OULUN AMMATTIKORKEAKOULU

Anton Orava

RPG-PELIEN RAKENTAMINEN HTML5- JA JAVASCRIPT-KIELILLÄ

RPG-PELIEN RAKENTAMINEN HTML5- JA JAVASCRIPT-KIELILLÄ

Anton Orava
Opinnäytetyö
Kevät 2017
Viestinnän koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Viestinnän koulutusohjelma, visuaalisen suunnittelun suuntautumisvaihtoehto

---

Tekijä: Anton Orava
Opinnäytetyön nimi: RPG-pelien rakentaminen HTML5- ja JavaScript-kielillä.
Työn ohjaaja: Tuukka Uusitalo
Työn valmistumislukukausi ja -vuosi: Kevät 2017 Sivumäärä: 48

Tutkielmani tavoitteena on luoda perusrakenne HTML5- ja JavaScript-ohjelmointikielillä rakennettavalle RPG-pelille ja mallintaa näiden kielten luovaa hyödyntämistä pelinkehityksessä. Sain idean tutkielmaani etsiessäni keinoja matalan kynnyksen pelinkehitykselle. Sen seurauksena valitsin verkossa käytettävät kielet ohjelmointikielekseni niiden helppouden ja monialustaisuuden vuoksi.

Tutkielmani tietoperusta koostuu materiaalista, jota kokosin lähdeaineistoani ja samanaikaisesti rakentamaani produktiota hyödyntäen. Rakentamani RPG-peli mahdollisti oppimieni uusien tekniikoiden jatkuvan hyödyntämisen ja soveltamisen. Tutkimukseni oli laadullinen, joten havainnointi oli siinä tärkeässä asemassa. Aineistoni pohjalta luotujen päätelmien ja analyysien avulla yhdistelin löytämiäni tekniikoita, jotka liittyivät tutkimuksessani esiin nousevien ongelmien ratkaisemiseen.

Tutkimukseni avulla näytin, kuinka HTML5 ja JavaScript soveltuvat RPG-pelien rakentamiseen ja erityisesti yksinkertaiseen ja luovaan koodin ja tekniikoiden uudelleenkäyttämiseen. Pelkkää koodia hyödyntäen esitin valmiita ratkaisu- ja ajatusmalleja, joita aloitteleva pelinkehittäjä tai muuten ohjelmointikielten käyttämän koodin parissa työskentelevä henkilö voi hyödyntää. Jätin tietoisesti tutkimuksestani pois pelimoottorit ja kirjastot juuri tästä syystä.

Tutkielmani aihe on ajankohtainen verkossa käytettävien ohjelmointikielten laajan levinneisyyden ja niistä ammennettavien monipuolisten mahdollisuuksien vuoksi. Tutkielmani avulla pelikehitykseen ja ohjelmointikielten opetteluun voidaan ottaa luonnollinen ensimmäinen askel, oli kyseessä sitten kokeneempi koodaaja tai vasta-alkaja.

Asiasanat: HTML, JavaScript, peliohjelmointi, sovellukset, pelit

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Communication, Option of Visual Communication

---

Author: Anton Orava
Title of thesis: Building a RPG-game with HTML5 and JavaScript.
Supervisor: Tuukka Uusitalo
Term and year when the thesis was submitted: Spring 2017 Number of pages: 48

This thesis is aimed to help create a template for HTML5 and JavaScript built RPG-game and also to show the creative use of these languages in game development. I had the idea for my thesis while I was searching light ways to create games and discovered the easy and multipurpose web languages.

The informational content of my thesis is assembled from the source material I used and from the production I built alongside this thesis. The RPG-game I created enabled me to apply and utilize the new techniques as I learned them. The research I did was qualitative so observation played a major role in it. The conclusions and analyses I made from the material I had, helped me to connect the found techniques in the process of problem solving.

In my research I proved how HTML5 and JavaScript can be useful in the development of RPG-games and in the creative and simple re-use of code and established techniques. With the use of clean code, I showed ready-made examples and models that a starting out game developer or coder can use. I deliberately left out game engines and frameworks for this exact reason.

The topic of my thesis is current due to the fact that web languages are wide spread and the possibilities of their use is great. This thesis can help people of varying skill levels to take the first step into the world of game design or coding in web languages.

Keywords: HTML, JavaScript, game development, applications, games

SISÄLLYS

TERMEJÄ

2D	2D eli kaksiulotteinen. Elementillä on kaksi ulottuvuutta (Two-Dimensional 2013, viitattu 8.12.2016).
HTML5	HTML5 on uusin standardi verkkosivujen tekemiseen käytettävästä HTML-merkintäkielestä (HTML Standard 2016, viitattu 8.12.2016).
Hybridisovellus	Hybridisovellus on sovellus, joka yhdistää natiivisovelluksen natiivikoodia ja HTML5-lähdekoodia (Riippi 2013, viitattu 8.12.2016).
JavaScript	JavaScript on ohjelmointikieli, joka määrittelee pelien toimintoja (van de Spuy 2012,1).
RPG-peli	RPG-peli eli Role-Playing Game on peligenre, joka yhdistää seikkailua ja strategiaa (Barton 2007, viitattu 8.12.2016).

1 JOHDANTO

Verkossa käytettävien ohjelmointikielten käytön helppous ja niiden tarjoamat valtavat mahdollisuudet tekevät pelien kehittämisestä jatkuvasti suositumpaa ja kehittyneempää. Toimivien ja rikkaiden monialustaisten hybridipelisovelluksien rakentaminen on helpompaa nyt kuin ikinä ennen – myös vasta-alkajille. Mobiilisovellusten luominen verkkotekniikoilla ja niiden julkaiseminen useaan eri sovelluskauppaan samanaikaisesti on tuotu kaikkien ulottuville. Mahdollisuuden ovat lähes rajattomat ja niiden puitteissa voi toimia luovasti omien kiinnostuksien mukaan.

Opinnäytetyöni aiheena on selvittää, miten HTML5- ja JavaScript-kieliä voidaan hyödyntää RPG-pelien rakentamisessa. RPG-peli eli Role-Playing Game on seikkailua ja strategiaa yhdistävä suosittu peligenre (Barton 2007, viitattu 8.12.2016). Aiheen valinta oli minulle helppoa, sillä olen ollut jo pitkään kiinnostunut HTML5-pelisovellusten kehittämisestä ja halusin laajentaa niin ammatillista kuin myös teknistä osaamistani tekemällä sellaisen täysin itse alusta asti. Halusin tehdä kaksiulotteisen RPG-pelin produktiossani, joten JavaScriptin hyödyntäminen HTML5:n ohella oli pakollista toimivan lopputuloksen aikaansaamiseksi. Valitsin aiheekseni RPG-pelin, sillä sen rakentaminen näillä kielillä on haastavaa, mutta monipuolisen pelikokemuksen mahdollistavaa.

HTML5 ja JavaScript ovat verkossa käytettäviä kieliä, joilla voidaan rakentaa verkkosivujen lisäksi toiminnoiltaan rikkaita ja monimutkaisia sovelluksia ja pelejä. Opinnäytetyöni tarkoitus on tutkia, miten näitä kieliä voidaan hyödyntää juuri RPG-pelin rakentamisessa. Tässä tutkielmassani tutkin 2D- eli kaksiulotteisen RPG-pelin rakentamista. Tavoitteenani on luoda kattava katsaus HTML5- ja JavaScript-käsitteiden eri osa-alueisiin ja mahdollisuuksin ja luoda opetus- ja tutkimusmateriaalia, jota alan henkilöt ja ammattilaiset voivat hyödyntää omissa projekteissaan. Kokoamaani materiaalia voidaan hyödyntää esimerkiksi monialustaisen hybridipelisovelluksen rakentamisessa. Hybridisovellus on natiivi- ja verkkosovelluksen välimuoto, joka rakennetaan verkkotekniikoilla, mutta voi myös hyödyntää laitteiden natiiviominaisuuksia (Riippi 2013, viitattu 8.12.2016).

Opinnäytetyöni on kvalitatiivinen eli laadullinen, joten tiedon analysointi ja havaintoihin perustuvat johtopäätökset ovat suuressa asemassa (Hirsjärvi ym. 2009, 224-225). Tutkimustyöni tiedonhakumenetelmään liittyy laajan lähdekirjallisuuden hankinta ja näiden tarjoaman tiedon perusteella tehdyt omat kokeilut ja esimerkkien soveltamiset. Opinnäytetyön alkuosa keskittyy käsitteiden avaa-

miseen ja näiden mahdollisuuksien kartoittamiseen. Jälkimmäinen puolisko on käytännönläheisempi ja perustuu opitun tiedon ja havainnointien hyödyntämiseen pelinrakentamisen kokonaisprosessissa. Opinnäytetyöni sisältää paljon englanninkielisiä termejä ja monimutkaisia käsitteitä, joten tietämys tai edes niihin liittyvä kiinnostus on lukijalle hyödyllistä. Lähdemateriaalini on pääosin englanninkielistä, joten käytän käsitteistä välillä erikseen suomennettuja versioita, alkuperäisen termin heti sen perässä suluissa mainiten.

Toisessa luvussa käyn läpi lyhyesti ja hyvin teknillispainotteisesti HTML5:n ja JavaScriptin keskeisiä käsitteitä ja tekniikoita, joita hyödynsin pelin rakentamisessa ja joihin viittaan myös myöhemmin. Erityisesti JavaScript on RPG-pelin rakentamisessa hyvin tärkeä työkalu, mutta se on aihealueena myös erittäin laaja. Tästä syystä läpi käymäni aiheet ovat ehdottoman tärkeitä ymmärtää jokaiselle, joka haluaa työskennellä JavaScriptin parissa pelinkehityksen näkökulmasta. Kolmannessa luvussa kerron tutkimusmenetelmästäni, sekä aineistostani. Neljäs luku on hieman pohdiskelevampi. Tässä kappaleessa käyn läpi asioita, joita kannattaa miettiä ennen varsinaista pelin rakentamisen aloittamista.

Tutkielmani luku 5 on omistettu RPG-pelin rakentamiselle. Pyrin luomaan lukijalle mahdollisimman kattavan käsityksen tavoista, joilla ”HTML5 ja JavaScript”-luvussa avattuja käsitteitä ja tekniikoita voidaan hyödyntää RPG-peleissä. Mallinnan ensin pelin sisäistä logiikkaa, jonka jälkeen kerron pelin hahmon luomiseen vaadittavia toimenpiteitä. Näihin kuuluvat muun muassa objektien, canvaksen ja animoinnin ymmärrys. Hahmon luomisen jälkeen rakennan pelille oman karttajärjestelmän, joka hyödyntää törmäyksen tunnistamista ja esineiden sijoittelua ilman tarkkojen pikseliarvojen määrittämistä. Kerron myös hieman, kuinka peliä voi kehittää tästä pisteestä eteenpäin opittuja tekniikoita hyödyntäen. Lopussa kokoan tutkielmani johtopäätökset yhteen ja pohdin tutkimuskysymykseni ratkaisuun ja suoriutumiseeni liittyviä asioita.

2 HTML5 JA JAVASCRIPT OHJELMOINTIKIELINÄ

HTML5 ja JavaScript ovat verkossa käytettäviä kieliä, joilla voidaan rakentaa verkkosivujen lisäksi toiminnoiltaan rikkaita ja monimutkaisia sovelluksia ja pelejä. Tässä luvussa avaan näiden kielten teknisiä ominaisuuksia ja termistöä, joihin viittaan myöhemmissä kappaleissa.

2.1 HTML5

HTML5 voi tarkoittaa kolmea eri asiaa asiayhteyden mukaan. Se voi olla HTLM-kielen uusi versio tai kehitysvaihe, yleisnimitys nykyaikaisille web-tekniikoille tai sovellusten toteuttamista webin tekniikoilla, joita ovat JavaScript, CSS ja HTML. (Korpela 2014, 3.) HTML5 on nykyaikaisten verkkosivujen pääasiallinen ”rakennusaines”, jota hyödyntämällä voidaan rakentaa mitä monimuotoisimpia sivustorakenteita. RPG-pelien luomisessa HTML5 on kuitenkin tärkeysasteeltaan toissijainen, sillä suurin osa pelin sisältävästä koodista on JavaScriptiä.

HTML-sivu on rakenteeltaan DOMin eli Document Object Modelin mukainen. DOM on rakenteeltaan puumainen ja muodostuu sen juurena olevasta Document-objektista ja tämän lapsisolmujen suhteista. Solmut eli nodet ovat DOMissa sisäkkäin olevia elementtejä, joilla on erilaisia suhteita toisiinsa. Solmu voi olla yläelementtinä toimiva isä (parent), isälle alisteinen lapsi (child) tai rinnakkainen sisarus (sibling). Elementit koostuvat yksinkertaisimmillaan alkutunnisteesta, sisällöstä ja lopputunnisteesta. Elementtien määritteet (attributes) kertovat elementin ominaisuuksista, esimerkiksi ” ” on kappale-elementti, joka alkaa alkutunnisteesta ”<>”, sisältää määritteen ”p” ja loppuu lopputunnisteeseen ”</>”. HTML-sivu muodostuu siis DOMin mukaisesti eri elementeistä, jotka muodostavat sivun rakenteen. (Korpela 2014, 47–48.)

Osa HTML5:n määritteistä on määritelty globaaleiksi (global attributes), jolloin niitä voi käyttää kaikissa elementeissä. Tapahtumat (events) ovat globaaleja JavaScriptissä hyödynnettäviä toimintoja. Esimerkiksi ladattavaan kuvaan voidaan liittää niin sanottu tapahtumakäsittelijän toiminto (addEventListener), jonka avulla koodin lukeminen voidaan pysäyttää kuvan lataamisen ajaksi. Tapahtumamääritteet alkavat ”on”-sanalla, kun taas id-määriteet liittävät elementtiin tunnisteen, jota voidaan kutsua elementteihin viittaavissa linkeissä, CSS:ssä ja JavaScriptissä. Esimerkkinä tapahtumamääritteestä voidaan pitää ”onclick”-määritettä, joka viittaa hiirellä klikkaamiseen. Id-tunniste

voi olla käytännössä mitä vain, kunhan se ei sisällä välilyöntejä ja se on uniikki. Elementteihin voidaan viitata ”querySelector()”-metodilla, jonka argumenttiin eli sulkujen sisäpuolelle voidaan kirjoittaa halutun elementin id-tunniste, class-tunniste tai pelkkä elementin määrite, kuten ”nav” tai ”p”. (Korpela 2014, 51, 141, 165–166.) Kuviossa 1 on havainnollistettu ”div”-elementin id-määrite ”esimerkki” sekä sen tapahtumamäärite ”onclick”. Div-elementin klikkaamisen jälkeen ”onclick” luo ponnahdusikkunan, jonka tekstiosuudessa lukee: ”Esimerkkiteksti”.

```
<!doctype html>
<body>
<div id="esimerkki" onclick="alert('Esimerkkiteksti')"></div>
</body>
```

KUVIO 1. Määritteiden esimerkkejä.

2.2 JavaScript

JavaScript on selainskriptien eli lyhemmin skriptien tekemiseen käytettävä kieli. Skripti on ohjelmakoodia, jota selain suorittaa selainohjelmionnissa (client side scripting) eli HTML-sivujen näyttämisen yhteydessä. (Korpela 2014, 55.) JavaScriptiä voidaan kirjoittaa suoraan HTML-dokumenttiin script-tunnisteilla (<script></script>), tai se voidaan liittää HTML-tiedostoon ulkoisesti dokumentin ylätunnisteeseen eli ”head”-osioon tai sen alapuolella sijaitsevaan ”body”-osioon “src”-määrittelyn avulla (<script src="tiedostonimi.js"></script>) (van de Spuy 2012, 62,106).

Kaksi kenoviivaa JavaScript-tiedostossa merkitsevät kommenttia, jolloin kyseisen rivin sisältöä kenoviivojen jälkeen ei huomioida koodissa. Kommentin voi myös laittaa merkkien ”/**” ja ”**/” väliin, jolloin kommentti voi olla yhtä riviä pidempi. (van de Spuy 2012, 67.) Kommentti on havainnollistettu kuviossa 2.

2.2.1 JavaScript funktiot, muuttujat ja objektit

JavaScriptin tärkeimpiä työkaluja ovat muuttujat (variables) ja funktiot (functions). Muuttujaa käytetään JavaScriptissä varastoimaan tietoa ja se määritetään tavalla: var esimerkki = ”muuttujanarvo”, jolloin muuttujan nimeen ”esimerkki” voidaan viitata myöhemmin koodissa. Funktiot sisäl-

tävät toimintoja ja niitä voidaan kutsua niiden määrittelyn jälkeen (funktio esimFunktio() { funktion-koodi }) pelkällä funktion nimellä ja sulkeilla (esimFunktio()). (van de Spuy 2012, 65, 80.) Kuviossa 2 on esitetty yksinkertainen muuttujan ja funktion määrittely sekä niiden kutsuminen.

```
var esimMuuttuja = 3; //Muuttujan määrittely arvoon 3
console.log(esimMuuttuja); //Muuttujaa voidaan nyt kutsua

function esimFunktio() { //Funktion määrittely
   console.log("Esimerkkiteksti");
}
esimFunktio(); //Funktio voidaan nyt kutsua
```

KUVIO 2. Muuttujan ja funktion määrittäminen sekä niiden kutsuminen. Lisänä kommentteja.

JavaScriptissä objekti on kokoelma nimettyjä ominaisuuksia (properties), jotka voidaan kirjoittaa aaltosulkujen sisään. Objektin määrittelyn jälkeen (var muuttuja = { ominaisuus: arvo, ominaisuus2: arvo }), siihen voidaan viitata pistemerkinnällä (muuttuja.uusiOminaisuus) tai hakasulkumerkinnällä (muuttuja['uusiOminaisuus']). Objektille voi myös lisätä arvoja sen muuttujan määrittelyn jälkeen: muuttuja.uusiOminaisuus = 'yksi' lisää objektiin ominaisuuden "uusiOminaisuus", jonka arvo on "yksi". (Korpela 2014, 59.) Kuvio 3.

```
//Objekti luodaan
var muuttuja = {
   ominaisuus: arvo,
   ominaisuus2: arvo
}

//Objektille annetaan uusi ominaisuus, jonka arvo on "yksi"
muuttuja.uusiOminaisuus = "yksi";
```

KUVIO 3. Objektin määrittely ja ominaisuuden lisääminen jo olemassa olevaan objektiin.

Objekteja voi myös monistaa aiemmin luodun mallin pohjalta loputtomasti "Object.create":n avulla. Yksinkertaisimmillaan uuden muuttujan perään merkitään Object.create, jonka sulkujen sisäpuolelle kirjoitetaan monistettavan objektin nimi. Uusi objekti saa monistettavan objektin ominaisuudet ja arvot. On tärkeää muistaa, että alkuperäisen objektin ominaisuuksien ja arvojen muuttaminen heijastuu myös siitä tehtyihin kopioihin. Kopioiden ominaisuuksien ja arvojen muuttaminen ei heijastu muihin objekteihin. (van de Spuy 2012, 310.) Kuvio 4.

```
//Muuttujasta tehdään objekti toisen objektin avulla
var uusiMuuttuja = Object.create(alkuperäinenObjekti);
```

KUVIO 4. Objektin monistaminen.

2.2.2 JavaScript taulukot ja silmukat

Pelien rakentamisen kannalta yksi JavaScriptin hyödyllisimmistä ominaisuuksista ovat taulukot (arrays). Muuttujat kykenevät varastoimaan vain yhden kappaleen tietoa kerrallaan, kun taas taulukoihin tietoa voidaan varastoida loputtomat määrät. Taulukot pitää muuttujien tavalla määritellä koodissa, ennen kuin niitä voidaan käyttää. Taulukoiden alkioihin, eli sen varastoimiin tiedonkappaleisiin, päästään käsiksi viittaamalla niiden indeksinumeroihin (index numbers). Jokaisella alkiolla on oma indeksinumeronsa sen mukaan, monesko kappale tietoa se on taulukossa. Numerointi alkaa nollasta. (van de Spuy 2012, 185.) Kuviossa 5 on esitetty taulukon määrittäminen ja sen alkioon viittaaminen.

```
var taulukko = ["alkio1", "alkio2"]; //Taulukon määrittäminen
taulukko[0]; /Alkioon viittaaminen. Tässä tapauksessa viitataan alkio1:een
```

KUVIO 5. Taulukon määrittäminen ja sen alkioon viittaaminen.

Taulukon alkioita voidaan käsitellä usealla tavalla. Taulukkoon voidaan liittää uusi alkio joko sijoittamalla se tiettyyn indeksinumeroon (taulukko[haluttu indeksinumero] = ”uusiAlkio”) tai työntämällä (push) se viimeiselle paikalle taulukon lopussa (taulukko.push(uusiAlkio)). Taulukosta voidaan poistaa alkioita joko ”pop”-metodilla kajoten taulukon viimeiseen alkioon (taulukko.pop();) tai “splice”-metodilla taulukon tiettyyn indeksinumeroon viitaten (taulukko.splice(”indeksinumero”, ”alkioiden poistettava määrä”)). (van de Spuy 2012, 187–189.) Taulukon pituus eli sen alkioiden määrä voidaan saada selville hyödyntämällä sen ominaisuutta ”length” (taulukko.length) (Korpela 2014, 59). Taulukoiden pituutta hyödynnetään esimerkiksi yhdessä silmukan kanssa.

Silmukka (loop) on etenkin taulukoiden kanssa käytettynä tehokas työväline. ”For-silmukan” avulla taulukon kaikki alkiot voidaan selata lävitse, jolloin niistä voidaan esimerkiksi etsiä tiettyä alkion arvoa tai halutulle alkiolle voidaan tehdä koodissa määritelty asia. (van de Spuy 2012, 192–193.)

Kuviossa 6 on esitetty yksinkertainen for-silmukka, joka käy lävitse taulukon kaikki alkiot ja kertoo niiden arvot.

```
for(var i = 0; i<taulukko.length; i++){
  console.log(taulukko[i]);
}
```

KUVIO 6. Taulukko silmukan kanssa.

2.2.3 If- ja switch-lausekkeet

If-lausekkeen (if statement) ja switch-lausekkeen (switch statement) avulla koodiin voidaan asettaa tiettyjä ehtoja asioiden tapahtumiselle. If-lauseke voi esimerkiksi olla muotoa: ”jos muuttujan arvo on neljä, tulosta muuttujan arvo”. Else if -lausekkeita voidaan liittää if-lausekkeen perään toteuttamaan vaihtoehtoisia ehtoja samalle muuttujalle. Else-lauseke voidaan liittää yksittäisenä if-lausekkeen loppuun suorittamaan haluttu toiminto muuttujan arvon ollessa eri kuin annettujen ehtojen vastainen. Kuviossa 7 on esitetty yksinkertainen if-lauseke. (van de Spuy 2012, 71–74.)

```
var muuttuja = 3; //Muuttuja määritellään

//If-lauseke tarkastaa muuttujan arvon ja suorittaa tässä tapauksessa else-lausekkeen
if (muuttuja === 4) {
  console.log("Muuttujan arvo on neljä"); //Koodi suoritetaan, jos muuttujan arvo on neljä
}
else if (muuttuja === 5) {
    console.log("Muuttujan arvo on viisi"); //Koodi suoritetaan, jos muuttujan arvo on viisi
}
else {
  console.log("Muuttujan arvo ei ole neljä eikä viisi"); //Koodi suoritetaan muussa tapauksessa
}
```

KUVIO 7. If-lauseke

Switch-lauseke on ikään kuin monimutkaisempi if-lauseke. Sen avulla koodiin voidaan kirjoittaa ehtoja, jos muuttujan arvo on jokin tietty. Switch-lausekkeeseen kirjoitetaan ”case” jokaisen muuttujan arvon kohdalle ja sen alapuolelle määritellään suoritettava koodi. Jokaisen case päätetään kirjoittamalla ”break;”, jolloin koodi ei lue ehtoa enää pidemmälle. Lausekkeen loppuun voidaan kirjoittaa myös ”default”-koodi, joka suoritetaan muuttujan arvon ollessa jotakin muuta kuin mitä

lausekkeen ehtoihin on määritelty. (van de Spuy 2012, 75–77.) Kuviossa 8 on esitetty yksinkertainen switch-lauseke.

```
var muuttuja = "arvo1"; //Muuttuja määritellään

switch (muuttuja) { //Haluttu muuttuja nimetään sulkeissa
  case "arvo1":
    console.log("Muuttujan arvo on arvo1"); //Muuttujan arvo on “arvo1” eli tämä koodi suoritetaan

  case "arvo2":
    console.log("Muuttujan arvo on arvo1");

  default:
    console.log("Muuttujan arvo on jotakin muuta");
}
```

KUVIO 8. Swich-lauseke.

2.2.4 DOM-puun käsittely JavaScriptillä

HTML-dokumentin DOM-puun osiin, eli käytännössä sen elementteihin, voi viitata Document-olion ominaisuuksien avulla. Document-objektista voidaan listata ”document.getElementsByName()”-metodin avulla kaikki elementit, joiden nimi vastaa metodin argumenttia. Argumentti sijoitetaan sulkeiden sisään. (Korpela 2014, 151–153.) Document-objekti alkaa aina ”<!doctype html>”-tunnisteella, jotta tietokone ymmärtää dokumentin olevan HTML5-dokumentti (van de Spuy 2012, 7).

JavaScriptillä voidaan käsitellä sivun elementtejä esimerkiksi funktiolla ”document.getElementById,” jonka argumentiksi sijoitetaan halutun elementin id-ominaisuuden arvo. Elementtien ominaisuuksien arvoja voidaan myös muuttaa tällä tavalla. HTML:n elementeillä on metodeina eli elementteihin liittyvinä funktioina valmiiksi määriteltyjä JavaScript-funktioita, kuten edellä mainittu ”document.getElementById”. JavaScriptillä voidaan myös luoda elementtejä dokumenttiin DOM-rakenteena funktiolla ”document.createElement”. (Korpela 2014, 59, 65–66.)

3 TUTKIMUSMENETELMÄ JA AINEISTO

Tietoperustan hankkimisen lisäksi rakensin opinnäytetyöni produktio-osassa HTML5:llä ja JavaScriptillä RPG-pelin alusta saakka löytämääni tietoa hyödyntäen. Pelin rakentaminen vaati soveltavaa tutkimustyötä ja aineiston hankintaa, joita avaan hieman seuraavaksi. RPG-pelien luominen näitä verkkotekniikoita hyödyntäen on aihealueena hyvin tarkka, eikä tällaiseen projektiin ole suoranaisesti minkäänlaista valmista lähdeaineistoa.

3.1 Tutkimusmenetelmän valinta

Tutkimuksessani selvitän, miten HTML5- ja JavaScript-kieliä voidaan hyödyntää RPG-pelien rakentamisessa. Tutkimuskysymykseeni vastauksen löytäminen vaati lähdeaineistoon syvällisen perehtymisen lisäksi luovaa ongelmanratkaisukykyä. Käytännössä opettelin itselleni ensi kokonaan uuden kielen, jonka jälkeen vasta tekemieni muistiinpanojen avulla kykenin perehtymään varsinaisiin, pelin kannalta tärkeisiin ennalta määrättyihin tavoitteisiin. Tutkimukseni on kvalitatiivinen eli laadullinen, joten tiedon analysointi ja havaintoihin perustuvat johtopäätökset ovat suuressa asemassa (Hirsjärvi ym. 2009, 266).

Pyrin ratkaisemaan tutkimustehtävänä olevan ongelmani hankkimalla ajanmukaista ja luotettavaa tietoa tutkittavasta aiheesta. Tällaista tietoa löytyy vain vähän ja sitä joutuu käyttämään luovasti. Lähdetietoni perustuu muutaman teknillispainotteisen kirjan lisäksi englanninkielisistä e-kirjoista ja muista digitaalisista lähteistä kerättyyn tietoon. Lähteistä saadun tiedon rajaaminen perustuu tutkimuskysymykseni ratkaisemiseen liittyviin olennaisiin tekniikoihin. Rajaan tietoa siis sen pohjalta, mikä osa siitä edistää ongelmani ratkaisua. Luovuus ongelmani ratkaisuun tulee siitä, että joudun soveltamaan eri lähteistä löydettyä tietoa käytäntöön produktiossani ilman täydellistä varmuutta lopputuloksesta.

Kirjallisen osuuden rinnalla rakennettava produktio tukee siis oppimisprosessiani esimerkiksi havainnoinnin näkökulmasta ja näin ollen se tukee kirjallista osaa osana aineistoa. Laadullisessa tutkimuksessa täsmentyneestä havaintomateriaalista voidaan päätyä selitysmalleihin ja teoreettiseen pohdiskeluun (Hirsjärvi ym. 2009, 266). Kirjasin havaintojani ylös hyödyntäen jatkuvasti karttuvaa tietoa, jota saan käytännön kokemuksesta ja teorioiden testaamisesta.

3.2 Tutkimuksen aineisto

Tutkimukseni aineisto sisältää materiaalia jota keräsin lähdetiedosta ja niiden sovellutuksista tekemieni johtopäätöksien ja havainnointien pohjalta. Analysoin aineistoani ymmärtämiseen pyrkivällä lähestymistavalla, jossa käytetään laadullista analyysia ja päätelmien tekoa (Hirsjärvi ym. 2009, 224–225). Analysoin keräämääni tiedon ja omien kokeilujeni pohjalta tehtyjä tuloksia jakamalla aineiston moniin eri osa-alueisiin. Osa-alueet pitävät sisällään RPG-pelien rakentamiseen vaadittavia perustavanlaatuisia rakennuspalikoita. Näitä olivat pelin logiikan kannalta tärkeät osat, canvas, kartat, erilaiset toiminnot, joita pelissä tarvitaan interaktiivisuuden takaamiseksi, sekä tietysti hahmot ja esineet. Näihin kategorioihin päädyin rakennettuani itse ensin toimivan RPG-pelin rungon. Kategoriat kattavat osin samoja tai saman tyyppisiä tekniikoita, mutta JavaScript-koodin kannalta ne käydään läpi merkityksellisessä järjestyksessä.

Pyrin analyysissäni luomaan lukijalle eräänlaisen perusrakenteen, jota voi jatkossa hyödyntää myös muunlaisiin käyttötarkoituksiin. Tutkimuskysymykseni vastaus on laaja ja monisyinen, joten sen jakamineen useaan osa-alueeseen myös jäsentää sen ymmärtämistä. Tulkinta tutkielmassani on analyysissa esiin nousevien merkitysten selkiyttämistä ja pohdintaa (Hirsjärvi ym. 2009, 229). Lopun pohdinta-osiossa kerään yhteen tekemäni johtopäätökset kattavaksi kokonaisuudeksi.

Keräämästäni teknillispainotteisesta aineistostani kykenin luomaan ajatusmalleja ja valmiita teknisiä rakenteita, joita käytin hyväkseni produktioni eri osissa. Toisistaan riippumattomat osat löysivät yhteisiä suhteita ja näin ollen loivat myös sisältöä aineistooni. Ohjelmointikielillä suoritettava koodin kirjoitus ei ole täysin yksiselitteistä eikä siihen ole aina yhtä oikeaa tapaa. Pyrinkin analyysissäni tuomaan esiin myös tapoja, jotka itse koin hyviksi tai tehokkaiksi. On kuitenkin hyvä muistaa, että jokaisella koodin parissa työskentelevällä on erilaisia työskentelytapoja ja käytäntöjä ongelmien ratkaisemiseen.

4 PELIN SUUNNITTELU JA TAVOITTEET

Ennen laajamittaista pelikoodin rakennusprosessia on hyvä tietää, mitä peliltään halutaan ja minkälaisia tavoitteita sille asetetaan. HTML5- ja JavaScript-tekniikoita hyödyntävien pelien luomiseen on olemassa jonkin verran erinäisiä ohjeita, joten saatavilla olevaa tieto kannattaa hyödyntää. Jotta oikeanlaista tietoa osaa etsiä, on hyvä pohtia seuraavanlaisia yksityiskohtia.

4.1 Pelin tyyppi

Verkkotekniikoita hyödyntämällä voidaan rakentaa usean eri tyypin pelejä. Muutamia eri tyyppejä ovat interaktiivinen fiktio, puzzlepelit, tasohyppely, taistelu, FPS eli ensimmäisessä persoonassa kuvattava ammunta ja RPG eli Role Playing Game (Burchard 2013, Contents). Edellä mainitut tyypit voivat olla myös kaksi- tai kolmiulotteisia. Tutkielmassani perehdyn kaksiulotteisen RPG-pelin rakentamiseen, mutta pelityyppiä valittaessa on hyvä pitää kaikki vaihtoehdot avoinna. Monet pelityypit sisältävät paljon saman tyyppisiä koodinpätkiä ja tekniikoita, joten yhden pelityypin opiskelu hyödyttää myös jatkossa muunlaisia pelejä rakentaessa.

4.2 Laiteympäristö

Peliä luodessa kannattaa miettiä, mille laitteille se on suunnattu. Verkkotekniikoiden suuri etu on kuitenkin se, että ne toimivat käytännössä kaikilla moderneilla sähköisillä päätelaitteilla, jotka tukevat HTML5:n ja JavaScriptin esittämistä. HTML5:n kanssa kannattaa hyödyntää responsiivista suunnitteluperiaatetta. Responsiivisessa suunnittelussa CSS-, JavaScript- ja HTML- kieliä hyödynnetään sisällön uudelleen järjestelyssä, elementtien koon muuttamisessa ja esimerkiksi piilottamisessa ikkuna- tai laitekoon muuttuessa (W3 Schools 2017, viitattu 17.02.2017). Yksi koodi voi siis soveltua usealle eri laitteelle, jos sen muistaa ja osaa optimoida eri näyttökoot mielessä pitäen.

4.2.1 Pelimoottorit ja kirjastot

Pelimoottorit (game engines) ovat väliohjelmistoja, jotka pitävät jo valmiina sisällään ohjelmoituna pelien perustoiminnallisuuksia ja rakenteita. Pelimoottoreita hyödyntämällä pelin asennuspaketin koosta saadaan myös mahdollisimman pieni. Pieni koko on tärkeää erityisesti mobiilipeleille, niille

asetettujen kokorajoitteiden takia muun muassa sovelluskaupoissa. (Tukiainen 2013, 26–28.) Peliä suunniteltaessa kannattaa siis aina pitää mielessä sen kohdelaite, jolle käyttäjä on sen tarkoitus ladata.

JavaScript-kirjastot (frameworks) on luotu helpottamaan pelin kehitystyötä ja toimimista pelikoodin parissa. Ne esimerkiksi yksinkertaistavat ja nopeuttavat funktioiden toteuttamista. Kirjastot eroavat toisistaan ja voivat hyödyntää toisistaan poikkeavia ratkaisumalleja samoihin ongelmiin. (Lappalainen 2012, 11.)

Omaa RPG-peliä rakentaessani en käyttänyt hyväksi pelimoottoreita tai JavaScript-kirjastoja, sillä halusin oppia mahdollisimman paljon manuaalisesta koodin kirjoittamisesta. Nämä ovat kuitenkin hyödyllisiä työkaluja ja voivat helpottaa pelin rakentamista sekä tuoda siihen erilaisia hyödyllisiä lisäyksiä.

4.2.2 Hybridisovellus

Produktiossani rakennettu peli on teknisesti ilmaistuna niin sanottu hybridisovellus. Hybridisovellus yhdistää HTML5-lähdekoodia ja sille tarkoitetun laitteen natiivikoodia. Käytännössä hybridisovellus voidaan siis luoda ainakin osittain HTML5-tekniikalla, joka pakataan sovelluksen sisälle eri työkaluilla. (Riippi 2013, viitattu 13.12.2016.) Tämä on yksi tekniikka muiden joukossa, mutta pelin julkaisemista silmällä pitäen päädyin tähän ratkaisuun. Peli voidaan siis luoda täysin toimivaksi kokonaisuudeksi verkkotekniikoita hyödyntäen, mutta se pakataan eri laitteille sopivaksi sovellukseksi ulkoisella palvelulla.

4.3 RPG-pelille asetettavat tavoitteet

Tutkimukseni tavoitteena on luoda perusteet RPG-pelin rakentamiselle verkkotekniikoita hyödyntäen. Ennen varsinaisen pelin rakentamisen kuvausta seuraavassa luvussa, haluan tehdä selväksi, mitä tämä käytännössä sisältää. Peliin tulee luoda tarvittavat HTML- ja JavaScript-tiedostot, joihin rakennetaan keskenään yhteen pelaava, toimiva sisäinen logiikka. Pelissä tulee olla yksi tai useampi hahmo. Hahmoa tulee pystyä liikuttamaan tietokoneen näppäimistön nuolinäppäimillä niin, että hahmon liikkuessa, tämän liike on animoitu sulavasti. Pelissä voi olla myös esineitä ja esteitä,

jotka toimivat teknisestä näkökulmasta katsottuna samankaltaisesti hahmon kanssa. Kaikki tämä vaatii objektin ja canvaksen käsitteiden ymmärtämistä ja joustavaa käyttöä.

Hahmo- ja esineobjektien luomisen jälkeen pelissä pitää olla jonkinlainen karttajärjestelmä, eli tausta- ja etualakuvat, sekä törmäys- ja objektikartat. Hahmon tulee myös pystyä liikkumaan sille luodussa maailmassa RPG-pelien seikkailulliseen tapaan. Karttojen sekä hahmojen ja esineiden välille saadaan vuorovaikutusta törmäystä ja muita tekniikoita hyödyntämällä. Törmäyskartta estää hahmon liikkumisen sille kiellettyihin paikkoihin, jolloin illuusio pelimaailmasta ei rikkoudu. Objektikartta on kartta, jonka avulla peliin sijoitetaan erinäisiä asioita, kuten esineitä ja erikoisruutuja. RPG-pelissä hahmon pitää pystyä keräämään pisteitä tai esineitä sekä liikkumaan karttojen välillä, joten muuttujien, taulukoiden ja objektien arvoja pitää pystyä muuttamaan vuorovaikutuksen seurauksena.

Tavoitteenani on pystyä esittämään kaikki tämän mahdollisimman ymmärrettävästi tavalla, jolloin jokainen verkkotekniikoilla omaa peliään rakentava henkilö pystyy hyödyntämään tietoa omissa projekteissaan. Tavoitteenani ei ole luoda täydellistä tai valmista peliä, vaan näyttää, kuinka kerran opittuja tekniikoita voi hieman luovuutta käyttäen kierrättää aina uudelleen. Pelin tarinan suunnittelu sekä monet muut asiat, kuten pelin genren valinta voivat vaikuttaa pelissä oleviin piirteisiin ja toimintojen monimutkaisuuteen. En käy tutkielmassani kuitenkaan läpi pelin tarinallista tai käsikirjoituksellista puolta, sillä haluan keskittyä täysin sen teknisen puolen rakentamiseen.

5 RPG-PELIN RAKENTAMINEN

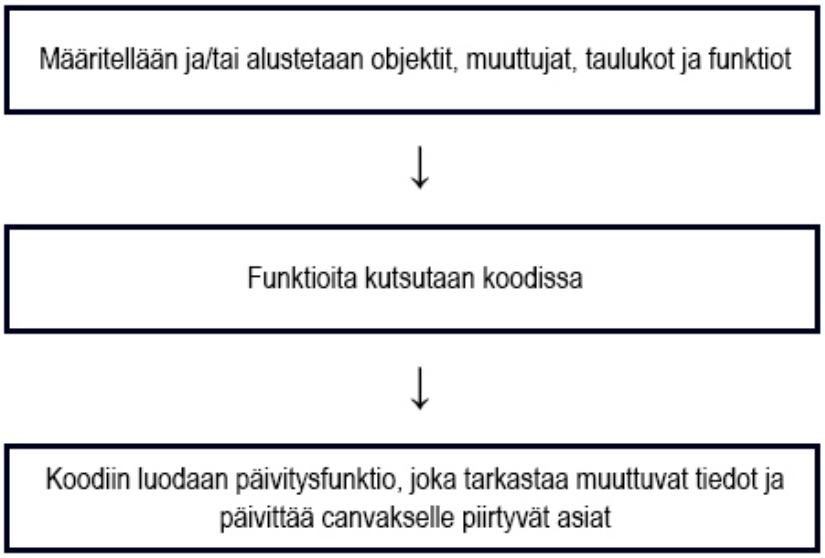
Yksinkertaisen RPG-pelin koodi voidaan jakaa muutamaan alakohtaan. Koodiin määritetään ensin pelissä esiintyvät objektit, alustetaan tai määritetään tarvittavat muuttujat ja taulukot sekä määritetään tarvittavat funktiot. Funktioita kutsutaan niiden määrittämisen jälkeen ja koodiin luodaan eräänlainen päivitysfunktio, joka päivittää pelin canvakselle piirtyvät asiat. Funktioita ja muuttujia voidaan liittää erillisillä JavaScript-tiedostoilla tai ne voivat kaikki sisältyä yhteen laajaan tiedostoon. Tässä luvussa avaan RPG-pelien luomisen kannalta tärkeimpiä teknisiä seikkoja, ja havaitsemiani huomionarvoisia tekniikoita. Käytän esimerkeissä hyödykseni tekemääni ”Musta Morsian” -nimistä RPG-peliä.

5.1 Pelin sisäinen logiikka

Peliä varten luodaan aluksi HTML-tiedosto, johon tarvittavat CSS- ja JavaScript-tiedostot linkitetään. HTML-tiedoston koodi aloitetaan doctype-tunnisteella, jonka jälkeen dokumentin head-osioon voidaan kirjoittaa haluttuja määrityksiä. HTML-tiedostoon voidaan luoda myös esimerkiksi pelinäkymässä esillä olevia div-elementtejä, kuten pelaajan esinevarasto tai vaikkapa erilaisia asetuksia sisältävä yläpalkki. HTML-tiedostoon luodaan myös canvas-elementti, johon muun muassa pelin objektit piirtyvät. JavaScript-tiedostot linkitetään viimeiseksi tiedoston loppuun. Canvas on HTML-elementti, joka luo tiedostoon tyhjän piirtoalueen. Canvas-elementtiin perehdyn tarkemmin hieman myöhemmin.

Funktiot ovat pelin toiminnallisuuden kannalta tärkein yksittäinen seikka, sillä ne sisältävät käytännössä kaiken pelissä olevan vuorovaikutteisuuden. On tärkeää muistaa, että funktiot tulee muuttujien tavoin määritellä ennen niiden kutsumista. Sos JavaScript-koodin siis haluaa hajauttaa useampaan tiedostoon, tiedostot tulee liittää HTML-tiedoston perään sellaisessa järjestyksessä, että koodissa ei tule viittauksia määrittelemättömiin funktioihin. Käytännössä tämä tarkoittaa sitä, että jos esimerkiksi pelissä olevat kartat hajautetaan omaan JavaScript-tiedostoon, tulee tiedosto liittää HTML-tiedoston koodiin ennen karttoihin viittaavaa pääasiallista JavaScript-koodia.

Pelin koodin pitää sisältää jokin jatkuvasti päivittyvä funktio, jolloin koodi tarkastaa ja päivittää ruudulla olevat toiminnot ja elementit reaaliajassa. Tämä saavutetaan käyttämällä ”requestAnimationFrame”-metodia. Kun lisätään metodin sulkeisiin päivittyvä funktio ja halutun canvaksen nimi (requestAnimationFrame(funktionNimi, canvaksenNimi);), metodi päivittää funktion hyödyntämällä näytön omaa päivitysnopeutta (van de Spuy 2012, 379). Päivitettävän funktion sisälle on hyvä laittaa ehdot ja koodit, jotka liittyvät esimerkiksi hahmon liikkeisiin, pelialueen mahdolliseen muutokseen ja pelin kameraan, sekä törmäyksen tunnistamiseen. Kuviossa 9 on esitetty yksinkertaistettu pelin logiikka.



KUVIO 9. Pelin sisäinen logiikka yksinkertaistettuna.

Ennen päivityksen aloittavan funktion kutsumista objekteihin (kuten hahmoihin) viittaavat kuvat ladataan koodissa etukäteen erinäisten ongelmien välttämiseksi. Pelin jokainen hahmo, irrallinen esine tai kerättävä asia on oma objektinsa. Jokaisen kartan jokainen solu, joka sisältää jonkin erikoispiirteen, on myös oma objektinsa. Jokainen objekti joka halutaan piirtää canvakselle, eli käytännössä peliruudulle, tarvitsee myös kuvatiedostoa säilyttävän muuttujansa.

Objektien määrittämisen jälkeen objektille luodaan rinnakkainen kuvamuuttuja, jolle annetaan arvoksi JavaScriptin ”new Image()”-lauseke. new Image() luo tiedostoon uuden näkymättömän ”<img>”-tunnisteen, johon voidaan viitata uuden muuttujan avulla. Tämän jälkeen muuttujaan liitetään tapahtumakäsittelijän toiminto (addEventListener), jonka argumenttiin sijoitetaan "load", ”loadHandler” ja ”false” toisistaan pilkuilla erotettuna. loadHandler on funktio, joka määritetään koodissa.

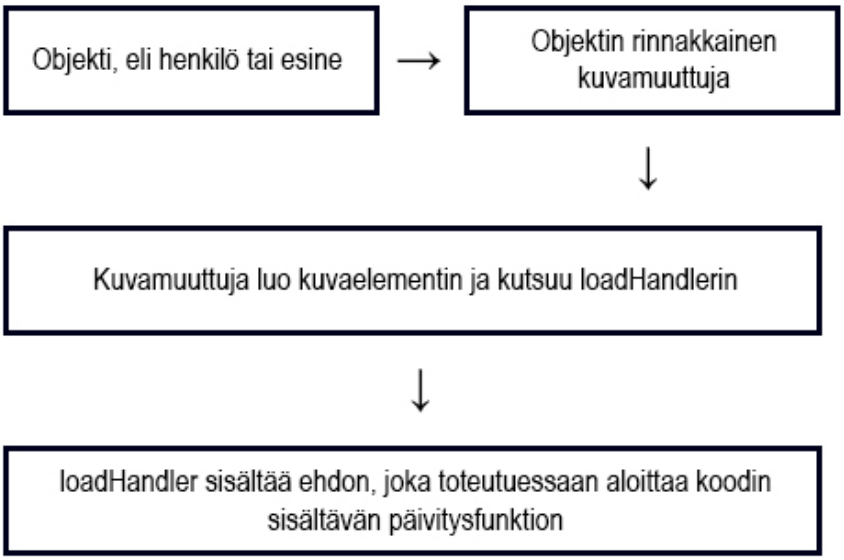
Sen tehtävä on laskea kaikki koodissa etukäteen ladattavat muuttujat, varmistaa, että ne on ladattu, ja jatkaa koodin lukemista vasta tämän jälkeen. (van de Spuy 2012, 299–300.)

Koodiin alustetaan nyt uusi taulukko (ladattavatKuvat) ja muuttuja (ladatutKuvat), joiden avulla loadHandler-funktio selvittää sille asetetut vaatimukset. Kuvamuuttuja työnnetään ”ladattavatKuvat”-taulukkoon sen määrittämisen lopuksi ja loadHandler-funktio lisää jokaisella sen kutsumalla yhden numeron alustettuun muuttujaan ”ladatutKuvat” (”++”-ominaisuudella). Kuvamuuttujaan lisätään myös kuvatiedoston sijainti ”src”-ominaisuudella. loadHandler on tärkeässä asemassa myöhemmin kuvien piirtämisessä canvakselle. (van de Spuy 2012, 496.) Funktiossa käytetään hyödyksi if-lauseketta ja for-silmukkaa sekä length-ominaisuutta. Jokaiselle pelin kuvalle tehdään sama toimenpide. Kuviossa 10 on havainnollistettu kuvamuuttujan määrittäminen ja siihen liitetty tapahtumakäsittelijän toiminto sekä yksinkertainen loadHandler-funktio.

```
//Alustus
var ladatutKuvat = 0;
var ladattavatKuvat = [];

//Kuvamuuttuja
var kuvaMuuttuja = new Image();
kuvaMuuttuja.addEventListener("load", loadHandler, false);
kuvaMuuttuja.src = "img/kuvatiedosto.png";
ladattavatKuvat.push(kuvaMuuttuja);

//loadHandler
function loadHandler() {
  ladatutKuvat ++; //Muutujaan lisätään yksi
  if(ladattavatKuvat === ladatutKuvat.length) { //Kun numerot täsmäävät, koodi etenee
    for(var i = 0; i < ladattavatKuvat.length; i++) { //Taulukko luetaan “length”-ominaisuudella
    suoritettava koodi; //Tähän sijoitetaan koodi, joka halutaan suorittaa
    }
    päivitysFunktio();
  }
}
```

KUVIO 10. Kuvamuuttuja ja tapahtumakäsittelijän toiminto sekä loadHandler-funktio.

Päivityksen aloittava funktio kuviossa 10 on sijoitettu loadHandler-funktion sisälle. Tästä eteenpäin koodin luodaan muuttujia ja funktioita, jotka tavalla tai toisella linkittyvät nyt jatkuvasti päivittyvään funktioon. Tämä funktio nimetään ”update”-funktioksi. Kuviossa 11 on esitetty koodin logiikka kaaviona.



KUVIO 11. Pelin objektin ja kuvamuuttujan sisäistä logiikkaa.

5.2 Hahmot ja esineet

Pelissä olevat hahmot ja esineet ovat perustoiminnallisuuksiltaan hyvin samanlaisia. Molemmille luodaan omat objektinsa, molemmat sijoitetaan kartalle samalla tavalla, molemmille luodaan omat kuvamuuttujat (jotka ladataan etukäteen) ja ne piirretään samalla tavalla canvakselle. Tässä luvussa perehdyn pääasiassa pelin hahmoihin liittyviin toimintoihin, mutta esineiden luominen ja niiden piirtäminen noudattavat hahmoihin sovellettua kaavaa.

5.2.1 Objektit

Hahmot ja esineet ovat siis objekteja. Objekti on kokoelma nimettyjä ominaisuuksia, joita voidaan hyödyntää, lisätä ja muuttaa pelin aikana. Koodiin alkuun luodaan muutama malliobjekti, josta pelin muut objektit monistetaan ”Object.create”-metodin avulla. Pelissä olevat objektit tulee luonnollisesti määritellä ennen niihin viittaamista tai niiden käyttöä. Pelattava hahmo tarvitsee esineistä ja taustoista poikkeavia arvoja, joten koodiin kannattaa luoda tässä tapauksessa useampi malliobjekti. Pelaajaa tullaan liikuttamaan ja animoimaan, joten pelaajan objekti ja kuvamuuttuja kannattaa myös työntää erillisiin taulukoihin. Kartoille ja pelin taustoille luodaan myös omat objektit ja taulukot, joista puhutaan enemmän myöhemmin.

Malliobjektille annetaan seuraavanlaiset arvot:

```
//Malliobjekti
var malliObjekti = {
  image: "",
  sourceX: 0,
  sourceY: 0,
  sourceWidth: 64,
  sourceHeight: 64,
  x: 0,
  y: 0,
  width: 32,
  height: 32,
  alpha: 1,
  shadow: true,
  vx: 0,
  vy: 0,
```

```
//Metodit
  centerX: function() {
    return this.x + (this.width / 2);},
  centerY: function() {
    return this.y + (this.height / 2);},
  halfWidth: function() {
    return this.width / 2;},
  halfHeight: function() {
    return this.height / 2;},
  //Määrittellään jokainen objektin sivu
  left: function() {
    return this.x;},
  right: function() {
    return this.x + this.width;},
  top: function() {
    return this.y;},
  bottom: function() {
    return this.y + this.height;}
};
```

Objekti varastoi image-ominaisuuden avulla objektiin liitettävän kuvatiedoston nimen, sourceX:n ja Y:n avulla lähdekuvan aloituspisteet x- ja y-akselilla ja sourceWidth- ja sourceHeight-arvojen avulla lähdekuvan alkuperäisen leveyden ja korkeuden. X ja y viittaavan objektin sijaintiin ruudulla, width ja height objektin leveyteen ja korkeuteen ruudulla, alpha objektin läpinäkyvyyteen, shadow objektin varjoon ja vx sekä vy objektin liikenopeuteen. (van de Spuy 2012, 333, 376–377, 395–396, 400.) Viittaan edellä esitettyyn malliobjektiin tulevissa esimerkeissä ilman sen ominaisuuksien toistuvaa kirjoittamista.

Objektiin lisätään myös kahdeksan sisäistä metodia, joista jokainen sisältää funktion. Metodiin viitatessa se palauttaa funktion laskeman arvon ”return”-ominaisuuden avulla. ”this”-sana ominaisuuden edellä kyseisen objektin valmiiksi määriteltyyn ominaisuuteen. centerX laskee objektin keskipisteen x-akselilla, centerY objektin keskipisteen y-akselilla ja halfWidth sekä halfHeight laskevat objektin puolikkaat leveys- ja pituusarvot. left, right, top ja bottom määrittelevät puolestaan objektin jokaisen sivun. (van de Spuy 2012, 86, 445–446, 457–458.)

Pelin hahmoille luodaan vastaava malliobjekti seuraavanlaisilla muutoksilla:

```
…
sourceWidth: 96,
sourceHeight: 256,
width: 32,
height: 64,
```

```
…
centerY: function() {
   return this.y + (this.height / 1.25);},
halfHeight: function() {
   return this.height / 4;},
```

sourceWidth muutetaan kolminkertaiseksi ja sourceHeight nelinkertaiseksi suuruuksiltaan. Y-keskipiste siirretään alemmas ja korkeus jaetaan neljällä kahden sijaan. Näitä muutoksia hyödynnetään myöhemmin törmäyksen tunnistamisessa ja liikkeen animoimisessa. Hahmolle luodaan seuraavaksi oma objekti, joka sijoitetaan malliobjektin perään, juuri ennen kuvamuuttujaa. On erittäin tärkeää, että objekti ja siihen liittyvä kuvamuuttuja sijaitsevat koodissa peräjälkeen, jotta niitä voidaan myöhemmin käyttää rinnakkaistaulukoina. Kuvamuuttujan tavoin hahmon objekti työnnetään myös taulukkoon, joka alustetaan ennen objektin luomista. Kuviossa 12 on esitetty hahmon objektin luominen kokonaisuudessaan malliobjektin pohjalta.

```
//Alustus
var kuvat = [];
var ladattavatKuvat = [];

//Hahmon objekti
var hahmo = Object.create(malliObjekti); //Hahmolle luodaan uusi objekti malliObjektin pohjalta
hahmo.x = canvas.width / 2; //Hahmon sijainti x-akselilla on muutetaan puoleen väliin canvasta
hahmo.y = canvas.height / 2;
hahmo.shadow = false; //Varjo otetaan pois näkyvistä
kuvat.push(hahmo); //Hahmo työnnetään taulukkoon

//Kuvamuuttuja
var hahmoKuva = new Image();
hahmoKuva.addEventListener("load", loadHandler, false);
hahmoKuva.src = "img/objects.png";
ladattavaKuvat.push(hahmoKuva);
```

KUVIO 12. Objektin ja kuvamuuttujan luominen.

5.2.2 Canvas ja piirtäminen canvakselle

Canvas on HTML-elementti, joka luo tiedostoon tyhjän piirtoalueen. Elementille voidaan antaa tuttuun tapaan id-tunniste, jonka avulla siihen voidaan viitata. Canvakselle piirtämiseen liittyy aina

myös canvaksen piirtämiskontekstikonteksti (context), joka luodaan getContext-metodilla ja johon viitataan aina piirtämishetkellä. Konteksti on objekti, johon liittyy useita piirtämiseen liittyviä sisäisiä metodeja. (Korpela 2014, 749–750.) Kuviossa 13 on esitetty canvas-elementin ja sen kontekstin luominen. Canvaksen piirtoalueen leveys ja korkeus on määritelty window-objektin innerWidth ja innerHeight-ominaisuuksien avulla, jolloin nämä arvot seuraavat sen hetkistä ikkunankokoa (W3 Schools 2017, viitattu 28.03.2017).

```
//HTML-tiedostossa sijaitseva canvas-elementin luominen
<canvas></canvas>

//JavaScript-tiedostossa tehty määrittely
var canvas = document.querySelector("canvas"); //Canvas -viittaus querySelectorin avulla
var ctx = canvas.getContext("2d"); //Kontekstin luominen
ctx.canvas.width  = window.innerWidth;
ctx.canvas.height = window.innerHeight - 50;
```

KUVIO 13. Canvas-elementin ja sen kontekstin luominen.

Seuraavaksi hahmo piirretään canvakselle render-funktion avulla. render-funktion tulee liittää ”update”-päivitysfunktioon, jotta esimerkiksi hahmon sijainnin muuttuessa hahmon liike päivittyy cavakselle. Itse render-funktioon liitetään canvaksen kontekstin drawImage-metodi, joka noudattaa seuraavanlaista kaavaa:

```
ctx.drawImage
(
  kuvamuuttuja,
  sourceX, sourceY, sourceWidth, sourceHeight,
  x, y, width, height
)
```

Kuten objektin yhteydessä jo mainittiin, sourceX:n ja Y:n avulla määritellään lähdekuvan aloituspisteet x- ja y-akselilla, sourceWidth ja sourceHeight-arvojen avulla lähdekuvan alkuperäinen leveys ja korkeus. X ja y viittaavan objektin sijaintiin ruudulla, width ja height objektin leveyteen ja korkeuteen ruudulla. Kuvamuuttujan nimi sijoitetaan drawImage-metodin alkuun. Lähdekuvan ja ruudulle piirrettävän kuvan ominaisuuksia drawImage-metodissa muuttamalla voidaan esimerkiksi rajata, venyttää tai suurentaa haluttuja kuvia. render-funktion alkuun liitetään myös koodinpätkä, joka tyhjentää piirtokontekstin aina ennen sille uudestaan piirtämistä (ctx.clearRect(0, 0, canvas.width, canvas.height);). (van de Spuy 2012, 303, 336.)

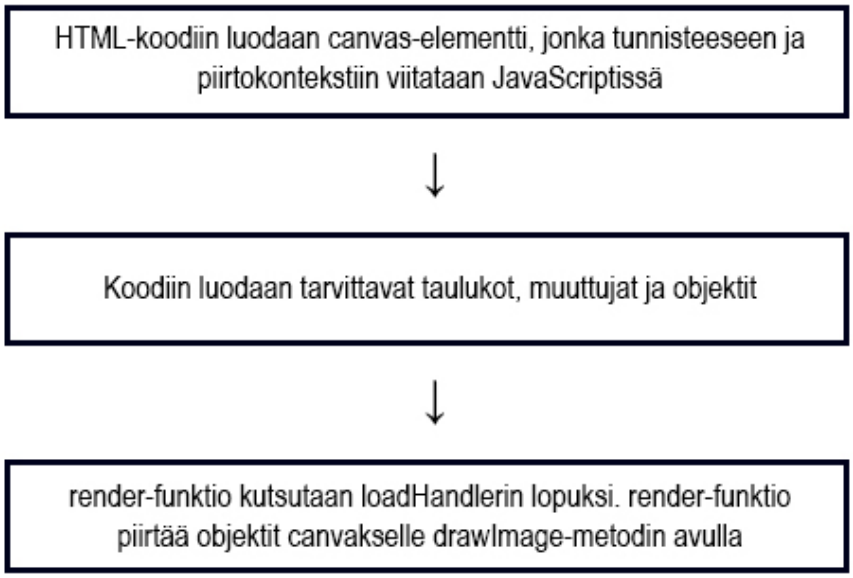
Jokaiselle objektille oman drawImage-metodin luominen on hidas ja työläs prosessi, joten käytän for-luuppia ja taulukoita tässä tapauksessa. Malliobjekti pitää sisällään ominaisuuden ”image”, jota hyödyntämällä kuvamuuttujan voi yhdistää kyseisen ominaisuuden arvoksi rinnakkaisella taulukolla. Objekti ja kuvamuuttuja pidetään siis ennallaan, mutta loadHandler saa uuden sisällön sulkeisiinsa kuvion 14 esittämällä tavalla. Koodi tarkistaa ensin, että ladattujen kuvien taulukon alkioiden määrä vastaa kertoja, jotka loadHandler on ladattu (ladattavatKuvat === ladatutKuvat.length). Tämä jälkeen ladattavatKuvat käydään läpi sen alkioiden määrän verran, ja kuvat-taulukon jokainen indeksiarvo eli senhetkinen muuttuja liitetään ladattavatKuvat-taulukon saman indeksiarvon eli senhetkisen objektin image-ominaisuudeksi. Rinnakkaisia taulukoita voi hyödyntää kyseisellä tavalla for-silmukassa i-muuttujan avulla (i = 0; i < ladattavatKuvat.length; i++)), jolloin i käy läpi taulukon jokaisen indeksiarvon kerrallaan ja suorittaa sulkujen koodin yksitellen jokaisella sille määritellyllä arvolla (1, 2, 3 jne.). Näin ollen length-ominaisuuden kanssa hyödynnettynä taulukoiden alkioiden määrää ei tarvitse tietää etukäteen lainkaan. Lopuksi loadHandler kutsuu päivitysfunktion.

Myös render-funktiossa hyödynnetään for-silmukkaa. Tällä kertaa kuvat-taulukon jokaisen indeksiarvon objektiin liitetään ”kuva”-niminen muuttuja, joka edustaa kyseistä objektia. kuva-muuttujan avulla kontekstiin piirretään oikean objektin kuvamuuttuja ja oikeat objektin arvot ilman, että varsinaista piirtokoodia tarvitsee kopioida useampaan kertaan. Kuviossa 14 on esitetty täydellinen objektin ja sen kuvamuuttuja piirtologiikka malliobjektin pohjalta ja kuviossa 15 sama logiikka on kuvattu kaaviona. Huomioitavaa on, että päivitysFunktion sijaan kuviossa kutsutaan suoraan render-funktiota, sillä koodi on tällä hetkellä riittävän yksinkertainen.

```
//HTML-tiedostossa sijaitseva canvas-elementin luominen
<canvas></canvas>

//JavaScript-tiedostossa tehty määrittely
var canvas = document.querySelector("canvas"); //Canvas -viittaus querySelectorin avulla
var ctx = canvas.getContext("2d"); //Kontekstin luominen
ctx.canvas.width  = window.innerWidth;
ctx.canvas.height = window.innerHeight - 50;

//Alustus
var kuvat = [];
var ladattavatKuvat = [];

//Hahmon objekti
var hahmo = Object.create(malliObjekti);
hahmo.x = canvas.width / 2; //Hahmon sijainti x-akselilla on muutetaan puoleen väliin canvasta
hahmo.y = canvas.height / 2;
hahmo.shadow = false; //Varjo otetaan pois näkyvistä
```

```
kuvat.push(hahmo); //Hahmo työnnetään taulukkoon

//Kuvamuuttuja
var hahmoKuva = new Image();
hahmoKuva.addEventListener("load", loadHandler, false);
hahmoKuva.src = "img/objects.png";
ladattavaKuvat.push(hahmoKuva);

//loadHandler
function loadHandler() {
  ladatutKuvat ++; //Muutujaan lisätään yksi
  if(ladattavatKuvat === ladatutKuvat.length) { //Kun numerot täsmäävät, koodi etenee
    for(var i = 0; i < ladattavatKuvat.length; i++) {
    kuvat[i].image = ladattavatKuvat[i]; //Objektin kuvaksi liitetään sitä  vastaava kuvamuuttuja
    }
    render();
  }
}

//render-funktio
function render() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);  //Konteksti tyhjennetään
  if(kuvat.length !== 0) {   //Koodi etenee, jos taulukossa on alkioita
    for(var i=0; i<kuvat.length; i++) {  //Taulukko käydään läpi
      var kuva = kuvat[i]; //Muuttuja “kuva”-liitetään kuvat-taulukon jokaiseen indeksiarvoon
    }
    ctx.drawImage (
        kuva.image, //Kontekstille piirtäminen suoritetaan kuva-muuttujan avulla
        kuva.sourceX, kuva.sourceY,
        kuva.sourceWidth, kuva.sourceHeight,
        Math.floor(kuva.x), Math.floor(kuva.y), //Math.floor pyöristää arvoja
        kuva.width, kuva.height
        );
    }
  }
```

KUVIO 14. Täydellinen piirtologiikka.



KUVIO 15. Täydellinen piirtologiikka kaaviona.

5.2.3 Hahmon liikuttaminen näppäimistöllä

Pelattavaa hahmoa tai mitä tahansa muuta pelissä esiintyvää objektia tai elementtiä voi liikuttaa canvaksella monella tavalla. Liikkeen voi esimerkiksi ohjata tietokoneen näppäimistöön, tietokoneen hiiren liikkeeseen tai näppäinten klikkaamiseen, ruudun kosketukseen, tai sen voi esimerkiksi ajastaa tapahtumaan itsekseen. Liike suoritetaan window- eli ikkuna-objektin ja tapahtumankäsittelijöiden sekä eri ehtolauseiden avulla. Tähän aihealueeseen on tarjolla paljon valmista materiaalia, joten käyn kyseisen aiheen läpi vain hyvin pikaisesti päästäkseni käsiksi liikkeen animoimiseen.

Peliin luodaan ensiksi neljä uutta muuttujaa kuvaamaan jokaista liikkumissuuntaa tietokoneen nuolinäppäimillä, ja näille muuttujille annetaan näppäimistön merkkejä vastaavat numerot (kuvio 16). Numerot ovat vakioita ohjelmointikielessä ja lista näistä näppäimistön numerokoodeista löytyy internetistä. Jokaiselle neljälle suunnalle luodaan myös muuttuja oletusarvolla ”false” (kuvio 16), jonka avulla voidaan todentaa hahmon liike. Koodiin lisätään tapahtumankäsittelijä (event listener), jonka sisälle sijoitetaan ”keydown”-tapahtumankäsittelijän toiminto, sekä sisäinen funktio. Funktio pitää sisällään argumentin ”event” ja switch-lausekkeen, joka tarkastaa yhdessä ”event.keyCode”-argumentin ja eri casejen avulla, onko näppäin painettuna alas. Täysin saman tyyppinen tapahtumankäsittelijä tehdään myös hahmottamaan hahmon liikkumattomuus (Kuvio 16). (van de Spuy 2012, 404–408.)

```
//Tapahtumankäsittelijät
window.addEventListener("keydown", function(event) {
  switch(event.keyCode) {
    case YLÖS:
      liikeYlös = true;
      break;
    case ALAS:
      liikeAlas = true;
      break;
    case VASEN:
      liikeVasemmalle = true;
      break;
    case OIKEA:
      liikeOikealle = true;
      break;
  }
}, false);

window.addEventListener("keyup", function(event) {
  switch(event.keyCode) {
    case YLÖS:
      liikeYlös = false;
      break;
    case ALAS:
      liikeAlas = false;
      break;
    case VASEN:
      liikeVasemmalle = false;
      break;
    case OIKEA:
      liikeOikealle = false;
      break;
}
}, false)

//Nuolinäppäinten koodit
var YLÖS = 38;
var ALAS = 40;
var VASEN = 37;
var OIKEA = 39;

//Suunnat
var liikeYlös = false;
var liikeAlas = false;
var liikeOikealle = false;
var liikeVasemmalle = false;
```

KUVIO 16. Näppäimistöllä liikkuminen.

Seuraavaksi update-funktioon luodaan if-lauseke, joka tarkastaa, ovatko näppäimet alhaalla vai ylhäällä, ja muuttaa objektin x- ja y-koordinaatteja halutun verran. Objektin koordinaatteja muutetaan objektin määrittelyssäkin mainittujen vx- ja vy-ominaisuuksien avulla, jotka kuvaavat objektin nopeutta (velocity). Arvot määritellään lisäykseksi objektin x- ja y-arvoihin. ”&&” muuttujien välissä tarkoittaa JavaScriptissä sanaa ”ja”, kuin taas ”!” ennen arvoa tai muuttujaa tarkoittaa käsitettä ”muuttujan arvo on false”, tai ”muuttujan arvo ei ole jotakin”. Hahmon liike määritellään nollaksi näppäinten ollessa ylhäällä, eli kyseisten arvojen ollessa ”false”. Kuvioon 17 on merkitty update-funktioon liitettävä koodi. (van de Spuy 2012, 408–410.)

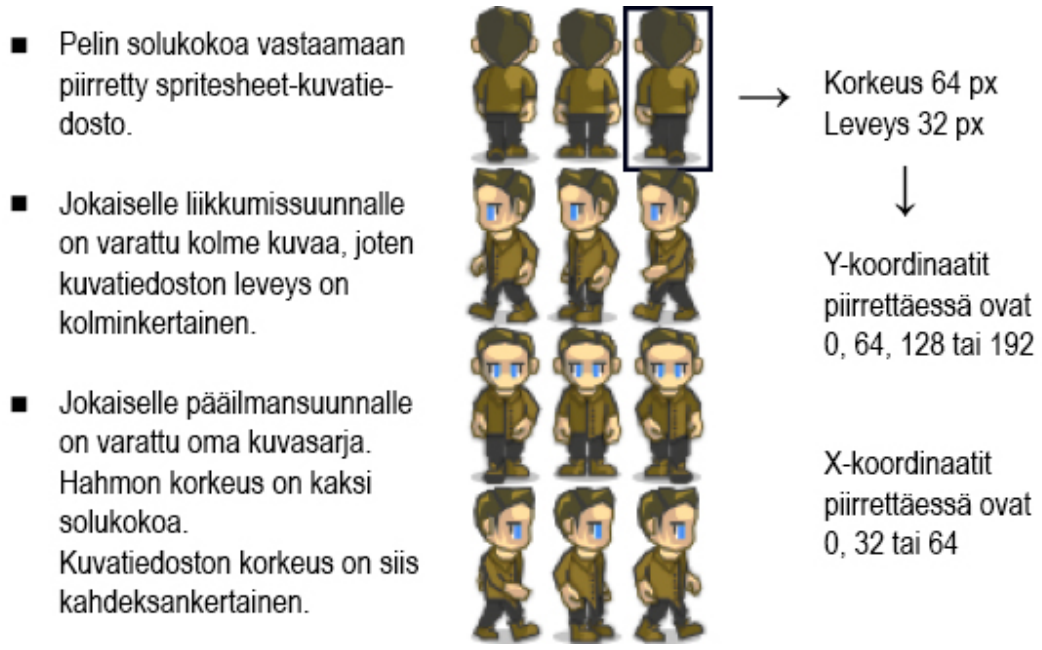
```
//Ylös
  if(liikeYlös && ! liikeAlas){
    hahmo.vy = -2;
}
//Alas
  if(liikeAlas && ! liikeYlös){
    hahmo.vy = 2;
}
//Vasen
  if(liikeVasemmalle && ! liikeOikealle){
    hahmo.vx = -2;
}
//Oikea
  if(liikeOikealle && ! liikeVasemmalle){
    hahmo.vx = 2;
}

//Jos näppäimiä ei paineta, liike on nolla
  if(!liikeYlös && ! liikeAlas) {
    hahmo.vy = 0;
}
  if(liikeVasemmalle && ) {
    hahmo.vx = 0;
}

  //Liikuta hahmoa
  hahmo.x += hahmo.vx;
  hahmo.y += hahmo.vy
```

KUVIO 17. update-funktion liikekoodi.

5.2.4 Liikkeen animointi

Liikkeen eli käytännössä objektin animoiminen sisältää monta pientä yksityiskohtaa, jotka tuodaan yhteen render-funktion drawImage-metodissa. Hahmon liike ilmaistaan piirtämällä kuvatiedoston sisältämät hahmon eri asennot canvakselle nopeassa tahdissa, objektin liikesuunta ja nuolinäppäimen painamisen tunnistus huomioon ottaen. Hahmolle luodaan ensiksi spritesheet-kuvatiedosto, johon hahmon kaikki liikkeet on piirretty yksittäisinä freimeinä eli kehysalueen sisällä olevina kuvina kuvion 18 mukaisesti. Valitsin pelilleni solukooksi 32 pikseliä kertaa 32 pikseliä, jolloin hahmoni on yksi solua leveä ja kaksi korkea (solukoosta lisää myöhemmin). Spritesheet-tiedostoon on jokaiselle neljälle liikkumissuunnalle varattu kolme kuvaa. Hahmolle luodun malliobjektin sourceWidth- ja sourceHeight-arvot ovat tästä syystä leveydeltään kolminkertaisia ja korkeudeltaan kahdeksan-

kertaisia pelin solukokoon verrattuna. Hahmon objektille luodaan muuttujat freimiKorkeus ja freimiLeveys, joiden avulla voidaan laskea spritesheet-kuvan jokaisen yksittäisen liikkeen koko. freimiLeveys on siis objektin sourceWidth jaettuna kolmella ja freimiKorkeus objektin sourceHeight jaettuna neljällä. Hahmon x- ja y-koordinaattien arvot ja sourceWidth ja sourceHeight arvot drawImage-metodissa korvataan freimiLeveys ja freimiKorkeus arvoilla, jolloin spritesheet-tiedostosta piirretään vain yksi freimi kerrallaan (kuvion 19 drawImage-metodi).



KUVIO 18. Hahmon spritesheet eli animointiin tarkoitettu kuvatiedosto Musta Morsian-pelissä.

drawImage-metodissa spritesheet-kuva rajataan y-akselin eri kohdista riippuen suunnasta, johon hahmo kävelee tai katsoo. Y-akselin rajaus drawImage-metodissa tapahtuu sourceY-ominaisuuden avulla, joten tähän arvoon luodaan muuttuja ”katsomisSuunta”, jonka avulla y-akselin sijainti kuvasta lasketaan aina hahmon liikkuessa eri suuntaan. katsomisSuunta kerrotaan freimikorkeus-muuttujalla, jolloin kuvasta piirretään korkeussuunnassa oikea freimi. katsomisSuunta-muuttujan arvot vaihtelevat välillä 0-3, jolloin y-koordinaatin arvot ovat joko 0, 64, 128 tai 192. Hahmon katsomissuunta muuttuu hahmon liikkuessa, joten katsomisSuunta-muuttuja sijoitetaan hahmoa liikuttaviin update-funktion ehtolauseisiin kuvion 19 esittämällä tavalla.

X-akselin rajaus riippuu siitä, onko hahmo liikkeellä vai levossa. Koodin pitää siis ottaa selvää, liikkuuko hahmo, ja laskea kuvalle uusi x-arvo tämän liikkeen mukaan. Koodiin luodaan uusi muut-

tuja kävelyAnimaatio, joka kerrotaan drawImage-metodin sourceX-muuttujan korvanneella freimiLeveys-muuttujalla. kävelyAnimaatio-muuttujan tulee siis vaihdella välillä 0–2, jolloin x-koordinaatin arvot ovat joko 0, 32 tai 64. kävelyAnimaatio-muuttujan lisäksi koodiin luodaan toinen muuttuja, spriteLaskuri, jonka avulla kävelyAnimaatio-muuttujan arvoa vaihdellaan. spriteLaskuri-muuttujalle luodaan oma ehtolause update-funktioon, joka korottaa muuttujan arvoa 0,1:llä, kun hahmoa liikutetaan ja koodi päivittyy. Vastaavasti jos hahmoa ei liikuteta, muuttujan arvo on aina 1, jolloin canvakselle piirretään keskimmäinen freimi spritesheet-kuvatiedostosta (1 * 32 = 32) (kuvio 19). Jotta kävelyAnimaatio päivittyy spriteLaskuri:n muuttuessa, koodiin luodaan kävelyAnimaatiot-funktio, joka päivitetään jatkuvasti update-funktiossa. kävelyAnimaatiot-funktio on seuraavanlainen:

```
function kävelyAnimaatiot() {
  kävelyAnimaatio = spriteLaskuri % 3;
}
```

Funktio määrää kävelyAnimaatio-muuttujan arvoksi spriteLaskuri:n sen hetkisen arvon. spriteLaskuri:n arvo käsitellään moduulilla (modulus), joka ei anna muuttujan arvon nousta kolmea korkeammaksi (van de Spuy 2012, 67). drawImage-metodin x-ja y-arvoihin lisätään myös Math.floor-ominaisuus, joka pyöristää sen sulkeiden sisällä olevan arvon lähimpään tasalukuun. drawImage-metodi näyttää nyt siis kokonaisuudessaan kuvioon 19 mukaiselta. Kuvioon 19 on nyt merkitty kaikki liikkeen animoimiseen vaadittava uusi koodi.

```
//Muuttujat
var freimiLeveys = hahmo.sourceWidth / 3;
var freimiKorkeus = hahmo.sourceHeight / 4;
var katsomisSuunta = 2; //Aluksi hahmo katsoo alaspäin
var spriteLaskuri = 0;
var kävelyAnimaatio = 2;

//Kävelyfunktio
function kävelyAnimaatiot() {
  kävelyAnimaatio = spriteLaskuri % 3;
}

//update-funktiossa sijaitseva liikekoodi
if(liikeYlös || liikeAlas || liikeOikealle || liikeVasemmalle){
    spriteLaskuri += 0.1; //Laskurin arvo nousee aina, kun hahmo liikkuu
  }
if(!liikeYlös && !liikeAlas && !liikeOikealle && !liikeVasemmalle){
    spriteLaskuri = 1; //Laskurin arvo on yksi, jos hahmo EI liiku
  }
kävelyAnimaatiot();
```

```
//Ylös
  if(liikeYlös && ! liikeAlas){
    katsomisSuunta = 0;
    hahmo.vy = -2;
}
//Alas
  if(liikeAlas && ! liikeYlös){
    katsomisSuunta = 2;
    hahmo.vy = 2;
}
//Vasen
  if(liikeVasemmalle && ! liikeOikealle){
    katsomisSuunta = 1;
    hahmo.vx = -2;
}
//Oikea
  if(liikeOikealle && ! liikeVasemmalle){
    katsomisSuunta = 3;
    hahmo.vx = 2;
}

//render-funktio
function render() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  if(kuvat.length !== 0) {
    for(var i=0; i<kuvat.length; i++) {
      var kuva = kuvat[i];
    }
    ctx.drawImage
     (
       hahmo.image,
       Math.floor(kävelyAnimaatio) * freimiLeveys, katsomisSuunta * freimiKorkeus,
       freimiLeveys, freimiKorkeus,
       Math.floor(hahmo.x), Math.floor(hahmo.y),
       hahmo.width, hahmo.height
     );
    }
  }
```

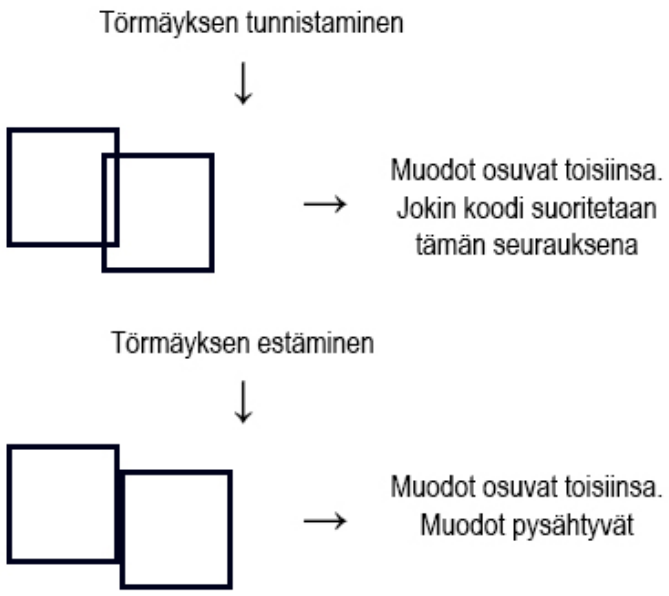
KUVIO 19. Animoimiseen tarvittava koodi.

5.2.5 Törmäys (collision)

Törmäys (collision) on kahden muodon tai esimerkiksi pisteen osumista toisiinsa. Törmäyksen tunnistamiseen on olemassa omat kaavansa, jotka hyödyntävät objektien metodeissa olevia arvoja.

Törmäystä voi hyödyntää esimerkiksi objektien osumisessa seiniin, vihollisiin tai erilaisiin maastonkohtiin, esineiden tai pisteiden keräilyyn, vipujen vetämiseen tai vaikkapa ansojen laukaisuun. Aiheesta on olemassa paljon materiaalia, joten käyn aiheen läpi hyvin pintapuolisesti. Yksityiskohtaisempaa tietoa aiheesta voi etsiä lähdekirjallisuuttani hyödyntäen.

Käytän pelissäni erillistä JavaScript-tiedostoa, joka liitetään HTML-tiedostoon tutulla tavalla. Tiedosto sisältää valmiit kaavat funktioiden muodossa erilaisten törmäysten tunnistamiseen. Pääasiallisessa JavaScript-tiedostossa näihin funktioihin viitataan liittämällä halutut objektit kutsuttavan funktion argumenttiin (törmäysFunktio(hahmo, seinä);). Erillisessä tiedostossa sijaitseva funktio suorittaa tarvittavan laskutoimituksen ja palauttaa koodiin tiedon törmäyksestä. Törmäyksen voi asettaa ehtolauseen sisään tai sen argumenttiin voi asettaa esimerkiksi taulukon. Käytän koodissani pääasiassa kahdenlaista törmäystä: toinen tunnistaa törmäyksen ja palauttaa arvoksi joko ”true” tai ”false”, toinen estää argumenttiin sijoitettujen objektien siirtymisen toistensa päälle. (van de Spuy 2012, 248, 466.) Kuviossa 20 on esitetty törmäys kaavion avulla.



KUVIO 20. Törmäys.

5.3 Kartat

Pelin kartat eli jokaisen ruudun taustakuvat ja niille kuuluvat objektit ja esineet luodaan useasta eri tasosta, joissa hyödynnetään läpinäkyvyyttä tukevia kuvia, JavaScript taulukoita (arrays), sekä objekteja. Käytän pelissäni osittain hyväksi RPG-peleissä suosittua ”tilemap”-tekniikkaa, jossa kartta luodaan pienistä laatoista, jotka kootaan spritesheet-kuvatiedostoihin (Tiles and tilemaps overview 2016, viitattu 14.03.2017). Karttoja varten voidaan luoda uusi JavaScript-tiedosto maps.js, joka kokoaa ja varastoi jokaiselle ruudulle tarvittavat kartat ja kuvatiedostot samaan paikkaan, jotta tiedostorakenne pysyy siistinä. Tämä maps.js linkitetään pelin html-tiedostoon ennen muita JavaScript-tiedostoja. Tiedostoon varastoidaan jokaisen ruudun kartan taustan ja etualan kuvien lähteet, törmäyskartta ja objektikartta. Pelin alkuperäisessä JavaScript-tiedostossa olevat muuttujat on koodissa merkitty aina lisäyksellä ”current” (tämänhetkinen) viestimään sitä, että muuttujan arvo ei ole vakio ja että sitä tullaan muuttamaan pelin edetessä.

5.3.1 Tausta ja etuala

Pelille valitaan aluksi korkeus ja leveys, joissa sitä pelataan. Valitsin itse pelilleni Full HD-leveyden ja sitä hieman suuremman korkeuden (1920 pikseliä * 1920 pikseliä), jolloin leveinkin näyttö peittyy täysin pelialueesta. Tämän jälkeen pelialue jaetaan ruudukoksi ja sen soluille määritellään koko. Valitsin solukooksi peliini 32px*32px, jolloin pelialue jakautuu tasan 60 pysty- ja vaakasoluun. Kaikki pelissä olevat hahmot, esineet ja erikoispiirteet, kuten törmäys, noudattavat valittua solukokoa. JavaScript-tiedostoon merkitään valittu solukoko muuttujan avulla ”vakiona” (constant), sillä tämä arvo ei tule pelissä enää muuttumaan (van de Spuy 2012, 248). Kuviossa 21 on määritelty pelin solukoko.

Aluksi ruudulle luodaan hahmojen alle sijoittuva taustakuva ja hahmojen päälle sijoittuva etualakuva objektien monistamista hyödyntämällä. JavaScript-tiedostoon luodaan kaksi objektia: currentTausta ja currentEtuala. Taustakuva eli currentTausta määritellään JavaScript-tiedostoon ennen pelin hahmojen ja muiden esineiden objekteja, jottei se peitä näitä. currentEtuala määritellään näiden jälkeen, jotta se on pelissä etualalla. Näin pelaajalle luodaan illuusio, että pelaaja voi kävellä esineiden taakse. Objekteille määritellään ominaisuuksia ja arvoja aiemmin luodun objektin (mal-

liObjekti) pohjalta. Molemmille kuville luodaan myös vastaava kuvamuuttuja, jotka piirretään canvakselle samalla tavalla kuin muutkin pelin muuttujat. Kuviossa 21 on määritelty myös pelin taustan objekti. Kuviossa 22 päällekkäin piirtyvät tausta- ja etualaobjektit on esitetty erillisinä kuvina.

```
//Solukoko
var SIZE = 32;

//Taustan objekti:
var currentTausta = Object.create(spriteObject); //Varsinainen objekti luodaan
currentTausta.sourceWidth = 1920; //Objektin arvoja muutetaan monistetusta objektista
currentTausta.sourceHeight = 1920;
currentTausta.width = 1920;
currentTausta.height = 1920;
currentTausta.x = 0;
currentTausta.y = 0;
kuvat.push(currentTausta);

//Taustan kuvamuuttuja
var currentTaustakuva = new Image();
currentTaustakuva.addEventListener("load", loadHandler, false);
currentTaustakuva.src = map13BackgroundSrc; //Taustan lähde maps.js tiedostossa
ladattavatKuvat.push(currentTaustakuva); //Kuvaobjekti työnnetään taulukkoon
```

KUVIO 21. Solukoko ja tausta- sekä etualaobjektit.

KUVIO 22. Tausta- sekä etualaobjektit kuvina Musta Morsian pelissä.

5.3.2 Taulukoiden hyödyntäminen

Taulukoita (arrays) hyödynnetään asioiden sijoittelussa kartalle. JavaScript-tiedostoon luodaan kaksi taulukkoa: currentCollision ja currentObjects. currentCollision on ruudun törmäyskartta, joka määrää, minkä solujen päälle pelaaja voi kävellä ja mitkä päinvastoin pysäyttävät hänet. currentObjects on ruudun objektikartta, jonka avulla kartalle voidaan sijoittaa esimerkiksi esineitä, hahmoja ja erikoisruutuja, kuten pelaajan vauhtia hidastavia kohtia. Törmäys- ja objektikarttoja voidaan luoda eri ohjelmilla, joista yksi suosituin on nimeltään ”Tiled”. Kyseisiä karttoja voi kuitenkin luoda myös manuaalisesti, vaikkakin se voi olla hyvin hidasta ja työlästä, jos kartat ovat suuria.

Kaksiulotteiset taulukot ovat taulukkoja taulukoiden sisällä. Käytännössä tämä tarkoittaa sitä, että jokaisen taulukon alkio on itsessään jo taulukko. Pelin kartat ovat juuri kyseisiä kaksiulotteisia taulukoita, joten ensiksi koodiin määritellään taulukoiden rivien ja sarakkeiden pituudet vakiomuuttujilla ROWS ja COLUMNS. Pituudet voidaan laskea pelin kartan avulla lisäämällä ”.length” (rivi) ja ”[0].length” (sarake) taulukoiden perään. (van de Spuy 2012, 243, 247.) Törmäyskartta on pelin solujen leveyden (32 px) ja korkeuden (32 px) mukaan 60 solua leveä ja 60 solua pitkä. Törmäyskarttaa voidaan siis hyödyntää tässä yhteydessä kuvion 23 esittämällä tavalla. Vakiomuuttujat ROWS ja COLUMNS pitävät nyt arvoinaan kartan rivien ja sarakkeiden pituuksia, joita voidaan hyödyntää kartan rakentamisfunktion silmukassa.

```
//Rivit ja sarakkeet
var ROWS = currentCollision.length; //Rivien pituus törmäyskartan avulla
var COLUMNS = currentCollision[0].length; //Sarakkeiden pituus törmäyskartan avulla
```

KUVIO 23. Rivien ja sarakkeiden pituudet.

Törmäyskartta voi yksinkertaisimmillaan olla kaksiulotteinen taulukko täynnä nollia ja ykkösiä. Pelini törmäyskartta on 60 solua leveä ja 60 solua korkea taulukko, jonka seinien kohdalla on numero 1 ja tyhjässä kohdassa numero 0. Tällöin koodiin merkitään kaksi uutta muuttujaa, jotka pitävät sisällään vastaavia arvoja (var EMPTY = 0; ja var COLLISION = 1;). Koodin luodaan nyt uusi funktio ”function buildMap(levelMap)”, jonka argumenttiin (levelMapin kohdalle) sijoitetaan myöhemmin haluttu kartta. Funktion sisälle sijoitetaan kaksi silmukkaa ja switch-lauseke, joiden avulla koodi laskee vakioiden paikat kartalla ja luo niille objektit oikeaan paikkaan kartalla. Silmukoissa hyödynnetään aiemmin määriteltyjä vakiomuuttujia EMPTY, COLLISION, ROWS ja COLUMNS. (van de Spuy 2012, 355.) Kuviossa 24 on havainnollistettu buildMap-funktion sisältö.

```
buildMap(currentCollision); //funktio kutsutaan ja törmäyskartta asetetaan sen argumentiksi

function buildMap(levelMap){ //buildMap funktio
  for(var row = 0; row < ROWS; row++){ //rivisilmukka
    for(var column = 0; column < COLUMNS; column++){ //sarakesilmukka

      var currentTile = levelMap[row][column]; //currentTile on kartan sen hetkinen solu

      if(currentTile !== EMPTY){ //jos solu ei ole tyhjä eli EMPTY, koodin luku jatkuu
        switch (currentTile) { //switch-lauseke
          case COLLISION: //jos solun numero on yksi, eli vakio on COLLISION
            var collision = Object.create(malliObjekti);
            collision.x = column * SIZE;
            collision.y = row * SIZE;
            collisions.push(collision);
            break;
}}}}}
```

KUVIO 24. buildMap-funktio.

Nyt koodiin luodaan uusi tyhjä taulukko ennen buildMap-funktiota (var collisions = [];). Switch-lauseke luo uuden objektin ”collision” jokaiseen törmäyskartan kohtaan, jossa on numero yksi, hyödyntämällä määriteltyjä vakioita. collision-objektit työnnetään collisions taulukkoon. Nyt törmäyksen (collision) avulla taulukoille voidaan asettaa ehtoja ”törmäys”-kappaleen mukaisesti. Samalla periaatteella switch-lausekkeeseen voidaan sijoittaa mitä tahansa vakioita, jotka vastaavat koodissa ja kartoissa määriteltyjen vakioiden numeroita. Yksikertaisimmillaan törmäys testataan koodissa kuvion 25 mukaisesti. Objektikartta noudattaa samaa periaatetta, kuin törmäyskartta, mutta objektikartan numeroita vastaamaan luodaan omat muuttujat ja vakiomuuttujat. Huomioitavaa on, että törmäyksen testaaminen sijoitetaan jatkuvasti päivitettävään koodin kohtaan eli update-funktioon.

```
//Törmäys
  for(var i = 0; i < collisions.length; i++) { //collisions-taulukko käydään läpi
  blockRectangle(hahmo, collisions[i]); //Törmäysfunktioon sijoitettavat argumentit
 }
```

KUVIO 25. Törmäyksen testaaminen.

5.4 Pelin kehittäminen eteenpäin

Peliin on nyt luotu hahmo ja kartta, jonka päällä ja osittain myös alla hahmo voi liikkua animoituna. Molempien edellä mainittujen luominen edellytti objektien ja niiden rinnakkaisten kuvamuuttujien luomista, sekä canvaksen jonkinasteista ymmärtämistä. Karttaan on myös luotu rinnakkainen törmäyskartta, joka estää pelaajaa liikkumasta tiettyihin kohtiin. Seuraavia luonnollisia askeleita voisivat olla esineiden kerääminen, kartan vaihtaminen ja pelikameran luominen ja liikuttaminen. Peliin voisi myös luoda esimerkiksi vihollisia tai ongelmanratkaisukykyä vaativia haasteita. Juurikaan enempää teknistä ymmärtämistä nämä ominaisuudet eivät kuitenkaan vaadi, sillä jo käyttämäni tekniikat ovat hyvin yleishyödyllisiä. Havainnollistan tätä väitettä esittämällä esineiden keräämiseen ja pelikartan vaihtamiseen vaadittavat toimenpiteet.

5.4.1 Esineiden poimiminen

Esineet ovat objekteja samalla tavalla kuin pelin hahmot. Esineelle luodaan täten oma objekti, kuvamuuttuja ja kuvatiedosto, joka ladataan luotuun kuvaelementtiin. Esineen objekti ja kuvamuuttuja työnnetään niille kuuluviin taulukoihin ja ne piirretään render-funktiossa. Jokaiselle esineelle voidaan luoda oma vakiomuuttuja, joka sijoitetaan objektitaulukkoon. Kartan latautuessa esine monistetaan, työnnetään esinetaulukkoon ja sijoitetaan sille määriteltyyn kohtaan. Nyt update-funktioon voidaan sijoittaa ehtolause, joka tarkastaa törmäyksen hahmon ja esineen välillä. Peliin voidaan luoda ”pisteLaskuri”-muuttuja, joka kohoaa aina yhdellä numeroarvolla pelaajan törmätessä esineeseen. Esine voidaan samalla poistaa ”splice”-metodin avulla sen taulukosta, jolloin se häviää myös peliruudulta.

Näin yksinkertaisella tavalla peliin on luotu täysin uusi ominaisuus aiemmin opittujen keinojen avulla. Jos esine halutaan siirtää pelaajan varastoon, poistamisen sijaan se voidaan työntää pelaajan esinevarasto-taulukkoon, jolle voidaan luoda vaikkapa oma div-elementti tai jopa canvas-elementti.

5.4.2 Pelikartan vaihtaminen

Pelikartan vaihtaminen hyödyntää myös aiemmin opittua. Objektikarttaan voidaan luoda oma solu, jolle annetaan törmäyksen sattuessa tietty tarkoitus. Tämä solu on esineiden tavalla oma objekti,

mutta se ei tarvitse omaa kuvamuuttujaa, sillä sitä ei piirretä canvakselle. Objekti tarvitsee oman vakiomuuttujansa, taulukkonsa ja määrityksen kartan luovaan funktioon esineiden tavalla. update-funktioon luodaan ehtolause, joka suoriutuu jälleen hahmon ja objektin törmäysestä. Ehtolause muuttaa karttaobjektin kuvamuuttujan kuvan ”src”-ominaisuutta vastaamaan uutta tausta- ja etualakuvaa sekä muuttaa currentCollision- sekä currentObjects-taulukoiden indeksiarvoa vastaamaan uutta karttaa. Hahmo voidaan samalla sijoittaa kartalle pikselitarkkuudella tai luomalla jälleen uusi objekti objektikarttaan, joka sijoittaa hahmon haluttuun soluun.

Taulukoiden ja silmukoiden avulla pelikarttajärjestelmän luominen on helppoa. Karttojen muuttamisessa ja vaihtamisessa ei käytetty mitään uutta tekniikkaa, vain pelkästään hienosäädettiin vanhaa. Karttojen määrässä tai monimutkaisuudessa ei ole myöskään käytännössä minkäänlaisia rajoituksia.

6 JOHTOPÄÄTÖKSET

Tutkielman tavoite oli löytää tapoja, joilla HTML5 ja JavaScript voisivat olla hyödyksi RPG-pelien rakentamisessa hyvin käytännönläheisellä tavalla. Tutkiessani verkossa käytettävien ohjelmointikielten hyödyllisiä ominaisuuksia ja tekniikoita, kykenin produktioni avulla samalla testaamaan näiden soveltuvuutta juurikin RPG-pelien luomisprosessissa. Löysin useita toimintamalleja ja teknisempiä ratkaisuja, joita pystyin analysoimaan ja käyttämään jatkossa uudelleen hieman eri tavalla, mutta periaatteen pysyessä sama. Avaan näitä seuraavaksi viitaten kappaleissani esitettyihin kokonaisuuksiin.

Aluksi esitin löytämäni hyödyllisen mallin pelin sisäiseen logiikkaan. Tutkimuksessani huomasin, että peli on hyödyllistä jakaa useampaan tiedostoon, joissa on selvä kokonaisuus. Tiedostojen sisäinen hierarkia ja keskinäiset viittaukset pitää kuitenkin rakentaa loogisesti, jottei koodissa kutsuta olemattomia tai vasta myöhemmin ladattavia muuttujia ja funktioita. Koodiin on ehdottoman tärkeää myös luoda sisäinen jatkuva päivitys. Lähdeaineistoni pohjalta loin tutkielmaani kaavan, jota hyödynsin objektien ja kuvamuuttujien välisen suhteen selkeyttämisessä. Taulukoiden, muuttujien ja objektien välisten suhteiden ymmärtäminen on pelin logiikan ymmärtämisen kannalta tärkeää.

Hahmoista ja esineistä kertoessani tutkin paljon objekteihin liittyviä tekniikoita. Objektien avulla loin aineistooni mallin hahmo-objektin luomiseen. Hahmon objekti pitää sisällään muista objekteista poikkeavia ominaisuuksia, joita voidaan hyödyntää erinäisissä toiminnoissa. Tutkimuksessani objektien ja niiden kuvamuuttujien suhde korostui erityisesti taulukoiden takia. Rinnakkaiset taulukot pitivät sisällään toisiinsa suhteutuvia arvoja, joita kykenin myöhemmin myös käyttämään rinnakkain. Tutkimusaineistossani hyödylliseksi malliksi nousi erityisesti canvaksen drawImage-metodin räätälöiminen objektien piirtämiseen. Käytin ehtolausetta ja for-silmukkaa yhdessä objektien ja niiden kuvamuuttujien taulukoiden kanssa luodessani yksinkertaisen tavan piirtää kaikki tarvittavat kuvat canvakselle.

Liikkeen animointi oli yksi tutkimukseni tärkeimmistä löytämistäni tekniikoista. Kokosin aineistoni pohjalta mallin, jonka avulla drawImage-metodiin liitetään animoinnin mahdollistavat muutamat tärkeät muuttujat. Tämän jälkeen kykyenin metodissa rajaamaan hahmoon liitettävän spritesheet-kuvan, aina hahmon liikkeen kannalta oikeasta kohdasta. Hahmoa varten tehdystä malliobjektista

luotiin yhteys animoitaviin objekteihin. Animointi hyödynsi useita JavaScriptille tyypillisiä ominaisuuksia, kuten muuttujia, funktioita, moduulia ja requestAnimationFrame”-metodin luomaa intervallia.

Kartan luominen useaan tasoon oli yksi tämän kappaleen tärkein pointti, joka mahdollisti syvyyden luomisen pelimaailmaan. Kartalle luotiin kaksi objektia ja kuvamuuttujaa, joille annettiin eriävät arvot ja jotka tuotiin eri syvyysasteille pelin hahmoon nähden. Täsmensin aineistossani myös karttajärjestelmän luomiseen tarvittavien taulukoiden rakennusprosessia, sillä karttojen luomisessa taulukoiden käyttö on ehdotonta. Loin törmäyskartalle mallin, joka hyödynsi muun muassa switch-lauseketta, taulukkoa ja vakiomuuttujia törmäysobjektien luomisessa ja kartalle sijoittelussa. Korostin myös lähdeaineistostani löytämääni törmäys-tekniikan luovaa hyödyntämistä vuorovaikutuksen luomisessa eri objektien välille.

Tutkielmani lopussa esitin tapoja, kuinka RPG-peliä voidaan helposti kehittää eteenpäin aineistosta löytämieni tekniikoiden avulla. Esineiden poimiminen on vain objektien törmäämistä toisiinsa, jolloin niiden arvoja ja paikkoja taulukoissa muutetaan. Tällä esimerkillä havainnollistin, kuinka vaikealta tuntuva käsite on vain joukko jo opittuja tekniikoita. Pelikartan vaihtaminen on hieman erilainen, mutta yhtä helposti lähestyttävä ongelma, joka voidaan ratkaista hieman mielikuvitusta käyttäen samankaltaisella ratkaisumallilla kuin esineiden poimiminen.

Käyttämäni tekniikat ja toimintamallit ovat täysin samankaltaisia esimerkiksi verkkosivujen, hybridisovellusten tai muiden peligenrejen pelien rakentamisessa, joten tutkimuksessa löytämäni tekniikat ovat hyödyllisiä monella tavalla myös tulevaisuudessa. Jos esimerkiksi aloitteleva pelinkehittäjä haluaa tutustua koodin kirjoittamiseen ja ohjelmointikielten tarjoamiin mahdollisuuksiin, ovat löytämäni mallit loistava ensimmäinen askel tähän maailmaan. Mallit tarjoavat myös helpon tavan rakentaa monialustainen esittelypeli, jota voidaan esitellä laajalla laitevalikoimalla tilaajalle suoraan verkon kautta. Ohjelmointikielillä tapahtuneesta pelinkehityksestä voi myös luonnollisesti loikata web-suunnittelun pariin tai päinvastoin.

7 POHDINTA

Tutkimuskysymyksessäni minun oli tarkoitus selvittää, miten HTML5 ja JavaScript -kieliä voidaan hyödyntää RPG-pelin tekemisessä. Lähestyin tutkimuskysymystäni hyvin teknisestä näkökulmasta käyden ensin läpi JavaScriptin ja HTML5:n käsitteitä ja tekniikoita, joihin viittasin ja joita hyödynsin myöhemmin rakentaessani varsinaista RPG-pelini koodia. Aloittaessani tutkimustani halusin löytää keinon matalan kynnyksen pelinkehitykseen ja uskoin HTML5:n ja JavaScriptin olevan ratkaisu tähän ongelmaan. Kyseiset tekniikat ovat hyvin yleisiä ja yleishyödyllisiä, mutta harva tulee ajatelleeksi, mihin kaikkeen näitä yleensä verkkosivujen rakentamiseen käytettyjä kieliä voi venyttää.

RPG-pelien luomiseen verkkotekniikoilla ei ole olemassa täsmällistä, yksiselitteistä ja helppoa ohjetta. Etenkin suomen kielellä jo pelkkä kirjallisuus verkkotekniikoilla suoritettavasta pelinkehityksestä on käytännössä olematonta. Tutkimuksessani löysin useita HTML5- ja JavaScript-tekniikoita, joita voi kierrättää ja käyttää useassa eri yhteydessä riippuen käsillä olevasta ongelmasta. Erityisesti JavaScriptiä on käytetty pelinkehityksessä yksinkertaisten pelien tekemisessä jo aiemmin, mutta tutkimuksessani löysin keinoja, joilla tämä ohjelmointikieli kyetään liittämään juuri RPG-peleihin.

Tutkimukseni oli laadullinen ja se piti sisällään JavaScriptillä rakennettavien pelien rakenteiden ymmärtämisen, niiden hyödyntämisen ja luovan uudelleen käytön. Tutkielmani rinnalla rakennettu produktio oli loistava tapa testata keräämäni materiaalin pohjalta luotuja koodin rakennustekniikoita, joita kykenin liittämään esimerkiksi hahmon liikkeen animointiin. Kokosin ainestoani hyväksi todetuista luovista ratkaisuista ja jatkoin päätelmien tekemistä yhä pidemmälle. Produktio oli siis olennainen osa oppimisprosessiani, sillä kykenin sen avulla havainnoimaan tekniikoiden tehokkuutta ja toimivuutta.

Teknisestä näkökulmasta löysin tapoja, kuinka muuttujia, funktioita ja erityisesti taulukoita ja silmukoita voidaan hyödyntää aina uudelleen eri yhteyksissä. Objektit, muuttujat ja funktiot olivat ratkaisevassa asemassa hahmoihin ja esineisiin liittyvissä ratkaisuissa. Hahmon luomisen yhteydessä luotiin kustomoitu objekti, jota käytettiin hyödyksi hahmon liikuttamisessa, piirtämisessä ja animoinnissa. Hahmo kyettiin liikkumisen yhteydessä animoimaan muutaman muuttujan, laskurin ja räätälöidyn drawImage-metodin avulla. Pelille luotiin hahmon lisäksi karttajärjestelmä rinnakkaisten taulukoiden avulla. Moniulotteinen kartta on voimakas visuaalinen lisä kaksiulotteisille peleille, jonka

avulla peliin voidaan luoda syvyyden tunnetta. Törmäyskartat mahdollistavat mielekkään liikkumisen kartoilla, kun taas objektikarttojen avulla peliin voidaan objektien sijoittamisen lisäksi tuoda RPG-peleille ominaista strategista näkökulmaa. Taulukot ja silmukat löysivät uusia käyttötarkoituksia erityisesti karttoja luodessani ja asioita sijoitellessani, kun taas ehtolauseet ja muuttujat olivat hyödyllisiä vuorovaikutuksen lisäämisessä eri objektien välillä. Vuorovaikutus on myös hyvin tärkeää kyseiselle peligenrelle. Törmäyksen tunnistus ja tämän hyödyntäminen muuttujien ja taulukoiden arvoissa on myös äärimmäisen tärkeä taito. Kyseisten tekniikoiden oppimisen jälkeen ilmensin, kuinka helposti niitä voi hyödyntää peliä eteenpäin kehittäessä.

Tavoitteeni tutkimusta aloittaessani oli luoda perusrakenne verkossa käytettäviin ohjelmointikieliin tutustuville, pelinkehityksestä kiinnostuneille henkilöille sekä tuoda esiin näiden kielien sisäisiä rakenteiden ja ominaisuuksien luovaa hyödyntämistä. Tärkein tutkimukseni tulos oli näin ollen tarkkojen teknisten kikkojen lisäksi ilmentää JavaScriptin monimuotoisuutta ja helppoutta ohjelmointikielenä. HTML5 ja JavaScript ovat aloittelevalle pelinkehityksestä kiinnostuneelle henkilölle todellinen matalan kynnyksen koodikieli. Erityisesti RPG-pelien luomisessa pelkästään näiden kielten hyödyntämisen osaaminen voi olla riittävä taito monimutkaisten pelien luomiselle. Kaikki esimerkit ja mallit, joita hyödynsin tutkimuksessani, olivat puhdasta koodia, eikä ulkoisia pelimoottoreita tai kirjastoja näin ollen tarvittu lainkaan. Tämä on tutkimuksessani mahdolliseksi todettu ja näin ollen myös uudelle pelinkehittäjälle hyvä uutinen.

LÄHTEET

Barton, M. 2007. The History of Computer Role-Playing Games Part 1: The Early Years (1980-1983). Gamasutra. Viitattu 8.12.2016,
<http://www.gamasutra.com/view/feature/132024/the_history_of_computer_.php>.

Burchard, E. 2013. The Web Game Developer's Cookbook. Addison-Wesley. Viitattu 17.02.2017, (ladattavissa maksullisena),
<https://www.amazon.com/Web-Game-Developers-Cookbook-JavaScript/dp/0321898389>.

Hirsjärvi, S., Remes, P. ja Sajavaara, P. 2009. Tutki ja kirjoita. 15. Uusittu painos. Helsinki: Tammi.

HTML Standard 2016. Is this HTML5?. Viitattu 8.12.2016,
<https://html.spec.whatwg.org/multipage/introduction.html#is-this-html5?>.

Korpela, J. 2014. HTML5-käsikirja. Jyväskylä: Docendo.

Lappalainen, H. 2012. Peliohjelmointi JavaScript-kirjastolla. Mikkelin ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö.
<http://www.theseus.fi/handle/10024/51563>.

Math is FUN. Two-Dimensional 2013. Viitattu 8.12.2016,
<http://www.mathsisfun.com/definitions/two-dimensional.html>.

Mozilla Developer Network. Tiles and tilemaps overview 2016. Viitattu 14.03.2017,
<https://developer.mozilla.org/en-US/docs/Games/Techniques/Tilemaps>.

Riippi J, 2013. Natiivi, hybridi ja HTML5. Vincit. Viitattu 8.12.2016,
<https://www.vincit.fi/en/blog/natiivi-hybridi-ja-html5/>.

Tukiainen T, 2013. Mobiilipelit ja pelimoottorit. Helsingin yliopisto. Tietojenkäsittelytieteiden koulutusohjelma. Pro gradu -tutkielma.
<https://helda.helsinki.fi/bitstream/handle/10138/42050/20131106graduv3.pdf?sequence=2>.

van de Spuy, R. 2012. Foundation Game Design with HTML5 and JavaScript. Viitattu 8.12.2016, <http://www.apress.com/us/book/9781430247166>.

W3 Schools. (Ei julkaisuvuotta). HTML Responsive Web Design. Viitattu 17.02.2017, <https://www.w3schools.com/html/html_responsive.asp>.

W3 Schools. (Ei julkaisuvuotta). Window innerWidth and innerHeight Properties. Viitattu 28.03.2017,
<https://www.w3schools.com/jsref/prop_win_innerheight.asp>.