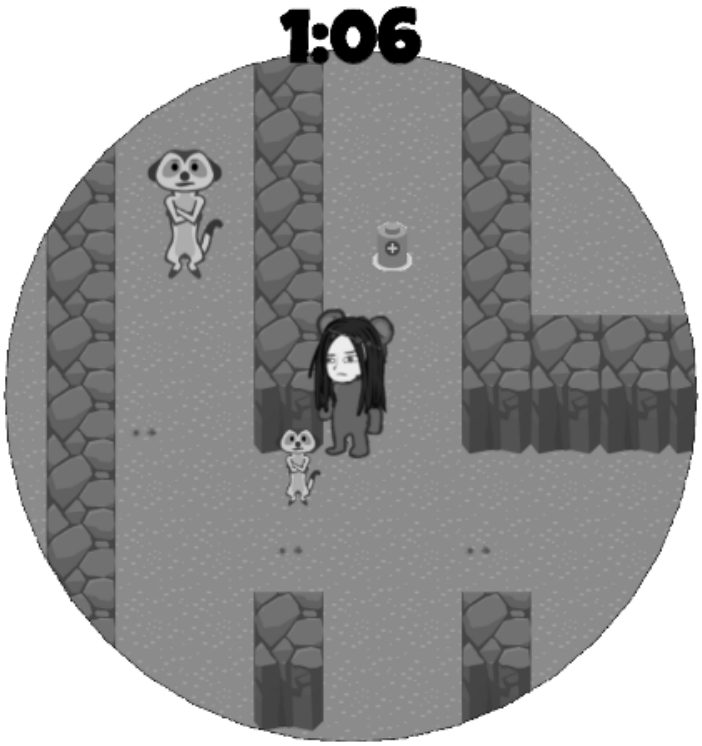
Phaserissa partikkeliefektejä voidaan luoda `Arcade.Emitter`-oliolla, jolla sprite-pohjaisia yksittäisiä partikkeliobjekteja on mahdollista hallita. `Emitter` sisältää tiedot partikkeleiden fysikaalisista ominaisuuksista ja luo, sekä tuhoaa yksittäisiä partikkeleita halutulla nopeudella. Tutkielman peleissä partikkeliefekteinä käytettiin yhdestä pisteestä eri suuntiin ammuttuja pieniä tähtiä, jotka katoavat nopeasti. Partikkeliefektien lisäksi peleissä käytetään mobiililaitteiden tukemaa värinää (engl. *vibrate*) tuomaan lisäpalautetta pelaajalle. Phaserin dokumentaatio ei toistaiseksi kuvaa värinäpalautteiden käyttöä, mutta määrittelee silti `navigator.vibrate()`-metodin, jolla laitteita on mahdollista saada värisemään määritellyn ajan.

6.7 Perhe-pelin toteutus

Graafisesti Perhe-pelin pelimaailma on kuvattu yläviistosta (kuvio 2). Syvyysvaikutelma tähän puhtaasti kaksiulotteiseen peliin on luotu tekstuurien avulla. Esimerkiksi pelin objektit kuvataan vain toiselta sivultaan, mutta labyrintin seinistä myös yläosa on pelaajalle näkyvissä. Phaserille on olemassa lisäosia, jotka helpottavat esimerkiksi *isometrisen* kuvakulman luomista, mutta Perhe-pelissä toteutus tehtiin puhtaasti Phaserin omilla toiminnallisuuksilla.

Pelin rajattu näkyvyys on toteuttu Phaserin peite-objektin (engl. *mask*) avulla, joka mahdollistaa muiden pelin objektien näkyvyyden suodattamisen. `mask` on yksinkertainen graafinen olio, joka määrittelee sen, mikä alue pelistä on näkyvää ja mikä peitetään pois. Perhe-pelissä pelaajan näkemää aluetta rajattiin normaalin koko peliruudun sijaan ympyrän muotoisella `PIXI.DisplayObject`-oliolla, joka asetettiin erikseen kaikkien peliobjektien `mask`-muuttujan arvoksi.

Peite-objektin halkaisijaa päivitetään pelissä jäljellä olevan ajan perusteella ja peitteen sijainti on sidottu pelaajan hahmon sijaintiin. Peite on mahdollista määritellä myös `World`-olioon, jolloin se asetetaan automaattisesti kaikille pelimaailman objekteille. Tämä ratkaisu olisi kuitenkin asettanut väistämättä `mask`-olion myös ruudulla näkyvälle kellonajalle, minkä ei haluttu koskaan peittyvän.



Kuvio 2. Perhe-pelin ulkonäkö.

Pelissä näkyvä kellonaika on `Text`-olio, jonka sisältämää tekstiä päivitetään vastaamaan jäljellä olevaa aikaa. Tekstin väri muutetaan punaiseksi, kun aikaa on jäljellä enää 30 sekuntia. Tekstin fontti voidaan itse määritellä ja pelissä käytetty fontti ladataankin pelin alkaessa dynaamisesti *Google Fonts* -palvelusta. Teksti pidetään manuaalisesti ruudun päällimmäisenä objektina `Text.bringToTop()`-metodin avulla.

6.7.1 Labyrintti

Pelin labyrintti luodaan luvussa 4.8.5 esiteltynä tilemap-tiilikarttana. Phaserissa tiilikartta luodaan `Tilemap`-oliona, joka sisältää tiedot kartan yksittäisistä tiilistä. Karttaan liitetään lisäksi kuvatiedosto, joka sisältää kuvat jokaisesta kartalla käytetystä tiilestä. Jotta tiilikartta voidaan varsinaisesti piirtää, luodaan kartasta lisäksi `TilemapLayer`-olio, joka on kartasta generoitu kuva. Koska pelissä labyrintti luodaan joka pelikertaa varten erikseen, ei sitä ladata tiedostosta, vaan käytetty `Tilemap`-olio täytetään algoritmin avulla.

Labyrinttien luomiseksi on olemassa suuri määrä erilaisia algoritmejä, jotka eroavat toteutus-

tavoiltaan ja generoivat juuri tietyn tyylisiä labyrintin käytäviä. On esimerkiksi mahdollista luoda labyrinttejä, jotka sisältävät suuren määrän pitkiä umpikujia. Tutkielman peliin riittävä algoritmi oli kuitenkin sellainen, joka kykenee luomaan suhteellisen yksinkertaisia käytäviä eri pelin vaikeustasoja varten. Pelin vaikeustaso vaikuttaakin vain labyrintin kokoon.

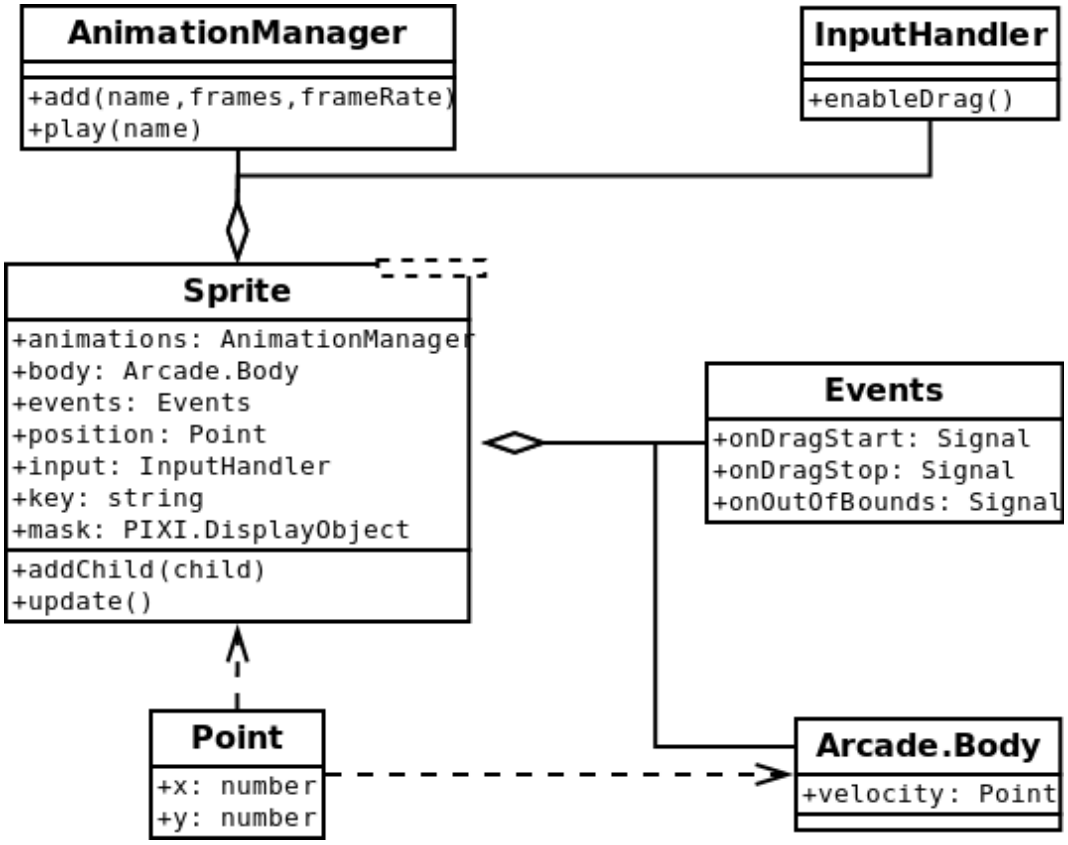
Labyrintin luomiseksi valittiin *recursive backtracking* -algoritmi, joka käy satunnaisesti jokaisen labyrintin käsittelemättömän ruudun läpi, palaten umpikujaan osuessaan takaisin edelliseen ruutuun (Singh ja Pandey 2014). Algoritmille voidaan määritellä satunnaisarvo, jonka perusteella vierekkäisiä käytäviä yhdistetään, jotta labyrintistä ei tulisi vain yksi pitkä käytävä. Algoritmi soveltuu kaikenkokoisten labyrinttien luomiseen ja se on yksinkertainen toteuttaa monilla eri ohjelmointikielillä. Labyrintin ruutujen tekstuurit pelin karttaan valitaan satunnaisarvoon perustuen, jotta vältytään liialta tekstuurien toistuvuudelta.

Fysiikan lisääminen tiilikarttaan onnistuu asettamalla manuaalisesti jokainen yksittäinen tiili kartalta seinäksi tai listaamalla vain ne tiilikuvien numerot, jotka edustavat seiniä. Syöttämällä numerot `Tilemap.setCollision()`-metodiin, Phaser merkitsee yksittäiset tiilet seiniksi, joihin on mahdollista törmätä. Varsinainen törmäystarkastelu tiilikartan seiniin määritellään jokaiselle pelin objektille erikseen.

6.7.2 Pelihahmot

Pelihahmojen oliot on periytetty `Sprite`-oliosta (kuvio 3) ja niille on määritelty lisäksi animaatioon ja tekoälyyn liittyviä toiminnallisuuksia. `State.update()`-metodin avulla on mahdollista toteuttaa tekoälyyn liittyvää laskentaa. Vaihtoehtoisesti tekoälyn apuna voidaan myös käyttää herätteitä, mutta niiden valikoima on rajoittunut suhteellisen yksinkertaisiin signaaleihin. Molempien edellä mainittujen menetelmien käyttäminen yhtä aikaa voi myös tehdä toteutuksesta sekavan. Perhe-pelissä herätteitä ei käytetty tekoälyn apuna.

Pelin mangustien sijainnit arvotaan pelin alkaessa satunnaisesti. Pelikenttä jaetaan ensin neljään osaan, joiden sisälle arvotaan pelaajalle sekä kolmelle mangustille omat paikkansa. Tällöin hahmojen aloituspaikat eivät koskaan voi päätyä liian lähelle toisiaan. Mangustihahmo alkaa seurata pelaajaa, kun tämä törmää siihen. Tällöin näytetään lisäksi luvussa 6.6.4 kuvattu partikkeliefekti mangustin ympärillä.



Kuvio 3. Sprite-olion yksinkertaistettu rakenne luokkakaaviona.

Koska pelimoottori on puhtaasti kaksiulotteinen, lomittuvat pelin spritet toistensa päälle siinä järjestyksessä, missä ne on alun perin luotu. Pelimaailman on kuitenkin tarkoitus näyttää osittain yläviistosta kuvatulta, jolloin pelin objekteilla on kolmatta ulottuvuutta edustava syvyysarvo. Tässä tapauksessa objektien syvyysarvo on täysin riippuvainen objektin y-koordinaatista. Käytännössä tämä tarkoittaa sitä, että objektit piirretään y-koordinaatin arvon perusteella pienemmästä suurempaan. Tällöin ruudun alareunan objektit piirretään aina ylempien päälle, sillä syvyysvaikutelman mukaan niiden on tarkoitus olla lähempänä kameraa.

Edellä kuvattua syvyysvaikutelmaa on mahdollista tehostaa skaalaamalla ruudun yläreunan objekteja pienemmäksi y-koordinaatin perusteella. Toteutettuun peliin tämä ei kuitenkaan ollut tarpeellista. Phaser mahdollistaa pelihahmojen helpon järjestämisen halutun ehdon mukaisesti `sort()`-metodilla, kun ne on lisätty samaan `Group`-olioon. `Group` on useiden yksittäisten peliobjektien (esim. `Sprite`) tallentamiseen ja järjestelyyn tarkoittu olio.

Pelaaja on toteutettu muiden pelihahmojen tapaan `Sprite`-oliona, joka on lisätty fysiikkamoottorin listalle. Kamera seuraa automaattisesti pelaajan liikkeitä pelimaailmassa, mikä on määritelty suoraan `Camera.focusOn()`-metodin avulla. Tämä metodi sitoo kameran sijainnin haluttuun pelin objektiin. `State.update()`-metodissa luetaan pelaajan antama syöte ja hahmoa liikutetaan sen mukaisesti.

Aktiivisen kursorin painalluksen kesto luetaan `Input.activePointer`-muuttujasta. Alle 500 millisekuntia kestävässä painalluksessa pelaajaa liikutetaan mahdollisen aktiivisen kursorin suuntaan ruudulla. Tätä kauemmin kestänyt painallus katsotaan pyyhkäisyksi ja pelaajaan nopeus asetetaan kursorin liikkeen suuntaiseksi. Mikäli kursori ei ole aktiivinen, näppäimistön syöte luetaan `Keyboard`-oliosta `isDown()`-metodilla. Pelaajan nopeus asetetaan `Sprite.body.velocity`-olioon, jota fysiikkamoottori käyttää uuden sijainnin laskentaan.

Mikäli pelaajalta ei ole tullut uutta syötettä edellisen logiikan päivityksen jälkeen, hidastuu hahmon nopeus automaattisesti `Sprite.body.drag`-olioon asetetun kitkaluvun mukaisesti. Pelaajan jättämät jalanjäljet ovat `Group`-olioon tallennettuja `Sprite`-olioita, joita luodaan pelaajan liikuttua riittävän kauas edellisistä jättämistään jalanjäljistä. Vanhoja jalanjälkiä poistetaan hiljalleen, jotta niitä ei kertyisi ennen pitkää liian paljon. Kaikkien pelihahmojen törmäykset luolaston seiniin ja toisiinsa käydään läpi, jotta hahmot liikkuisivat pelimaailmassa mahdollisimman realistisesti.

6.7.3 Tekoäly

Pelin tekoäly on verrattain yksinkertainen. Pelaajaa seuraavat mangustit tallentavat tasaisin väliajoin pelaajan sijainnin ja seuraavat näitä pisteitä järjestyksessä. Mikäli mangusti on liian lähellä pelaajaa, se pysähtyy ja poistaa tallentamansa sijaintipisteet. Vastaavasti hiiret liikkuvat pelaajasta poispäin kun ne havaitsevat pelaajan. Hiirten pakonopeus säilytetään vakiona kulkusuunnan mukaisena, jolloin seiniin törmätessään ne alkavat liikkua seinän suuntaisesti. Tekoälyyn olisi ollut helppo lisätä Phaser-lisäosan avulla jokin polunhakualgoritmi, kuten A*, mutta se katsottiin pelin suhteellisen yksinkertaisuuden vuoksi tarpeettomaksi.

6.8 Terveys-pelin toteutus

Perhe-peliin verrattuna Terveys-pelin pelimekaniikka perustuu pikemminkin animaatioihin ja tekoälyyn, kuin pelaajan jatkuvaan kontrolliin. Pelissä ei myöskään ole fysiikan mallinnusta tai kameran liikettä. Huomionarvoista on, että Terveys-peli toteutettiin muutama kuukausi Perhe-pelin jälkeen, jolloin kehittäjillä oli jo parempi tietämys pelimoottorista ja sen ominaisuuksista.

Pelimaailma Terveys-pelissä (kuvio 4) on rajattu pelkästään kameran näyttämän alueen kokoiseksi `World.setBounds()`-metodin avulla, ja pelimaailman kokoinen taustakuva on asetettu maailman keskelle. Pelissä soi lisäksi pelaamisen aikana taustamusiikki, joka on luotu `Sound`-oliona. Käyttäjän syötteet saadaan kontrolliesineiden kautta, eikä niitä lueta manuaalisesti kuten Perhe-pelissä. Terveys-pelissä käytettyjen `Sprite`-olioiden rakenne on yhtäläinen luvussa 6.7.2 esitellyn mallin kanssa.

Kuvio 4. Terveys-pelin ulkonäkö.

6.8.1 Kontrolliesineet

Kontrolliesineet ovat yksinkertaisia `Sprite`-olioita, joille on määritelty syötteen mukainen raahaus `InputHandler.enableDrag`-muuttujan kautta. Tällöin pelaaja voi esinet-

tä koskettaessaan raahata sitä ruudulla. Raahauksen alkaminen ja loppuminen kaapataan herätteiden `Events.onDragStart` ja `Events.onDragStop` avulla. Esineen raahauksen loppuessa tarkastetaan `Sprite.overlap()`-metodilla, onko se lemmikin päällä ja annetaan tarvittaessa kyseisen esineen tiedot lemmikin tekoälylle.

Esineille on määritelty satunnaiset tween-efektit, joilla esineet paikallaan ollessaan hieman pyörivät, liikkuvat tai tärisevät, jotta ne näyttäisivät enemmän dynaamisilta peli-elementeiltä. Esineet myös siirretään raahauksen jälkeen automaattisesti takaisin alkuperäisille paikoilleen `Tween`-olioiden avulla. Jos lemmikki hyväksyy annetun esineen, näytetään esineen kohdalla vahvistukseksi yksinkertainen partikkeliefekti ja mobiililaitteilla annetaan lisäksi lyhyt värinäpalaute.

6.8.2 Virtuaalilemmikki

Pelin lemmikki on jaettu graafisesti eri osiin, joista jokainen on oma `Sprite`-olionsa. Joitain näistä `Sprite`-olioista on määritelty tarvittaessa toistensa aliobjekteiksi, jotta niiden sijainti on ankkuroituna pelimaailman sijaan toisiin objekteihin. Esimerkiksi lemmikin silmät ovat lemmikin pään aliobjekteja ja liikkuvat tällöin automaattisesti aina pään mukana. Lemmikin pitämät äänet ovat musiikin tavoin yksittäisiä `Sound`-olioita.

Lemmikin animaatiot on toteutettu sprite-animaatioiden ja tweenien yhdistelminä. Yleisanimaatiot lemmikille aloitetaan jokaista animaatiota varten erikseen määritellyllä metodikutsulla. Metodissa luodaan jokaiselle halutulle lemmikin osalle oma `Tween`, johon ketjutetaan `Tween.to()`-metodilla halutut liikkeet. Samalla käynnistetään myös mahdolliset sprite-animaatiot. Animaatioiden ajastukset on määritelty käsin ja niiden muuttaminen vaatii aina testaamista.

Kaikkien raajojen animaatioiden toteuttaminen puhtaasti sprite-pohjaisena olisi vaatinut valtavan määrän yksittäisiä kuvia ja rajoittanut pelin toteutusta. Esimerkiksi lemmikin nokka on toteutettu sprite-animaationa, joka sisältää erilliset kuvat nokasta jokaisessa eri asennossa. Sprite-animaatiot määritellään `AnimationManager`-olion kautta, jossa listataan kuvat siinä järjestyksessä, jossa ne halutaan animaatiossa näyttää. Samaa menetelmää ei ole kuitenkaan käytetty hahmon raajoihin, vaan näitä yksittäisiä kuvia liikutetaan tweenien avulla.

6.8.3 Tekoäly

Pelin lemmikin tekoäly on toteutettu tilakoneena[3], jossa tilojen välisiä siirtymiä tehdään esineiden nimien perusteella. Lemmikin alkutila valitaan satunnaisesti muutamasta perusvaihtoehdosta, jotka mahdollistavat riittävän pitkän peliajan. Tilakoneet ovat hyvin tyypillisiä tietojenkäsittelytieteen käsitteitä ja niitä on helppo mallintaa moderneilla ohjelmointikielillä (esim. Orponen 1994, s. 7–30).

Perinteisestä tilakoneesta poiketen jokaiseen tilaan on määritelty lista animaatioista, joita näytetään kyseisen tilan aikana, jotta pelaaja tietäisi mitä lemmikki milloinkin haluaa. Lisäksi esineitä edustaviin tilasiirtymiin on määritelty kaksi animaatiota. Toinen mahdollisista animaatioista näytetään, kun pelaaja tarttuu kyseiseen esineeseen ja toinen kun esine lopulta annetaan lemmikille. Tilakoneessa on mahdollista tehdä myös automaattinen siirtymä toiseen tilaan määritellyn ajan kuluessa, jolloin tiloja voidaan käyttää välianimaatioiden esittämiseen.

3. tarkalta määritelmältään *epädeterministinen äärellinen automaatti*

7 Arviointi

Tässä luvussa käydään läpi tutkielmassa toteutettujen pelien kehityksen, testauksen sekä laadullisten ominaisuuksien arviointia. ISO 9126-1 on yleisesti käytetty sovellusten laatustandardi ja se määrittelee useita laadullisia ominaisuuksia sovellusten arviointiin (ISO/IEC 2000). Tässä tutkielmassa ei ole tarpeen arvioida niitä kaikkia ja niistä poimitaankin vain tutkimuksen kannalta oleellisimmat. Pelien laadullisessa arvioinnissa keskitytään suorituskykyyn ja yhteensopivuuteen. Luvussa arvioidaan myös pelien kehityksessä ja testauksessa kohdattuja ongelmia. Lopuksi tarkastellaan alkuperäisten pelivaatimusten täyttymistä, pohditaan tulosten yleistymistä muihin HTML5-pelitoteutuksiin ja esitetään konstruktion pohjalta syntyneitä kysymyksiä mahdollisia jatkotutkimuksia varten.

Suorituskyvyn osalta arvioitiin sen vaikutusta yleiseen pelikokemukseen ja vertailtiin tarkemmin laitteiden ruudunpäivitysnopeuksia. Claypool ja Claypool (2009) ovat tutkimuksensa perusteella todenneet, että pelien pelattavuus putoaa itsessään merkittävästi, mikäli ruudunpäivityksen nopeus on alle 15 ruutua sekunnissa. Pelin riittävä suorituskyky ei kuitenkaan kosketa vain esteettisiä piirteitä, sillä he lisäksi huomauttavat, että pelaajien yleinen suoriutuminen pelissä paranee, mikäli ruudunpäivitys on vähintään 30 ruutua sekunnissa. Tämän tutkielman peleissä 15 ruutua sekunnissa katsottiin ehdottomaksi alarajaksi ja vähintään 30 ruutua sekunnissa pelien ideaaliseksi tavoitteeksi.

Yhteensopivuutta arvioitiin vertaamalla pelejä eri laitteilla sekä selaimilla ja havainnoimalla niiden välisiä eroja. Ideaalisena tavoitteena yhteensopivuuden osalta on saada pelit ulkoasultaan ja keskeisiltä toiminnoiltaan täsmälleen samanlaisiksi kaikilla eri kokoonpanoilla. Jotta pelin voidaan katsoa olevan yhteensopiva, tulee sen vastata toisella järjestelmällä toimivaa toteutusta kaikin muun tavoin paitsi syötelaitteiden osalta.

Yhteensopivuuden ohella arvioitiin pelitoteutusten luotettavuutta, jotka kategorisoitiin yhteensopivuusongelmiksi, mikäli ne tapahtuivat pelitoteutuksesta riippumattomista syistä. Esimerkiksi pelin kaatuminen tai virheilmoitusten näkyminen minipelien aikana juuri tietyillä laitteilla katsottiin yhteensopivuuden rajoitteiksi. Testauksessa vertailtiin pelien pelaamista ja yleistä pelikokemusta eri laitteistoilla. Testauksen näkökulmasta katsottuna laitejärjes-

telmän ei tulisi haitata pelaamista tai tehdä pelikokemuksesta muista laitteista poikkeavaa. Testauksessa esille tulleet käytettävyysongelmat huomioitiin ja kirjattiin erikseen osaksi laitejärjestelmien välisiä yhteensopivuusongelmia.

7.1 Testaus

Toteutettuja pelejä testattiin sekä lasten että itse peliprojektin tutkimusryhmän toimesta. Lapset ovat pelien suunniteltu kohderyhmä, ja Hass (2008, s. 253–254) huomauttaa, että juuri kohderyhmän suorittama testaus on sovellukselle tärkein käytettävyyden arviointikeino. Tutkimusryhmä teki testipalautteen kyselynä ja lapset seurannan sekä kyselypalautteen avulla. Lasten antama palaute peleistä oli keskittynyt pelin sisältöön, mutta palaute kohdistui myös pelien teknisiin ongelmiin sekä suorituskykyyn. Tutkimusryhmä keskittyi testaamisessaan pelien yhteensopivuuteen ja mahdollisiin pelaamisen eroavaisuuksiin eri laitejärjestelmillä.

7.1.1 Koulutestaus 1

Keskeneräistä Perhe-peliä testattiin osana muita peliprojektin minipelejä alakoululla toteutetussa tilaisuudessa 27.4.2015, jossa viidesluokkalaiset (yksi luokka) kokeilivat pelejä 10 minuuttia kerrallaan. Lasten pelaamista havainnoitiin ja he saivat kirjoittaa peleistä mielipiteensä post-it -lapuille otsikoiden ”mikä toimii” ja ”mikä ei toimi” -alle. Testaus suoritettiin iPad-tableteilla ja Safari-selaimilla. Lapsille kerrottiin yleinen kuvaus Sisunmaan Sankarit -pääpelistä, mutta minipelien ohjeistusta ei annettu erikseen ennen pelaamista.

Testatusta Perhe-pelin versiosta puuttui tuolloin aloitusdialogi, mikä aiheutti sekaannusta pelin lopullisesta tavoitteesta ja tarkoituksesta. Lisäksi pelissä jäljellä oleva aika kulki nopeammin, kuin mitä todellisuudessa olisi pitänyt. Syynä tähän olivat pelimoottorin ajastimen ongelmat laitteilla, joiden ruudunpäivitysnopeus oli merkittävästi hitaampi kuin 60 ruutua sekunnissa. Laitteiden ruudunpäivitysnopeus oli huonoimmillaan vain noin 5 ruutua sekunnissa, mikä johtui tiilikartan puutteellisesta suorituskyvystä (kuvattu tarkemmin luvussa 7.2.2). Edellä olevista ongelmista johtuen pelin syvällisempi testaus jäi lopulta vähäiseksi.

Yleinen palaute kaikista minipeleistä kosketti pelien hidasta latautumista. Testaukseen varauduttaessa ei otettu huomioon, että yksittäiset minipelit lataavat kaikkien minipelien resurssit

jokaista pelikertaa varten, mikä aiheutti pitkiä odotusaikoja pelien välillä. Latausongelma korjattiin seuraavaa testauskertaa varten minipelikohtaiseksi.

7.1.2 Koulutestaus 2

Toinen koululaisille järjestetty testauskerta suoritettiin 8.5.2015 ensimmäisen testauskerran tavoin viidesluokkalaisten toimesta (toinen koululuokka). Tällä kertaa myös minipeleistä kerrottiin lapsille ennen testausta hieman tarkemmin. Tässä testauksessa lapset saivat vapaasti kokeilla 20–25 minuutin ajan muiden minipelien ohella molempia tutkielmassa kehitettyjä pelejä. Terveys-peli oli tuolloin vielä keskeneräinen sisältäen vain muutaman animaation ja tilasiirtymän. Perhe-pelin ajastin- ja ruudunpäivitysongelmia oli korjattu edellisen testauskerran jälkeen.

Lapset pitivät kaikkia testaamiaan minipelejä hauskoina pelata. Peleihin kaivattiin eri vaikeustasoja, joita ei vielä testausvaiheessa oltu implementoitu. Pelien yleinen ohjeistus koettiin edelleen puutteelliseksi, sillä pelien varsinaista mekaniikkaa ja kontrolleja ei minipeleissä selitetty erikseen. Tällä kertaa minipelit latautuivat edellistä testitilaisuutta nopeammin, eikä siitä enää huomautettu.

Perhe-peliä pidettiin hieman liian vaikeana ja sen tavoitteet olivat edelleen vaikeita ymmärtää. Lapset eivät mm. olleet varmoja siitä, kuinka monta mangustia luolastosta tulee etsiä. Lapset myös kokivat pelin valokeilan hyppäämisen ruudulla eri kohtaan häiritseväksi. Valokeilan hyppäys johtui luvussa 7.2.2 kuvattavasta suorituskyvyn ongelmasta, jota yritettiin ratkaista liikuttamalla pelimaailmaa vain ruutu kerrallaan. Lapset tykkäsivät pelin mangustihahmoista ja yleinen pelimekaniikka todettiin pelaamista havainnoitaessa toimivaksi. Testauspalautteessa saatiin peliin kehitysideana ”valokatkaisin”-bonus, jossa pelaaja luolastosta valokatkaisimen löydettyään näkisi kerralla koko ympäröivän labyrintin.

Terveys-pelin lemmikkihahmoa pidettiin lasten mielestä miellyttävänä, joskin pelin toiminnallisuus koettiin vielä sekavaksi. Lemmikillä ei ollut testauksessa vielä ääniä, mikä aiheutti osaltaan hämmennystä lemmikin esittämistä tarpeista. Lemmikki myös vaipui lopulta uneen eikä herännyt, ennen kuin peli käynnistettiin uudelleen. Uneen vaipuminen oli jo ennen testausta tiedostettu kehitysratkaisu, sillä varsinaista pääpeliä pelattaessa lemmikin ei olekaan

tarkoitus enää herätä uudelleen silloin, kun pelaaja on pelaamassa minipeliä, vaan tällöin pelin tulee päättyä.

7.1.3 Tutkimusryhmän kysely

Osa peliprojektin tutkimusryhmässä toimivista henkilöistä suoritti pelien testauksen yleisen pelikokemuksen ja yhteensopivuuden osalta. Testaus suoritettiin kyselyllä, jossa arvioitiin pelien soveltuvuutta eri laitteille ja selaimille. Kyselyn tarkoituksena oli kartoittaa eri laitteilla tapahtuvan pelaamisen eroja. Tämän tutkielman rajoissa kyselyä pidettiin riittävänä, sillä tutkimusryhmällä oli jo valmis käsitys pelin vaatimuksista ja he pystyivät helposti arvioimaan pelien toimintaa eri laitejärjestelmillä. Testatut pelien versiot olivat samat, kuin luvussa 7.1.2 kuvatussa koulutestauksessa.

Kyselyyn vastanneet suorittivat pelien testaamisen 8.5.2015–12.5.2015 välisenä aikana. Vastaajat käyttivät testaukseen omia laitteitaan. Tällä haluttiin saavuttaa laajempi otanta erilaisia laitejärjestelmiä. Testaajia pyydettiin käyttämään mahdollisuuksien mukaan kolmen eri kategorian laitteita, älypuhelinta, tablettia sekä pöytätietokonetta tai kannettavaa. Samalla rohkaistiin myös kokeilemaan sellaisia vanhempia laitteita, jotka eivät olleet enää välttämättä päivittäisessä käytössä.

Kyselyssä kysyttiin testaukseen käytettyjen laitteiden lisäksi seuraavat kysymykset:

1. Oliko pelien suorituskyvyissä havaittavia tai häiritseviä puutteita?
2. Tuliko esiin ongelmia, jotka olisivat haitanneet/häirinneet pelaamista?
3. Oliko testattujen laitteiden välillä eroja, jotka tekivät pelistä tai pelikokemuksesta erilaisen?
4. Millä laitteella oli miellyttävintä pelata ja miksi?
5. Muita huomioita?

Kyselyyn vastasi 4/6 henkilöä (liite A), joiden käyttämät laitteet on listattu taulukossa 4. Vastaajien määrä oli vähäinen, mutta esimerkiksi Myers, Sandler ja Badgett (2012, s. 143–155) esittävät, että syvällisemmässä käytettävyystestauksessa jo neljä testaajaa voisivat löytää lähes 80 % ohjelmiston mahdollisista virheistä. Heidän mukaansa myös testattavan sovelluksen monimutkaisuus vaikuttaa tarvittavan testauksen määrään. Tutkielman pelit koettiin

varsinaisilta toiminnallisuuksiltaan yksinkertaisiksi, eikä niiden teknisen testauksen katsottu vaativan täysimuotoisten testitapausten suunnittelua. Kysely tämä tutkielman puitteissa voidaan katsoa jo onnistuneeksi, jos sen avulla löydetään yhteisiä tekniikan ongelmakohtia, sillä tutkimus itsessään painottaa nimenomaisesti HTML5-tekniikan ongelmattomuutta.

	Käytetyt laitteet	Internet-selaimet
Vastaaja 1.	iPhone 6, iOS 8.3	Safari
	Mac-kannettava	Chrome 42, Firefox 31
	PC-kannettava	Chrome 42
Vastaaja 2.	iPhone 4S, iOS 8.3	Safari
	MacBook mid 2009, OS X 10.10.3	Safari 8.0.5
	Microsoft Surface, Windows RT 8.1	IE 11
Vastaaja 3.	Android-tabletti	Chrome, Firefox
	PC-kannettava, Windows 7	Chrome
	PC-pöytätietokone, Windows 7	Chrome
Vastaaja 4.	Lumia 735, Windows 8.1	IE
	MacBook Pro 13", OS X 10.8.5	Chrome 42

Taulukko 4. Kyselyyn vastanneiden käyttämät laitteet ja selaimet.

Kysymykseen 1. vastaajat ilmoittivat laitteiden toimineen pääsääntöisesti ilman mainittavia suorituskykyongelmia. Vastaajat 3. ja 4. totesivat yhden testaamansa laitteen saavuttaneen vain noin 14–16 ruutua sekunnissa, mutta mainitsivat, ettei se silti vaikuttanut negatiivisesti pelaamiseen. Kysymykseen 2. vastaaja 3. mainitsi, että Perhe-pelin pelikenttä muuttui mustaksi pelattaessa Android-laitteella ja Firefox-selaimella. Vastaajat 1. ja 4. kokivat mangustien ajoittaisen jälkeen jäämisen pelaajasta osittain pelaamista häiritseväksi. Mangustien jälkeen jääminen johtui pelin puutteellisesta tekoälystä ja se korjattiin myöhemmin. Vastaajat 1. ja 2. kokivat pelin valokeilan hyppäykset häiritseviksi, kuten luvun 7.1.2 koulutestauksessa.

Vastaajat eivät kokeneet, että laitteiden välinen pelikokemus olisi poikennut merkittävästi eri laitteilla. Teknisesti pelit miellettiin identtisiksi, mutta syötelaitteiden valinnalla oli joitain eroja. Esimerkiksi hiiren käyttö Perhe-pelissä koettiin hieman vaikeaksi ja vastaajat 2. sekä 4. pitivät sen pelaamista helpoimpana näppäimistöllä. Kosketusnäytöt koettiin yleisesti miellyttäviksi pelata, mutta Perhe-pelissä kielteiseen pelikokemukseen kosketusnäytöillä vaikutti

pelimaailman liikuttaminen ruutu kerrallaan, sillä mm. vastaajan 2. mukaan tämä vaati pelaajalta kosketuksen uudelleen asemoimista ruudulla. Vaikutus ilmeisesti korostui pienillä ruuduilla, sillä älypuhelimien näytöt koettiin häiritsevän pieniksi juuri Perhe-peliin. Tämä tulos ei ole välttämättä yllättävä, sillä mm. Thompson, Nordin ja Cairns (2012) esittävät, että suurempi kosketusnäytön ruutu voi yleisesti katsoen antaa paremman pelikokemuksen.

7.2 Kehityksen arviointi

Tässä luvussa arvioidaan pelien kehityksen aikana vastaan tulleita ongelmia yhteensopivuuden, suorituskyvyn ja käytettävyyden osalta, sekä tarkastellaan pelien toiminnallisten vaatimusten täyttymistä.

Phaser-pelimoottorin käyttöönotto koettiin nopeaksi ja sen avulla pelejä oli mahdollista luoda suhteellisen vähäisellä määrällä varsinaista lähdekoodia. Perhe-pelissä pelkkiä JavaScript-koodirivejä oli noin 500 ja Terveys-pelissä 800. Käytetyt apuohjelmat Grunt ja Texturepacker miellettiin kehityksen aikana hyödyllisiksi, sillä varsinkin Grunt-työkalun todettiin helpottavan pelien testijulkaisuihin käytettävää työmäärää. Käyttöjärjestelmien eroja ei pelejä kehittäessä tai julkaistaessa ollut tarvetta huomioida lainkaan, sillä pelitoteutukset eivät vaikuttaneet olevan riippuvaisia käytetystä järjestelmästä.

Phaserin dokumentaatio todettiin osin puutteelliseksi, sillä joidenkin ominaisuuksien käyttöönotto vaati käytännön kautta testaamista, ennen kuin niiden toimintaa oli mahdollista ymmärtää täysin. Pelien kehityksen HTML5:llä todettiin vaativan lisäksi paljon käytännön testaamista eri laitteilla ja selaimilla, sillä laitteiden käytettävyydessä todettiin olevan eroja, vaikka ne olisivatkin teknisesti samankaltaisia. Testattavien laitteiden tulee olla myös tehokkuudeltaan vaihtelevia, jotta kaikki mahdolliset ongelmat voidaan löytää.

Pelimoottori ei mahdollistanut vaivattomasti käyttöliittymän tai tekstiä sisältävien dialogien piirtämistä. Näiden implementointi itse on mahdollista, mutta vaatii huomattavan määrän työtä, sillä Phaserissa ei ole erillistä piirtojonoa GUI:lle. Käytännössä tämä tarkoittaa sitä, että GUI-elementit tulee Phaserissa piirtää pelimaailman ”sisälle”, muiden näkyvien objektien sekaan. Tällöin kehittäjän on itse pidettävä huoli siitä, että kaikki GUI:n objektit pysyvät ruudulla aina päällimmäisinä ja näkyvillä. Esimerkiksi Perhe-peliin toteutettu rajoitettu nä-

kyvyys peitteen avulla aiheutti kehityksen aikana ongelmia, sillä tekstin piirtäminen `mask`-olion ulkopuolelle ei ollut mahdollista, jos peite asetettiin kerralla koko pelimaailmaan.

Suurten tekstuurien kanssa havaittiin luvussa 7.2.2 kuvattavan suorituskykyyn liittyvän ongelman lisäksi kuvaongelmia. Suuria tekstuureja ylös-alas liikuteltaessa ruudulla näkyi tasaisesti liikkuva kokoruudun levyinen ohut häiriöefekti. Tämä muutaman pikselin korkuinen graafinen virhe ruudulla häiritsi osittain pelaamista, sillä pelaaja ei voinut olla huomaamatta sitä. Virhe toistui kaikilla laitteilla, mutta sen syytä ei selvitetty tarkemmin, koska käyttöön otettiin lopulta pienemmät tekstuurit. Kuvaongelma voinee johtua sekä pelimoottorin tai selaimen piirtovirheestä, sillä se toistui kaikilla laitteilla.

Itse pääpelin lataaminen verkon kautta kaikkine minipeleineen todettiin projektin kohderyhmään nähden hitaaksi. Pääpelissä alkuperäisenä tarkoituksena oli ladata kaikki minipelien resurssit heti peliä aloitettaessa, mutta tiedostojen määrän kasvaessa latausaika oli lopulta hyvin pitkä. Minipelikohtaiset resurssit on mahdollista ladata tilakohtaisesti minipelejä aloitettaessa, jolloin pelaajalle on näytettävä minipeliä ensi kertaa avattaessa latausikkuna/kuva. Tutkielman pelien kohdalla suurimmat minipelikohtaiset resurssit (pääasiassa äänitiedostot) asetettiinkin latautumaan vasta minipeliä avattaessa.

Olisi ollut erittäin toivottavaa, jos pelimoottori olisi mahdollistanut resurssien lataamisen myös pelaamisen aikana. Tällöin pääpelin avauduttua minipelien resursseja olisi voitu ladata taustalla täysin pelaajan tiedostamatta ja aikaa säästäen. Teknistä estettä tälle ei HTML5:n osalta olisi luultavasti ollut, sillä esimerkiksi Ajax-rajapinta on sallinut asynkroniset tiedostolataukset web-sivustoilla jo ennen HTML5-tekniikkaa (Paulson 2005).

Pelin tiedostojen päivittäminen käyttäjille palvelimen kautta ei onnistunut odotetusti, sillä Internet-selaimet eivät aina ladanneet uusimpia tiedostoja suoraan palvelimelta, vaan lukivat ne sen sijaan selaimen välimuistista. Esimerkiksi kuvatietoja määrittelevät JSON-tiedostot jäivät joskus päivittämättä toisin kuin JavaScript-tiedostot, mikä johti ristiriitoihin näiden tiedostojen välisissä kuvaviittauksissa. Ongelma ratkesi viimeistään käyttäjän pakottaessa selain lataamaan sivu uudelleen (engl. *refresh page*) selaimen kontrolleista.

7.2.1 Yhteensopivuus

Varsinaisten testaustilaisuuksien lisäksi pelejä kokeiltiin kehityksen aikana säännöllisesti eri laitteilla ja Internet-selaimilla. Käytössä olleet laitteet käyttöjärjestelmineen ja Internet-selaimineen on listattu taulukossa 5. Kuten aiemmin luvussa 4.4 todettiin, eri Internet-selaimet voivat tukea vielä puutteellisesti kaikkia HTML5:n tarjoamista ominaisuuksista. Varsinkin vanhentuneet versiot selaimista voivat olla huonosti yhteensopivia. Tämän tutkielman puitteissa ei katsottu olevan tarpeellista kartoittaa tarkemmin vanhojen laitteiden ja selainversioiden toimivuutta, ja Internet-selaimista testattiin vain niitä versioita, jotka arvioinnissa käytetyillä laitteilla olivat jo asennettuina.

Laite	**Käyttöjärjestelmä**	**Käytetyt selaimet**
Acer Iconia Tab W500	Windows 7	Chrome 42, IE 10
Ainol Novo Hero	Android 4.3.1	Firefox 37, vakioselain
iMac 27", late 2009	OS X 10.10	Safari 8.0
iPad 1st gen.	iOS 5.1.1	Safari 5.1
iPad 4th gen.	iOS 8.3	Safari 8.0
iPhone 4	iOS 7.1	Safari 7.0
LG L4	Android 4.1.2	Chrome 43, vakioselain
LG L7	Android 4.4.4	Firefox 37, vakioselain
LG L90 D405	Android 4.4.2	Vakioselain
PC	Windows 7	Chrome 42, Firefox 31, IE 11
PC	Windows 8.1	Firefox 37, IE 11
PC (Toshiba P50)	Xubuntu 15.04	Chrome 43, Firefox 37

Taulukko 5. Yhteensopivuuden ja suorituskyvyn arviointiin käytetyt laitteet.

Jotta voitiin vähentää itse pelien ohjelmoinnista johtuvia virheitä yhteensopivuuden arvioinnissa, pelien koodin ei sallittu antavan esimerkiksi JavaScript-virheilmoituksia selaimen konsoliin. Tätä yritettiin saavuttaa huolellisella ohjelmoinnilla ja staattisella koodianalyysillä, kuten luvussa 6.6.1 on kuvattu. Ocariza ym. (2013) ovat kartoittaneet selainsovellusten JavaScript-ohjelmointivirheitä ja todenneet, että niistä jopa 65 % ovat web-sivujen *DOM*-rakenteisiin liittyviä. Näitä tutkielman peleissä ei hallittu ohjelmallisesti. He myös kohtasivat tutkimuksessaan selainkohtaisia virheitä, jotka pahimmillaan saivat selaimen suorituksen

pysähtymään. Vastaavat ongelmat myös tässä tutkielmassa luokitellaan selainten yhteensopivuuden ongelmiksi.

Älypuhelinten ja tablettien tukemissa värinäefekteissä havaittiin puutteita, sillä ne saatiin pääsääntöisesti toimimaan vain Android-järjestelmillä. Lisäksi mobiililaitteet eivät yleensä sallineet kokoruudun tilan asettamista peleille, mikä osalla testatuista selaimista tarkoitti osoitepalkin jatkuvaa näkymistä ruudulla. Kuten luvussa 4.4 todettiin, on kokoruudun tilan puute pelikokemuksen kannalta ongelmallinen. Tämän lisäksi mobiililaitteilla esimerkiksi Firefox-selain piilotti osoitepalkin automaattisesti, mutta palkki tuli kuitenkin näkyviin kosketusruutua ylös tai alas pyyhkäistäessä, mikä häiritsi pelaamista. Ongelma korostui erityisesti älypuhelimilla, sillä pienemmillä ruuduilla vahinkopyyhkäisyjä oli helpompi tehdä. Pöytätietokoneissa vastaavia ongelmia ei havaittu.

Windows-tabletilla käyttäjä menetti pelin kontrollit, mikäli kuvaruudun pyyhkäisy päättyi pelialueen ulkopuolelle. Peli ei enää tämän jälkeen vastannut syötteisiin, ennen kuin käyttäjä oli pienentänyt ja aktivoinut Internet-selaimen uudestaan. Myös tutkimusryhmän testauksessa (liite A) kaksi vastaajista oli kohdannut Windows-mobiililaitteilla saman ongelman. Koska tätä ei havaittu millään muilla laitejärjestelmillä, se voi hyvinkin merkitä laite- ja selainkohtaista yhteensopivuusongelmaa. Syynä voivat toki olla myös juuri Phaser-pelimoottorin puutteet aktiivisen kursorin hallinnassa.

Vanhimman testatun iPadin (1st gen.) yhteensopivuus toteutusratkaisujen kanssa oli kokeilluista laitteista huonoin, ja pelien toteuttaminen sille osoittautui vaikeaksi. Terveys-pelissä esineiden raahaaminen ei onnistunut lainkaan, sillä pelin toiminnallisuus pysähtyi kokonaan, mikäli pelaaja kosketti esineitä. Perhe-peliä laitteella oli kehityksen alkuvaiheessa vielä mahdollista pelata, mutta kun peliin lisättiin äänet, laski ruudunpäivitysnopeus lähes nollaan.

Kehityksen aikana Android-tabletti *Ainol Novo Hero* oli mahdollista saada suuri määrä (yli 300 kpl) pelin objekteja lisäämällä täysin vastaamattomaan tilaan, josta sitä ei voinut sammuttaa edes virtanäppäintä pohjassa pitämällä. Ongelma koettiin toteutuksen kannalta vakavaksi, sillä sovelluksen ei tulisi pystyä milloinkaan kaatamaan koko käyttöjärjestelmää. On kuitenkin epäselvää, johtuiko tämä Internet-selaimesta vai käyttöjärjestelmästä, sillä ongelma oli toistettavissa sekä Firefox- että Chrome-selaimilla. Tapahtunutta pidettiin muuhun

testilaitteistoon nähden poikkeuksellisena ilmiönä.

Peleissä oli myös muiden mobiililaitteiden osalta satunnaista Internet-selainten suorituksen pysähtymistä, kun peliin lisättiin mm. erityisen suuri tiilikartta ja pelin tiloja vaihdettiin useaan kertaan. Ongelman syyt olivat luultavasti selainten muistinhallinnassa tai muissa rajoituksissa, sillä ohjelmallisen virheen pelissä itsessään tulisi normaalitilanteissa vain pysäyttää pelin suoritus. Kohdatuissa virhetilanteissa koko selain sulki itsensä, mikä teki ongelman syyn selvittämisestä hyvin vaikeaa. Lopulta ongelmaa kierrettiin vaihtamalla tarvittaessa virhetilanteita aiheuttavia toteutusratkaisuja.

7.2.2 Suorituskyky

Peleissä käytetyn Phaser-pelimoottorin tilemap-tuki koettiin suorituskyvyltään puutteelliseksi, sillä näkyvän alueen liikuttaminen oli joillain laitteilla ajoittain hidasta. Ongelmaa yritettiin ratkaista kolmannen osapuolen tilemap-lisäosalla (*phaser-tiled*), jonka suorituskyky oli parempi, mutta se ei tukenut haluttua fysiikkamallinnusta tai satunnaisen labyrintin generointia. Jotta pelistä saatiin lopulta useimmilla laitteilla pelattava, kamera asetettiin kuvaamaan staattisesti pelaajan lähiympäristöä. Vasta kun pelaaja siirtyy kuvan reunalle, kameran sijaintia päivitetään ja näytetty alue siirtyy näin pelaajan mukana kuvaruutu kerrallaan. Tämä tarkoittaa väistämättä myös pelihahmon uudelleen sijoittamista ruudulla näytettävää aluetta vaihdettaessa.

Taulukossa 6 on Perhe-pelin mittaustulokset keskimääräisen ruudunpäivitysnopeuden osalta. Mittauksessa peliin luotiin sama labyrintti ja sitä pelattiin jokaisella laitteella identtisesti hetken aikaa. Mittaus on oletettavasti raskaampi, kuin luvussa 6.6.2 suoritetussa resoluution valinnassa, sillä Perhe-peliä mitattaessa mukana oli enemmän kuvia ja pelilogiikkaa (esim. fysiikanmallinnus). Terveys-pelin suorituskykyä ei mitattu erikseen, mutta yksinkertaisuutensa vuoksi sen voidaan olettaa olevan Perhe-peliä hieman parempi.

Mittaustulosten mukaan PC-koneet olivat pääsääntöisesti hyvin suorituskykyisiä, eikä pelattavuuteen vaikuttavia ongelmia juurikaan havaittu. Vastakohtaisesti taas osa testatuista mobiililaitteista ei kyennyt ruudunpäivitysnopeudeltaan vähimmäisvaatimukseksi asetettuun 15 ruutuun sekunnissa. Varsinkin vanhemmilla mobiililaitteilla oli selkeitä suorituskykyongel-

Tabletit	**Julkaisuvuosi**	**Selain**	**Ruutua/sekunti**
Acer Iconia Tab W500	2011	Chrome 42	26
		Internet Explorer 10	53
Ainol Novo Hero	2012	Firefox 37	11
		Vakioselain	9
iPad 1st gen.	2010	Safari 5.1	17*
iPad 4th gen.	2012	Safari 8.0	30
Älypuhelimet	**Julkaisuvuosi**	**Selain**	**Ruutua/sekunti**
iPhone 4	2010	Safari 7.0	15
LG L4	2013	Chrome 43	11
		Vakioselain	10
LG L7	2012	Firefox 37	8
LG L90 D405	2014	Vakioselain	39
PC / Mac	**Julkaisuvuosi**	**Selain**	**Ruutua/sekunti**
iMac 27"	2009	Safari 8.0	37
PC (Windows 7)	—	Chrome 42	60
		Firefox 31	20
		Internet Explorer 11	64
PC (Windows 8.1)	—	Chrome 40	60
		Firefox 37	60
		Internet Explorer 11	60
PC (Xubuntu 15.04)	2013	Chrome 43	60
		Firefox 37	60

*äänten lisääminen peliin teki pelaamisesta mahdotonta

Taulukko 6. Perhe-pelin suorituskyky eri laitteilla.

mia, jotka mitä ilmeisimmin johtuivat puutteellisesta laitteistokiihdytyksestä, sillä käyttöjärjestelmä ja Internet-selain olivat usein joka tapauksessa ajan tasalla. Joko laitevalmistajat eivät päivitä riittävästi vanhempien laitteidensa tukemia ominaisuuksia tai laitteiden komponentit itsessään eivät yksinkertaisesti mahdollista parempaa suorituskykyä, sillä kuten luvussa 4.5 mainittiin, on mobiililaitteiden GPU usein varsin heikkotehoinen.

Pelien suorituskykyä olisi voitu oletettavasti parantaa käyttämällä pienempää resoluutiota ja tekstuureja. Tämä olisi toisaalta vaikuttanut negatiivisesti pelien ulkonäköön, eikä resoluutio olisi enää vastannut teknisiä vaatimuksia. Pienempi resoluutio olisi vaihtoehtoisesti voitu ottaa käyttöön vain silloin, kun peliä olisi pelattu pieniruutuisella laitteella (esim. älypuhelin). Teknisesti kahden tai useamman natiiviresoluution tukeminen olisi vaatinut lisätyötä, eikä sitä sen johdosta toteutettu. Oletettavasti myös joitain erillisiä lähdekoodin optimointikeinoja olisi voitu ottaa käyttöön pelien suorituskyvyn parantamiseksi.

7.2.3 Perhe-pelin vaatimusten toteutuminen

Perhe-pelin toteutus saavutti valitulla pelimoottorilla sille määritetyt toiminnalliset vaatimukset, poislukien luvussa 7.2.2 mainittu tiilikarttojen suorituskyvyn puute, joka esti pelaajaa jatkuvasti seuraavan kameran käyttämisen. Tämä vaikutti testauksen perusteella negatiivisesti myös pelikokemukseen. Sisällöllisesti pelin alkuperäinen suunnitelma toteutui ja järjestetyissä testauksissa peliin saatiin vielä lisää kehitysideoita. Kehityksen aikana suurin haaste liittyi pelin jäljellä olevan ajan näyttöön. Phaser-pelimoottori ei tarjoa kunnollista reaaliaikaista ajanseurantaa, vaan useimmat `Timer`-olion metodit on sidottu suoraan ruudunpäivitysnopeuteen. Tämä tuli kehityksen aikana suurena yllätyksenä, sillä vasta vähemmän tehokkailla laitteilla testattaessa huomattiin pelin jäljellä olevan ajan juoksevan selkeästi muita laitteita nopeammin.

7.2.4 Terveys-pelin vaatimusten toteutuminen

Terveys-pelin toteutus täytti sille asetetut vaatimukset. Tweenit olisivat kaivanneet muutamia yksinkertaisia metodeja, joilla niiden ketjutusta toisiinsa olisi voitu hallita tehokkaammin, sillä varsinkin eri `Sprite`-olioiden tweenien ketjutus ja samanaikainen ajoittaminen olisi ollut hyödyllinen ominaisuus. Peli ei kärsinyt suurista suorituskykyongelmista, mikä saattoi johtua pelin liikkuvien tekstuurien vähäisestä määrästä tai siitä, että pelissä ei ollut fysikaalista tai muuta raskasta laskentaa. Suorituskyvystä johtuen peli oli myös testauksen perusteella Perhe-peliin verrattuna miellyttävämpi pelata mobiililaitteilla.

7.3 Yleinen arvio ja jatkokysymykset

Tutkielman pelit kehitettiin yhden avoimen lähdekoodin HTML5-pelimoottorin avulla, joten kohdattuja ongelmia ei voida yksiselitteisesti yleistää koko HTML5-tekniikkaan. Osa kehityksen aikana esiin tulleista ongelmista oli ainakin joiltain osin selkeästi Internet-selainten syytä. Selaimet saatiin esimerkiksi kaatumaan todella suurilla tiilikartoilla, vaikka tämän ei tulisi pelisovelluksen puolesta olla mahdollista. Joillain mobiililaitteilla selaimet ja/tai käyttöjärjestelmät häiritsivät pelaamista mm. kontrollipalkkien esiin tulemisella. Näihin ongelmiin valitulla pelimoottorilla ei välttämättä pystytä vaikuttamaan ja ne voitaneen listata itse HTML5-tekniikan puutteiksi.

Selainvalmistajat tulevat todennäköisesti korjaamaan näitä edellä mainittuja ongelmia tulevissa julkaisuissaan. Kuten luvussa 4.4 mainittiin, on selainten HTML5-tuki parantunut tasaisesti jokaisen uuden version myötä. Pelaajan kannalta onkin tärkeää, että käytössä olisi aina uusin mahdollinen selain. Kohdatuista yhteensopivuuden ongelmista huolimatta pelit olivat useimmilla laitteilla täysin pelattavia. Varsinkaan kannettavilla tai pöytätietokoneilla ei pääsääntöisesti esiintynyt pelaamista haittaavia tekijöitä.

Myös osa mobiililaitteista toimi ongelmattomasti ja testausten perusteella pelaaminen niillä oli kosketusnäytön ansiosta miellyttävää. Poikkeuksena tähän olivat pieniruutuiset älypuhelimet toista toteutettua peliä pelattaessa. Yhteensopivuuden osalta ei voida eritellä yksittäisiä Internet-selaimia tai käyttöjärjestelmiä, joilla pelit eivät nimenomaisesti olisi toimineet, sillä ongelmat koskettivat sekalaisesti vain joitain testatuista laitteita. Tulosten pohjalta syntynyt yleinen arvio on, että laitteiden iällä voi olla vaikutusta käytettävyyden ja yhteensopivuuden ongelmiin.

Suorituskykyyn vaikuttavat ongelmat ovat vaikeammin tulkittavissa. Valitun pelimoottorin piirtonopeudessa oli puutteita mm. tiilikarttojen osalta, sillä vaihtoehtoinen ohjelmistokirjasto olisi ollut alkuperäistä suorituskykyisempi. Jo pelien resoluutioiden kartoitus luvussa 6.6.2 paljasti, ettei pelimoottorilta välttämättä voida odottaa erinomaista suorituskykyä mobiililaitteilla, vaikka graafinen esitys olisikin yksinkertainen. Koska kaikki laitteet eivät joka tapauksessa vielä täysin tue WebGL:ää, on mahdollista, etteivät myöskään vaihtoehtoiset HTML5-pelikirjastot kykenisi vastaavissa tilanteissa tutkielman toteutusratkaisua parem-

paan suorituskykyyn.

Ongelma ratkennee väistämättä tulevaisuudessa uusien laitteiden myötä, mutta pelikehittäjien on määriteltävä ja testattava pelit huolellisesti juuri niitä laitteita silmällä pitäen, jotka valtaosalla pelaajista ovat yhä käytössä. Tutkielman pelien suorituskykyä olisi ollut luultavasti mahdollista parantaa entisestään optimointien avulla, mikä olisi voinut tehdä peleistä pelattavampia mobiililaitteilla. Tähän kuluvaa työmäärää tai optimoinnin onnistumista on vaikea arvioida etukäteen ja se saatetaan vielä toteuttaa myöhemmin tutkielman peleille. Peleille tehtyä testausta voidaan pitää onnistuneena, sillä niiden avulla löydettiin useita pelaamista haittaavia häiriötekijöitä sekä yhteensopivuusongelmia. Tutkimusryhmälle osoitettu kysely tuotti myös tietoa, jota ei peliprojektin palavereissa oltu aiemmin nostettu esille.

Kokonaisuutena pelien toteutusta käytetyllä HTML5-tekniikalla voidaan pitää onnistuneena, sillä pelien vaatimukset täytettiin kaikkien muiden, paitsi suorituskyvyn ja yhteensopivuuden osalta mobiililaitteilla. Teknisten vaatimusten puutteet eivät kuitenkaan välttämättä kosketa kaikkia mobiililaitteita, sillä osa tablet-laitteista soveltui pelaamiseen ongelmitta. Luvussa 4.9 esitettiin ongelmallisille HTML5-peleille julkaisuratkaisuksi mm. CocoonJS-työkalua, joka luo pelistä hybridisovelluksen puhtaan web-sovelluksen sijaan. Tutkielman peleille tämä ei ollut vaihtoehtona, sillä pelien vaatimukset painottivat, ettei pelejä tarvitsisi erikseen asentaa.

Tutkielman aikana nousi esille monia mielenkiintoisia kysymyksiä, jotka saattaisivat soveltua jatkotutkimusten aiheiksi. Esimerkiksi 3D-peleihin liittyvät mahdollisuudet rajattiin tämän tutkielman ulkopuolelle, sillä niihin liittyvä selainteknologia koettiin osittain keskeneräiseksi. On kuitenkin selvää, että kolmiulotteiset HTML5-pelit tulevat vielä ennen pitkää tekemään läpimurron. Tästä todisteena ovat WebGL:ää hyödyntävät teknologiademot (esim. *Quake 3*[1]), jotka näyttävät jo kykenevän täysin pelattavan 3D-grafiikan esittämiseen (Charland ja Leroux 2011). Olisikin hyödyllistä tutkia, mikä on kolmiulotteisten HTML5-pelien todellinen nykytilanne ja mille osa-alueille tekniikka vielä kaipaa kehitystyötä.

Toinen tutkimuskysymys liittyy selainpelaamisen yleisiin tulevaisuudennäkymiin, sillä onko todella mahdollista, että kaikki videopelit siirtyisivät ennen pitkää puhtaasti Internetissä pe-

1. `http://media.tojicode.com/q3bsp/`

lattaviksi? Jos selainteknologia todella kykenee saavuttamaan (lähes) asennettavia videopelejä vastaavan suorituskyvyn, miksi pelejä enää ylipäätään kehitettäisiin joitain poikkeuksia lukuun ottamatta muilla tekniikoilla? Selainpelaaminen merkitsee samalla pelimarkkinoiden painopisteen siirtymistä pelin ostamisesta, pelin sisällä tapahtuvaan rahoitusmalliin. Olisi myös mielenkiintoista tietää, mitä tämä merkitsee piratismille, sillä mikäli pelin sisältöä ladataan käyttäjälle vain tarpeen mukaan, vaikeuttanee se pelien laitonta levitystä.

Mikäli tutkimusta halutaan jatkaa kaksiulotteisista HTML5-peleistä, lienee yksi ajankohtaisimmista siihen liittyvistä aiheista moninpelaaminen. Moninpelaamista HTML5:llä olisi tarpeen tutkia tarkemmin latenssien, rajapintojen, käyttäjien määrän ja yleisen pelattavuuden kannalta. Moninpelaaminen on hyvin tärkeä osa juuri selainpelejä, sillä kuten luvussa 2.1 todettiin, ovat sosiaaliset elementit tärkeitä varsinkin ajanvietepelien suosiolle.

Aihe herättää myös mielenkiintoisia kysymyksiä tietoturvasta. Mikäli tulevaisuudessa on todella mahdollista pelata kaikkia pelejä suoraan selaimesta ilman pelien asentamista, tarkoittaako tämä sitä, ettei pelejä voida käyttää enää lainkaan haittaohjelmien levitykseen? Vai alistaako HTML5 päinvastoin pelaajien tietokoneet täysin uusille uhkakuville? JavaScriptin dynaamisuus voi osaltaan tarkoittaa sitä, että haittaohjelmia saattaa olla mahdollista lisätä suoraan pelikoodiin. Myös peleissä huijaaminen voinee mahdollistua juuri dynaamisuuden vuoksi. Web-standardit ovat avoimen lähdekoodinsa ja valtavien käyttäjämääriensä vuoksi joka tapauksessa hyvin ylläpidettyjä ja suurimmat riskit voinevat kohdistua niihin käyttäjiin, jotka käyttävät vanhentuneita selaimia ja käyttöjärjestelmiä.

Tärkeimmäksi jatkotutkimuksen aiheeksi voitaneen joka tapauksessa nostaa yhä HTML5-tekniikan todellinen monialustaisuus. HTML5:n vahvuus piilee sen monialustaisuuden mahdollisuuksissa, joka ei vaadi asentamista ja koko teknologian tulevaisuus onkin siihen vahvasti sidoksissa. Tässä tutkielmassa keskityttiin mobiili- ja pöytätietokoneiden väliseen yhteensopivuuteen, mutta HTML5-pelejä tulisi pystyä pelaamaan kaikilla muillakin Internet-selaimen sisältävillä laitteilla. Olisikin hyödyllistä tehdä laajempi selvitys HTML5-pelien yhteensopivuudesta erilaisille laitejärjestelmille.

8 Johtopäätökset

Tässä tutkielmassa luotiin ensin katsaus nykypäivän pelaamisen tarpeisiin ja pelimarkkinoihin. Laitekohtainen pelinkehitys todettiin monin tavoin ongelmalliseksi, sillä pelejä pelataan hyvin erilaisilla järjestelmillä, joilla kaikilla on omat julkaisukanavansa ja kehitystyökalunsa sovelluksia varten. Pelinkehitys perinteisillä tekniikoilla voi olla työlästä, jos pelit halutaan kaikilla eri järjestelmillä pelaavien ihmisten saataville. Monialustatekniikat ovat usein keskittyneet joko yksinomaan mobiililaitteisiin tai perinteisiin tietokoneisiin, eivätkä mahdollista helppoa julkaisua ja kehitystä yhtä aikaa molemmille edellä mainituista laitejärjestelmistä.

Tutkielmassa esiteltiin HTML5-tekniikka, jolla pelejä voidaan ideaalitapauksissa pelata suoraan Internet-selaimista ilman erillistä asentamista kaikilla niillä laitteilla, joissa on selain. Tutkielmassa toteutettiin Jyväskylän yliopiston tutkimushankkeen peliprojektin vaatimuksiin pohjautuen kaksi HTML5-peliä, joiden kehitystä ja laadullisia ominaisuuksia arvioitiin. Pelit kehitettiin avoimen HTML5-pelimoottorin avulla. Tulosten mukaan peleissä oli yhteensopivuus- ja suorituskykyongelmia mobiililaitteiden osalta. Mobiililaitteilla oli myös havaittavissa käytettävyysongelmia, sillä osa testatuista selaimista mm. avasi selaimen kontrollipaneelin ruutua huolimattomasti pyyhkäistäessä.

Kannettavat ja pöytätietokoneet olivat arvioiduista laitteistoista parhaiten toteutettuihin peleihin soveltuvia, eivätkä kärsineet ongelmista, jotka olisivat merkittävästi vaikuttaneet pelikokemukseen. Näillä laitteilla HTML5:tä voidaankin jo pitää kilpailukykyisenä monialustaratkaisuna ainakin pienten pelitoteutusten kehitykseen. Mobiililaitteiden osalta testattu tekniikka vaatii vielä erinäisten ongelmien, kuten älypuhelinten suorituskyvyn parannuksia, ennen kuin tutkielmassa käytettyjä HTML5-tekniikoita voidaan kokonaisuudessaan pitää kykenevänä varsinkaan kaupallisten pelijulkaisujen vaatimuksiin.

Toteutettujen pelien kohtaamat mobiiliongelmat ovat vaatimuksiin nähden ristiriitaisia, mutta on kuitenkin muistettava, ettei toista yhtä helpon pelijulkaisutavan mahdollistavaa monialustaratkaisua ole olemassa. Tutkielmassa kuvatun peliprojektin vaatimuksiin HTML5 oli kartoituksen perusteella käytännössä ainoa tekniikka, jolla pelit oli mahdollista saada vaivattomasti lasten pelattaviksi. Toiminnallisilta vaatimuksiltaan toteutettujen pelien tavoitteet

saavutettiin, mutta tekniset vaatimukset jäivät osalla mobiililaitteista paikoin täyttämättä.

Tutkielman keskeisenä tavoitteena oli löytää vastaus kysymykseen, onko HTML5 varteenotettava vaihtoehto monialustaisten 2D-pelien toteuttamiseen? Tutkimuksen tulosten pohjalta vastaus on kyllä, huolimatta tutkielmassa kohdatuista ongelmista. Tekniikka on vasta aloittamassa maailmanvalloitustaan, eivätkä sen alkutaipaleen ongelmat yksittäisillä laitteilla tarkoita, että koko tekniikka tulisi tuomita toimimattomaksi. Yksinkertaisten monialustaisten ajanvietepelien toteuttamiseksi HTML5 voinee jo lähitulevaisuudessa olla ensisijainen tekniikka, jolla pelit saadaan kehitettyä kerralla kaikille mahdollisille laitejärjestelmille.

HTML5:n mahdollisuudet kaupallisiin monialustajulkaisuihin ovat vielä osin hämärän peitossa, sillä pelien julkaisukanavat loistavat yhä poissaolollaan. JavaScriptin avoin luonne voi tehdä pelien lähdekoodin turvaamisen vaikeaksi ja altistanee pelit esimerkiksi moninpeleissä tapahtuvalle huijaamiselle. Monet pelinkehittäjät eivät välttämättä myöskään halua paljastaa pelin sisäistä logiikka kilpailijoilleen. Toisaalta osa HTML5-pelien toiminnoista voidaan sijoittaa palvelimelle, jolloin pelaajilla ei ole mahdollisuutta päästä niihin käsiksi.

HTML5:n tulevaisuus vaikuttaisi erittäin valoisalta, jos selainvalmistajat kykenevät kuluvina vuosina tuomaan tukemansa standardit sille tasolle, ettei käytetyllä selaimella olisi pelaamisen kannalta mitään merkitystä. Myös mobiililaitevalmistajien on kyettävä vastaamaan pelien tuomiin laitteistovaatimuksiin ja varmistettava riittävä tuki tarvittavalle laitteistokiihdytykselle. HTML5:n yhteensopivuus kaikkien mahdollisten laitejärjestelmien kesken kaipaa yhä jatkotutkimusta, sillä tarve monialustaisuudelle kasvaa jatkuvasti uusien pelilaitteiden myötä. Tämä tutkielma tutki tekniikan soveltuvuutta yhden peliprojektin osalta, eikä voi näin eritellä yksityiskohtaisesti yhteensopivuuden keskeisimpiä ongelmia.

Avoimen lähdekoodin pelimoottorit ja työkalut toimivat omalta osaltaan todisteena HTML5:n kehityksestä ja siitä, että tekniikalle on jo merkittävästi kysyntää. Nazarov ja Galletly (2013, s. 24) toteavatkin esimerkiksi WebGL:stä, että ilmaista työtä tekevät harrastelijat pakottavat omalla työllään tämän tekniikan väistämättä kehittymään. Nämä yhteisöt ja käyttäjät ovat myös osaltaan se kantava voima, jotka ajavat teknologian tulevaisuutta eteenpäin, sillä hyvät ohjelmistokirjastot, työkalut ja dokumentaatio ovat lopulta yhdessä se, jotka mahdollistavat monipuolisen ja laadukkaan pelikehityksen.

Lähteet

Achilleos, Achilleas P., ja Georgia M. Kapitsaki. 2014. “Enabling Cross-Platform Mobile Application Development: A Context-Aware Middleware”. Teoksessa *Web Information Systems Engineering–WISE 2014,* 304–318. Springer. Viitattu 12. maaliskuuta 2015. http://link.springer.com/chapter/10.1007/978-3-319-11746-1_22.

Aho, Eero, Kimmo Kuusilinna, Tomi Aarnio, Janne Pietiäinen ja Jari Nikara. 2012. “Towards real-time applications in mobile web browsers”. Teoksessa *Embedded Systems for Real-time Multimedia (ESTIMedia), 2012 IEEE 10th Symposium on,* 57–66. IEEE. Viitattu 23. tammikuuta 2015. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6507030.

Andersson, Tobias, ja Erik Johansson. 2014. “A closer look and comparison of cross-platform development environment for smartphones”. Viitattu 6. huhtikuuta 2015. http://www.diva-portal.org/smash/record.jsf?pid=diva2:726850.

Anttonen, Matti, Arto Salminen, Tommi Mikkonen ja Antero Taivalsaari. 2011. “Transforming the Web into a Real Application Platform: New Technologies, Emerging Trends and Missing Pieces”. Teoksessa *Proceedings of the 2011 ACM Symposium on Applied Computing,* 800–807. ACM. Viitattu 24. helmikuuta 2015. http://dl.acm.org/citation.cfm?id=1982357.

Apple. 2015. “App Store Review Guidelines”. Viitattu 26. huhtikuuta 2015. https://developer.apple.com/app-store/review/guidelines/.

Asm.js. 2015. Viitattu 18. huhtikuuta 2015. http://asmjs.org/.

Blewitt, William, Gary Ushaw ja Graham Morgan. 2013. “Applicability of gpgpu Computing to Real-Time AI Solutions in Games”. *Computational Intelligence and AI in Games, IEEE Transactions on* 5 (3): 265–275. Viitattu 5. huhtikuuta 2015. http://elysianshadows.com/updates/wp-content/uploads/2011/05/cpe790FINAL.pdf.

Bouras, Christos, Andreas Papazois ja Nikolaos Stasinos. 2014. “A Framework for Cross-platform Mobile Web Applications Using HTML5”. Teoksessa *2014 International Conference on Future Internet of Things and Cloud,* 420–424.

Browne, Kevin, ja Christopher Anand. 2012. “An empirical evaluation of user interfaces for a mobile video game”. *Entertainment Computing* 3 (1): 1–10.

Cai, Wei, Victor CM Leung ja Min Chen. 2013. “Next Generation Mobile Cloud Gaming”. Teoksessa *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on,* 551–560. IEEE. Viitattu 17. huhtikuuta 2015. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6525574`.

Charaf, Hassan, Péter Ekler, Tamás Mészáros, Imre Kelényi, Bence Kovari, István Albert, Bertalan Forstner ja László Lengyel. 2014. “Mobile Platforms and Multi-Mobile Platform Development”. *Acta Cybernetica* 21:529–552. Viitattu 12. maaliskuuta 2015. `http://www.inf.u-szeged.hu/actacybernetica/edb/vol21n4/pdf/Charaf_2014_ActaCybernetica.pdf`.

Charland, Andre, ja Brian Leroux. 2011. “Mobile Application Development: Web vs. Native”. *Communications of the ACM* 54 (5): 49–53. Viitattu 12. maaliskuuta 2015. `http://dl.acm.org/citation.cfm?id=1941504`.

Claypool, Mark, ja Kajal Claypool. 2009. “Perspectives, Frame Rates and Resolutions: It’s all in the Game”. Teoksessa *Proceedings of the 4th International Conference on Foundations of Digital Games,* 42–49. ACM. Viitattu 24. huhtikuuta 2015. `http://dl.acm.org/citation.cfm?id=1536530`.

Cocos2d-X. 2015. *Cocos2d-x Programmers Guide v3.3.* Viitattu 13. huhtikuuta 2015. `http://www.cocos2d-x.org/programmersguide/ProgrammersGuide.pdf`.

Construct 2. 2015. Viitattu 13. huhtikuuta 2015. `http://www.scirra.com`.

Corral, Luis, Alberto Sillitti ja Giancarlo Succi. 2012. “Mobile multiplatform development: An experiment for performance analysis”. *Procedia Computer Science* 10:736–743. Viitattu 16. maaliskuuta 2015. `http://www.sciencedirect.com/science/article/pii/S1877050912004516`.

Corral, Luis, Alberto Sillitti, Giancarlo Succi, Alessandro Garibbo ja Paolo Ramella. 2011. "Evolution of Mobile Software Development from Platform-Specific to Web-Based Multiplatform Paradigm". Teoksessa *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software,* 181–183. ACM. Viitattu 10. helmikuuta 2015. https://dl.acm.org/citation.cfm?id=2157457.

Curran, Kevin, ja Ciaran George. 2012. "The Future of Web and Mobile Game Development". *International Journal of Cloud Computing and Services Science (IJ-CLOSER)* 1 (1): 25–34. Viitattu 6. helmikuuta 2015. http://iaesjournal.com/online/index.php/IJ-CLOSER/article/view/233.

Curran, Michael, Nigel McKelvey, Kevin Curran ja Subaginy Nadarajah. 2015. "Mobile App Stores". Viitattu 31. maaliskuuta 2015. doi:10.4018/978-1-4666-5888-2.ch561. http://scisweb.ulster.ac.uk/~kevin/papers/IGI2014-mobileappstores.pdf.

Di Loreto, Ines, ja Abdelkader Gouaïch. 2010. "Social casual games success is not so casual". Viitattu 23. maaliskuuta 2015. http://hal-lirmm.ccsd.cnrs.fr/lirmm-00486934.

Ding, Jonathan, Yuqiang Xian, Yongnian Le, Kangyuan Shu, Haili Zhang ja Jason Zhu. 2012. "GRAPHICS ACCELERATION FOR HTML5 AND JAVASCRIPT ENGINE JIT OPTIMIZATION FOR MOBILE DEVICES". *Intel Technology Journal* 16 (4): 164–177.

EaselJS. 2015. Viitattu 13. huhtikuuta 2015. http://www.createjs.com/EaselJS.

Evans, Elizabeth. 2015. "The economics of free freemium games, branding and the impatience economy". *Convergence: The International Journal of Research into New Media Technologies:* 1354856514567052. Viitattu 12. maaliskuuta 2015. http://con.sagepub.com/content/early/2015/02/18/1354856514567052.abstract.

Feijoo, Claudio, José-Luis Gómez-Barroso, Juan-Miguel Aguado ja Sergio Ramos. 2012. "Mobile gaming: Industry challenges and policy implications". *Telecommunications Policy* 36 (3): 212–221. Viitattu 11. maaliskuuta 2015. http://www.sciencedirect.com/science/article/pii/S0308596111002242.

Flanagan, David. 2011. *JavaScript: The Definitive Guide.* 6. painos. O'Reilly Media, Inc. ISBN: 978-0-596-80552-4.

GameMaker. 2015. Viitattu 13. huhtikuuta 2015. http://docs.yoyogames.com/.

Garaizar, Pablo, MA Vadillo ja Diego Lopez-de-Ipina. 2012. "Benefits and pitfalls of using HTML5 APIs for online experiments and simulations". Teoksessa *Remote Engineering and Virtual Instrumentation (REV), 2012 9th International Conference on,* 1–7. IEEE.

Gaudiosi, John. 2015. "Mobile game revenues set to overtake console games in 2015". Tammikuu. Viitattu 14. maaliskuuta 2015. http://fortune.com/2015/01/15/mobile-console-game-revenues-2015/.

Gawley, Rachel, Jonathan Barr ja Michael Barr. 2012. "Native to HTML5: A Real-World Mobile Application Case Study". Teoksessa *Mobile Computing, Applications, and Services,* toimittanut JoyYing Zhang, Jarek Wilkiewicz ja Ani Nahapetian, 95:188–206. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg. ISBN: 978-3-642-32319-5, viitattu 6. helmikuuta 2015. http://dx.doi.org/10.1007/978-3-642-32320-1_13.

Gong, Liang, Michael Pradel, Manu Sridharan ja Koushik Sen. 2015. *DLint: Dynamically Checking Bad Coding Practices in JavaScript.* Tekninen raportti. Technical Report UCB/EECS-2015-5, EECS Department, University of California, Berkeley. Viitattu 9. toukokuuta 2015. http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-5.pdf.

Google. 2015. "Google Play Terms of Service". Viitattu 26. huhtikuuta 2015. https://play.google.com/about/android-developer-policies.html.

Gregory, Jason. 2014. *Game Engine Architecture.* CRC Press. ISBN: 978-1-4665-6006-2.

Grunt. 2015. Viitattu 15. huhtikuuta 2015. http://gruntjs.com/.

Grupp, Jonathan. 2012. "WebGL and other Technologies for hardware accelerated 3D in Browsers". Viitattu 4. maaliskuuta 2015. http://www.dictatemusic.com/WebGL.SotA_no%20literature.pdf.

Hartmann, Gustavo, Geoff Stead ja Asi DeGani. 2011. “Cross-platform mobile development”. *Mobile Learning Environment, Cambridge.* Viitattu 7. maaliskuuta 2015. https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf.

Hass, Anne Mette Jonassen. 2008. *Guide to Advanced Software Testing.* Artech House. ISBN: 978-1-59693-285-2.

Heitkötter, Henning, Sebastian Hanschke ja Tim A. Majchrzak. 2013. “Evaluating Cross-Platform Development Approaches for Mobile Applications”. Teoksessa *Web information systems and technologies,* 120–138. Springer. Viitattu 24. helmikuuta 2015. http://link.springer.com/chapter/10.1007/978-3-642-36608-6_8.

Hevner, Alan R., Salvatore T. March, Jinsoo Park ja Sudha Ram. 2004. “Design Science in Information Systems Research”. *MIS quarterly* 28 (1): 75–105. Viitattu 22. huhtikuuta 2015. http://www.springerlink.com/index/pdf/10.1007/s11576-006-0028-8.

Holzer, Adrian, ja Jan Ondrus. 2011. “Mobile application market: A developer’s perspective”. *Telematics and informatics* 28 (1): 22–31. Viitattu 1. huhtikuuta 2015. http://www.sciencedirect.com/science/article/pii/S0736585310000377.

Holzinger, Andreas, Peter Treitler ja Wolfgang Slany. 2012. “Making Apps Useable on Multiple Different Mobile Platforms: On Interoperability for Business Application Development on Smartphones”. Teoksessa *Multidisciplinary research and practice for information systems,* 176–189. Springer. Viitattu 16. maaliskuuta 2015. http://link.springer.com/chapter/10.1007/978-3-642-32498-7_14.

HTML5test. 2015. “How well does your browser support HTML5?” Viitattu 26. huhtikuuta 2015. https://html5test.com/.

IDC. 2015. “Smartphone OS Market Share, Q4 2014”. Viitattu 18. maaliskuuta 2015. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

ISO/IEC. 2000. *Information technology — Software product quality — Part 1: Quality model 9126-1.*

Jeon, Won, Tasneem Brutch ja Simon Gibbs. 2012. “WebCL for Hardware-Accelerated Web Applications”. Teoksessa *TIZEN Developer Conference May,* 7–9.

Juntunen, A., E. Jalonen ja S. Luukkainen. 2013. “HTML 5 in Mobile Devices – Drivers and Restraints”. *System Sciences (HICSS), 2013 46th Hawaii International Conference on* (tammikuu): 1053–1062. ISSN: 1530-1605, viitattu 23. tammikuuta 2015. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6479961.

Juul, Jesper. 2010. *A casual revolution: Reinventing video games and their players.* MIT press. ISBN: 978-0-262-01337-6.

Järvinen, P., ja A. Järvinen. 2004. *Tutkimuksen metodeista.* Tampereen Yliopistopaino Oy. ISBN: 952-99233-2-5.

Kasari, Olli. 2012. “Käyttöliittymäkehitys kosketuskäyttöiselle älypuhelimelle”. Tutkielma, Jyväskylän yliopisto. Viitattu 28. helmikuuta 2015. http://urn.fi/URN:NBN:fi:jyu-201302131215.

Khan, Faiz, Vincent Foley-Bourgon, Sujay Kathrotia, Erick Lavoie ja Laurie Hendren. 2014. “Using JavaScript and WebCL for numerical computations: a comparative study of native and web technologies”. Teoksessa *Proceedings of the 10th ACM Symposium on Dynamic languages,* 91–102. ACM. Viitattu 30. tammikuuta 2015. http://dl.acm.org/citation.cfm?id=2661090.

Kultima, Annakaisa. 2015. “Online Games, Casual”. *The International Encyclopedia of Digital Communication and Society.* Viitattu 16. maaliskuuta 2015. http://onlinelibrary.wiley.com/doi/10.1002/9781118767771.wbiedcs107/full.

Kuuskeri, Janne, ja Tommi Mikkonen. 2009. “Partitioning Web Applications Between the Server and the Client”. Teoksessa *Proceedings of the 2009 ACM symposium on Applied Computing,* 647–652. ACM. Viitattu 28. maaliskuuta 2015. http://dl.acm.org/citation.cfm?id=1529416.

Li, Xianfeng, ja Zhiqiang Bao. 2014. “Performance Characterization of Web Applications with HTML5 Enhancements”. Teoksessa *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on,* 252–258. IEEE. Viitattu 23. tammikuuta 2015. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6945697.

Ludei. 2015. “HTML5 deployment platform”. Viitattu 26. huhtikuuta 2015. https://www.ludei.com/cocoonjs/.

Marmalade. 2015. Viitattu 22. huhtikuuta 2015. www.madewithmarmalade.com.

Marszałkowski, Jakub. 2011. “The importance of advertising exchange for marketing browser games”. *Homo Ludens* 3 (1): 103–116.

MelonJS. 2015. Viitattu 25. huhtikuuta 2015. http://melonjs.org/.

Microsoft. 2015. “WebGL (Windows)”. Viitattu 4. huhtikuuta. https://msdn.microsoft.com/en-us/library/ie/bg182648(v=vs.85).aspx.

MightyFingers. 2015. Viitattu 13. huhtikuuta 2015. http://mightyfingers.com/.

Mikkonen, Tommi, ja Antero Taivalsaari. 2007. “Using JavaScript as a Real Programming Language”. Viitattu 17. huhtikuuta 2015. http://dl.acm.org/citation.cfm?id=1698202.

Moreira, Átila V. M., Vicente V. Filho ja Geber L. Ramalho. 2014. “Understanding mobile game success: a study of features related to acquisition, retention and monetization”. *SBC* 5 (2). Viitattu 26. huhtikuuta 2015. http://www.seer.ufrgs.br/jis/article/viewFile/45696/31858.

Myers, Glenford J, Corey Sandler ja Tom Badgett. 2012. *The art of software testing.* 3. painos. John Wiley & Sons. ISBN: 978-1-118-03196-4.

Mäyrä, Frans. 2015. “18. The conflicts within the casual: The culture and identity of casual online play”. Teoksessa *Playful Identities,* 321–333. Amsterdam University Press.

Mäyrä, Frans, ja Laura Ermi. 2014. “Pelaajabarometri 2013: Mobiilipelaamisen nousu”. Viitattu 16. huhtikuuta 2015. http://urn.fi/URN:ISBN:978-951-44-9425-3.

Nazarov, Rovshen, ja John Galletly. 2013. “Native browser support for 3D rendering and physics using WebGL, HTML5 and Javascript”. Teoksessa *BCI (Local),* 21. Viitattu 4. maaliskuuta 2015. http://ceur-ws.org/Vol-1036/p21-Nazarov.pdf.

Newzoo. 2015. “Global Games Market Will Reach $102.9 Billion in 2017”. Viitattu 19. maaliskuuta. http://www.newzoo.com/insights/global-games-market-will-reach-102-9-billion-2017-2/.

Ocariza, Frolin, Kartik Bajaj, Karthik Pattabiraman ja Ali Mesbah. 2013. “An Empirical Study of Client-Side JavaScript Bugs”. Teoksessa *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on,* 55–64. IEEE. Viitattu 14. toukokuuta 2015. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6681338.

Ohrt, Julian, ja Volker Turau. 2012. “Cross-Platform Development Tools for Smartphone Applications”. *Computer* 45 (9): 72–79.

Olsson, Carl Magnus. 2015. “2. Fundamentals for writing research”. Teoksessa *Game Research Methods,* 9–20. ETC Press. ISBN: 978-1-312-88473-8, viitattu 9. huhtikuuta 2015. http://press.etc.cmu.edu/content/game-research-methods-overview.

Orponen, Pekka. 1994. *Laskennan teoria.* Helsingin yliopisto, Tietojenkäsittelytieteen laitos.

Paulson, Linda Dailey. 2005. “Building Rich Web Applications with Ajax”. *Computer* 38 (10): 14–17. Viitattu 15. toukokuuta 2015. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1516047.

Phaser. 2015. Viitattu 13. huhtikuuta 2015. http://phaser.io/.

PhoneGap. 2015. Viitattu 13. huhtikuuta 2015. http://phonegap.com/about/.

Pixi.js. 2015. Viitattu 13. huhtikuuta 2015. http://www.pixijs.com/.

Puputti, Kimmo. 2012. “Mobile HTML5: Implementing a Responsive Cross-Platform Application”. Tutkielma, Aalto University.

Raivio, Miikka. 2013. “Alustariippumattoman mobiilisovelluskehityksen tekniikat”. Tutkielma, Tampereen teknillinen yliopisto.

Rastogi, Aseem, Nikhil Swamy, Cédric Fournet, Gavin Bierman ja Panagiotis Vekris. 2014. *Safe & efficient gradual typing for TypeScript.* Tekninen raportti. MSR-TR-2014-99. Viitattu 4. huhtikuuta 2015. `http://research.microsoft.com/pubs/224900/safets.pdf`.

Rodríguez, Rocío Andrea, Pablo M Vera, Federico Vallés ja María Roxana Martínez. 2014. “Analysis of Current and Future Web Standards for Reducing the Gap between Native and Web Applications”. Teoksessa *XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014).* Viitattu 14. toukokuuta 2015. `http://sedici.unlp.edu.ar/handle/10915/42347`.

Santelices, Raúl A, ja Miguel Nussbaum. 2001. “A framework for the development of videogames”. *Software: Practice and Experience* 31 (11): 1091–1107.

Singh, Bhawana, ja Anil Kumar Pandey. 2014. “Maze using Hybrid Genetic Algorithm”. Viitattu 19. huhtikuuta 2015. `http://ijari.org/CurrentIssue/ICARI2014/ICARI-CS-14-02-104.pdf`.

Sommer, Andreas, ja Stephan Krusche. 2013. “Evaluation of Cross-Platform Frameworks for Mobile Applications”. Teoksessa *Software Engineering (Workshops),* 363–376.

Souders, Steve. 2008. “High-performance web sites”. *Communications of the ACM* 51 (12): 36–41. Viitattu 2. toukokuuta 2015. `http://dl.acm.org/citation.cfm?id=1409374`.

Taivalsaari, Antero, Tommi Mikkonen, Matti Anttonen ja Arto Salminen. 2011. “The Death of Binary Software: End User Software Moves to the Web”. Teoksessa *Creating, Connecting and Collaborating through Computing (C5), 2011 Ninth International Conference on,* 17–23. IEEE. Viitattu 10. helmikuuta 2015. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5936687&tag=1`.

Taivalsaari, Antero, Tommi Mikkonen, Dan Ingalls ja Krzysztof Palacz. 2008. “Web browser as an application platform: The Lively Kernel experience”. Viitattu 28. maaliskuuta 2015. http://dl.acm.org/citation.cfm?id=1698208.

TexturePacker. 2015. Viitattu 13. huhtikuuta 2015. https://www.codeandweb.com/.

The Qt Company. 2015. Viitattu 17. huhtikuuta 2015. http://www.qt.io/developers/.

Thompson, Matt, A Imran Nordin ja Paul Cairns. 2012. “Effect of Touch-Screen Size on Game Immersion”. Teoksessa *Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers,* 280–285. British Computer Society. Viitattu 14. toukokuuta 2015. http://dl.acm.org/citation.cfm?id=2377952.

Three.js. 2015. Viitattu 13. huhtikuuta 2015. http://threejs.org/docs/.

Tiled. 2015. Viitattu 13. huhtikuuta 2015. http://www.mapeditor.org/.

Torrente, Javier, Ángel Serrano-Laguna, Ángel del Blanco Aguado, Pablo Moreno-Ger ja Baltasar Fernandez-Manjon. 2014. “Development of a Game Engine for Accessible Web-Based Games”. Teoksessa *Games and Learning Alliance,* 107–115. Springer. Viitattu 30. tammikuuta 2015. http://link.springer.com/chapter/10.1007/978-3-319-12157-4_9.

Turbulenz. 2015. Viitattu 13. huhtikuuta 2015. http://biz.turbulenz.com/.

Unity. 2015. Viitattu 13. huhtikuuta 2015. http://docs.unity3d.com.

Unreal Engine. 2015. Viitattu 13. huhtikuuta 2015. https://www.unrealengine.com/unreal-engine-4.

W3C. 2004. “Position Paper for the W3C Workshop”. Viitattu 20. maaliskuuta 2015. http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html.

———. 2014. “HTML5: A vocabulary and associated APIs for HTML and XHTML”. Viitattu 6. helmikuuta 2015. http://www.w3.org/TR/2014/REC-html5-20141028/.

W3Counter. 2015. “Web Browser Market Share, April 2015”. Viitattu 8. toukokuuta 2015. http://www.w3counter.com/globalstats.php.

Vanhatupa, Juha-Matti. 2010. “Browser Games for Online Communities”. *International Journal of Wireless & Mobile Networks (IJWMN), Vol. 2,* numero 3 (elokuu). Viitattu 6. helmikuuta 2015. http://airccse.org/journal/jwmn/0203ijwmn03.pdf.

———. 2011. “On the Development of Browser Games — Technologies of an Emerging Genre”. Teoksessa *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on,* 363–368. IEEE. Viitattu 2. maaliskuuta 2015. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6088206.

Weeks, Michael. 2014. “Creating a web-based, 2-D action game in JavaScript with HTML5”. Teoksessa *Proceedings of the 2014 ACM Southeast Regional Conference,* 7. ACM. Viitattu 30. tammikuuta 2015. http://dl.acm.org/citation.cfm?id=2638466.

Ylönen, Jani. 2014. “Videopelien historia ja pelinkehitys 2D-pelimoottoreiden vertailu”. Tutkielma, Jyväskylän yliopisto. Viitattu 2. maaliskuuta 2015. http://urn.fi/URN:NBN:fi:jyu-201501081057.

Yogya, Resa, ja Raymond Kosala. 2014. “Comparison of Physics Frameworks for WebGL-Based Game Engine”. Teoksessa *EPJ Web of Conferences,* 68:00035. EDP Sciences. Viitattu 30. tammikuuta 2015. http://www.epj-conferences.org/articles/epjconf/pdf/2014/05/epjconf_icas2013_00035.pdf.

Zhang, Yu. 2012. “Developing Effect of HTML5 Technology in web game”. *International Journal on Computational Sciences & Applications (IJCSA) Vo2,* numero 6. Viitattu 6. helmikuuta 2015. http://airccse.org/journal/ijcsa/papers/2612ijcsa03.pdf.

Liitteet

A Kyselyn vastaukset

Vastaaja 1.

Testaamiseen käytetyt laitteet ja niiden Internet-selaimet.

iPhone 6 (iOS8.3) / Safari
PC-kannettava / Chrome 42, Firefox 31 (ohjaus hiirellä)
Mac-kannettava / Chrome 42 (ohjaus koneen omalla touchpadilla)

Oliko pelien suorituskyvyissä havaittavia puutteita?

Ei, pelit toimivat sujuvasti kaikilla testilaitteilla.

Tuliko esiin ongelmia, jotka olisivat haitanneet/häirinneet pelaamista?

Perhepelissä pelaajan mukana seuraavat mangustit jäivät välillä hetkeksi melko kauas taakse (varsinkin mutkissa käännyttäessä), joskus jopa seinän taakse, ja ilmestyivät vasta hieman myöhemmin takaisin jonon jatkoksi. Varsinkin ensimmäisellä kerralla tuo katoaminen hieman hämäsi ja vei hetkeksi keskittymistä. Myös valokeilan hyppäykset eri kohtaan hieman häiritsivät aluksi. (sama kaikilla laitteilla)

Oliko testattujen laitteiden välillä eroja, jotka tekivät pelistä tai pelikokemuksesta erilaisen?

Terveyspelin pelikokemus oli melko samanlainen kaikilla laitteilla. Perhepelin osalta ohjaaminen oli selvästi sujuvinta iPhonella (kosketusnäyttö). Hiiren avulla pelattaessa ohjaaminen ei ollut yhtä sujuvaa, minkä vuoksi pelikokemus ei ollut niin hyvä kuin kosketusnäyttölaitteella. Hiirtä käyttäessä joutui ensin vähän aikaa kokeilemaan, miten hahmoa kannattaa liikuttaa (klikkailemalla, raahaamalla, tms.).

Millä laitteella oli miellyttävintä pelata ja miksi?

Terveyspelin kohdalla ei ollut suuria eroja sen suhteen, millä laitteella oli miellyttävintä pelata.Hahmon ohjaamisen intuitiivisuuden ja sujuvuuden puolesta perhepeliä oli miellyttävintä

pelata puhelimen kosketusnäytöllä, toiseksi miellyttävintä kannettavalla sen omaa touchpadia käyttäen ja vähiten miellyttävää kannettava + hiiri -yhdistelmällä.

Vastaaja 2.

Testaamiseen käytetyt laitteet ja niiden Internet-selaimet.

a) MacBook 2009 mid, OS X 10.10.3 (Safari 8.0.5), ohjaus näppäimistö/touchpad
b) iPhone 4S, iOS 8.3 (Safari)
c) Microsoft Surface, Windows RT 8.1 (Internet Explorer 11, Metro-versio)

Oliko pelien suorituskyvyissä havaittavia puutteita?

1. Perhe-peli, 2. Terveys-peli
1a) fps n.25
1b) fps n.20
1c) fps n.22
2a) fps n.35
2b) fps n.35
2c) fps n.35

Tuliko esiin ongelmia, jotka olisivat haitanneet/häirinneet pelaamista?

1. Perhe-peli, 2. Terveys-peli

mangustit tuntuivat välillä jumittuvan, mutta kohta aina kirivät ja löysivät reitin.
1a) Ei mainittavia ongelmia.
1b) Alun tekstilaatikko ei ihan kokonaan mahtunut ruudulle, kaksoikosketus ei toimi, reunaan koskeminen tuo selaimen menut esiin, jolloin pelin koko juuttuu, eikä palaa enää kokonäyttöön, jonka jälkeen selainnäkymä voi scrolalta hieman ja kadottaa pelinäkymästä osan. Näytöllä ahdasta, pelihahmo hukkuu ruudun reunoilla sormien alle. Jos pelastamisessa ei onnistu, niin tekstilaatikko painikkeineen ei sovi näyttöön. (Painikkeet voisivat olla ei väriset ja varustettu ikoneilla.) Ohjaus toimii miellyttävästi muuten mutta kun lähestyy reunaa niin näkymä hyppää, mutta sormi ei. Peli ihan mahdollista läpäistä, lähinnä pieni näyttö haittasi.
1c) Peli toimi hyvin kun näyttö oli isompi, epähuomiossa tehty pyyhkäisy(?) kerran kadotti

kontrollin kokonaan, ajan loputtua kyselyikkuna lopulta otti painikkeenpainalluksen vastaan tosin silti vaikka muu peli ei reagoinut. Sama pieni käytettävyysepämiellyttävyys on kun sormen joutuu asemoimaan uudestaan näytön hypähtäessä. Teknisesti muuten toimi.

2 Tavoite ei ihan selkeä, mutta helposti alkaa kokeilemalla selvittää. Kontrollit toimivat hyvin hiirellä ja kosketuksella. Palaute valinnoista selkeä. Musiikki loppui kesken, mutta toimi kaikilla testilaitteilla.
2b) ruudulla jos liian reunalta raahasi jotain, niin selain tulkitsi sen back-komennoksi

Oliko testattujen laitteiden välillä eroja, jotka tekivät pelistä tai pelikokemuksesta erilaisen?

1a/1b ohjauksen erot vaikuttivat hieman kokemukseen, 1b vähän hankalampi, mutta ei lopulta vaikuttanut.
2a/b/c, ainut ero pienempi näyttö 2b:ssä, teknisesti toimivat identtisesti.

Millä laitteella oli miellyttävintä pelata ja miksi?

Kokeilluista vaihtoehdoista a ja c olivat miellyttävimmät, perhepelissä a koska ohjaus oli näppäimistöllä (koska kosketusohjauksessa sormen uudelleenasemointi näkymän hypätessä) ja c terveyspeli kosketusnäytöllä. A ja c oli riittävän suuri näyttö verrattuna b:hen, jossa tapahtumat kyllä näki riittävän selvästi, mutta ohjauksessa käytetty sormi peitti näkymää.

Vastaaja 3.

Testaamiseen käytetyt laitteet ja niiden Internet-selaimet.

PC-pöytäkone (Win 7, Chrome)
PC-kannettava (Ubuntu 14, Chrome)
Android-tabletti (Firefox ja Chrome)

Oliko pelien suorituskyvyissä havaittavia puutteita?

Tabletilla perhe-pelin FPS melko pieni (n. 14), mutta haittasi pelaamista yllättävän vähän. Muilla ei ongelmia.

Tuliko esiin ongelmia, jotka olisivat haitanneet/häirinneet pelaamista?

Perhe-peli näytti android-firefoxilla pelkän mustan ruudun (plus aikalaskurin), eli pelaaminen ei onnistunut lainkaan. Chromella ongelmaa kuitenkaan ei ollut. Muilla alustoilla ei ongelmia.

Oliko testattujen laitteiden välillä eroja, jotka tekivät pelistä tai pelikokemuksesta erilaisen?

Aika vähän loppupeleissä. Kosketuskontrollit hieman toki muuttavat pelikokemusta, mutta näissä peleissä ei mielestäni merkittävästi.

Millä laitteella oli miellyttävintä pelata ja miksi?

Terveys-pelin raahailu ehkä kivempaa tabletilla, perhe-pelissä liikkuminen tuntuu parhaimmalta näppäimistöllä (täppäily kuitenkin hiirtä parempi ohjausmuoto).

Vastaaja 4.

Testaamiseen käytetyt laitteet ja niiden Internet-selaimet.

Lumia 735, windows 8.1 Internet Explorer (versio ?, viimeisin kummiskin)
MBP 13", OSX 10.8.5 (Chrome 42)

Oliko pelien suorituskyvyissä havaittavia puutteita?

Ei puutteita.

Lumia: perhe-pelissä fps taisesti n. 16 luokkaa. Toimi ihan sujuvasti.
Lumia: terveys-pelissä fps n. 40. Ei ongelmia
Chrome: perhe-pelissä fps n. 35
Chrome: terveys-pelissä fps n. 55

Tuliko esiin ongelmia, jotka olisivat haitanneet/häirinneet pelaamista?

Lumialla hahmo välillä jumahti paikalleen perhepelissä. Ihan kuin joku tietty suunta, johon hahmo oli liikkumassa jäisi päälle (hahmon "kävelyanimaatio"pyöri edelleen). Sen jälkeen peli ei enää reagoi uusiin kosketuksiin ja sivu piti ladata uudelleen.

Oliko testattujen laitteiden välillä eroja, jotka tekivät pelistä tai pelikokemuksesta erilaisen?

Suorituskyvyn kannalta ei huomattavaa eroa. Perhe-pelissä kännykällä tuntui koko ajan että sormi/käsi peittää liikaa peliä ja on tiellä.

Millä laitteella oli miellyttävintä pelata ja miksi?

Läppärillä pelaaminen tuntui jouhevimmalta, varsinkin perhe-peli kun pelasi käyttämällä näppäimistöä. Myös isompi näyttö teki pelaamisesta miellyttävämpää.

Muita huomioita?

Peliaika taitaa kulua perhe-pelissä jo kun alun info-ikkuna on auki.

Terveys-pelissä ainakin välillä vettä voi juottaa hahmolle rajattomasti.