Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Bisnesakatemia

2014

Aleksi Tossavainen

HTML5 PELISOVELLUKSEN KEHITTÄMINEN

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Bisnesakatemia

20.05.2014 | 42

Päivi Killström

Aleksi Tossavainen

HTML5 PELISOVELLUKSEN KEHITTÄMINEN

Tämän opinnäytetyön tavoitteena oli tutustua pelikehitykseen HTML5:n näkökulmasta .

Opinnäytetyössä ohjelmoitiin verkkoselaimessa toimiva yksinkertainen kaksiulotteinen peli. Samalla tutustuttiin pelien yksinkertaisten toimintojen rakentumiseen kuten liikkuminen, ampuminen, vihollisten tuhoutuminen ja pisteidenlasku.

Peli toteutettiin HTML5 ja JavaScript kielillä Adobe Dreamweaver -kehitysympäristössä. Pelin Grafiikat toteutettiin sprite-tekniikalla. Työn lopputuloksena syntyi yksinkertainen esimerkkipeli, jonka avulla käytettyjä tekniikoita havainnollistetaan. Peli on kyllä toimiva, mutta ei vielä riittävän mutkikas ja houkutteleva virallista julkaisua varten.

ASIASANAT:

HTML, JavaScript, sovelluskehittimet, mobiilipelit, mobiilisovellukset, verkko-ohjelmointi

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology| Business Academy

20.05.2014 | 42

Päivi Killström

Aleksi Tossavainen

DEVELOPING HTML5 GAME APPLICATION

The goal of this thesis was to gain an insight to game development with help of HTML5.

A simple 2-D game that works on a browser was programmed in this thesis. At the same time the focus was on getting familiar with the game's simple actions such as moving, shooting, destroying enemies and counting scores.

The game was designed with HTML5 and JavaScript languages in Adobe Dreamweaver environment. Game's graphics was done with sprite-technique. The final outcome of the process is a game that is working. However it is not yet completed and attractive enough to be officially released

KEYWORDS:

HTML, JavaScript, Application Development Systems, Mobile Games, Mobile Applications, Web-programming

SISÄLTÖ

SANASTO **6**

1 JOHDANTO **7**

2 MOBIILIPELIT, HTML5 JA ADOBE DREAMWEAVER **9**

2.1 Mobiilipelien kehitys 9

2.2 Pelimoottorit (game engine) 10

2.3 HTML5 12

2.3.1 HTML5:n syntaksi 12

2.3.2 Uudet elementit 13

2.3.3 Canvas-elementti 15

2.4 JavaScript 15

JavaScriptin syntaksi 17

2.5 Adobe Dreamweaver 18

3 ESIMERKKIPELIN OHJELMOINTI **19**

3.1 Pelin luominen Adobe Dreamweaverillä 19

3.2 Canvaksen luonti 21

3.3 Spritet 21

3.4 Aloitusvalikon ja taustakuvan luominen 22

3.4.1 Aloitusvalikko 22

3.4.2 Taustakuva ja pelin käynnistäminen 25

3.5 Päähenkilön luominen 27

3.6 Vihollisen luonti ja sen päätoiminnot 30

3.7 Vihollisten tuhoaminen ja pisteiden lasku 32

3.7.1 Vihollisten tuhoaminen 32

3.7.2 Pisteiden lasku 38

4 YHTEENVETO **41**

LÄHTEET **43**

LIITTEET

Liite 1. Pelin koodi.

KUVAT

Kuva 1. JavaScriptin versiot 16
Kuva 2. Hello World 17
Kuva 3. Uuden projektin luominen Dreamweaverilla. 19
Kuva 4. Uuden projektin luomisvalikko Dreamweaverissä. 20
Kuva 5. Spritejen luominen. 22
Kuva 6. Aloitusvalikko 23
Kuva 7. Valikon piirtäminen sprite-sivulta. 24

TAULUKOT

Taulukko 1 HTML5: uudet ominaisuudet 13

SANASTO

2D	kaksiulotteinen
3D	kolmiulotteinen
Adobe	Ohjelmistoalan yritys
Dreamweaver	Sovelluskehitystyökalu
HTML5	Ohjelmointikieli
JavaScript	Ohjelmointikieli
Sprite-grafiikka	Usein 2D-peleissä käytetty tietokonegrafiikan menetelmä, joka perustuu pieniin pikseligrafiikkana toteutettuihin kuvahahmoihin, joita kutsutaan spriteiksi (Wikipedia 2014)

1 JOHDANTO

Mobiilipelit ovat tulleet osaksi lähes jokaisen elämää. Ne ovat koukuttavia ja niitä pelataan pääosin puhelimilla tai tableteilla. Esimerkiksi rautatieasemilla ja muilla julkisilla paikoilla voi hyvin nähdä useiden ihmisten päiden olevan kyyryssä ja heidän puhelimien näytöillä voi nähdä vilauksen lentävästä linnusta tai räjähtävästä karkista. Myös tietokoneen selaimessa pelattavat mobiilipelit ovat kasvattaneet suosiotaan.

Mistä näiden pelien suosio sitten johtuu? Miten tai kuka niitä tekee? Mobiilipelit ovat yleensä hyvin yksinkertaisia ja sidoksissa johonkin sosiaaliseen mediaan, kuten Facebookiin. Omia pelisaavutuksiaan voi verrata Facebook-kavereiden kanssa ja heiltä voi pyytää myös apua, jotta pääsisi omassa pelissään eteenpäin. Pelien yksinkertaisuus ja sosiaalisen median kytkökset lisäävät pelien viehättävyyttä ja saavat käyttäjän palaamaan yhä uudelleen pelin pariin.

Tämän opinnäytetyön tavoitteena on tutkia ja avata mobiilipelien toteuttamista HTML5:n näkökulmasta sekä luoda esimerkkipeli. Työssä esitellään HTML5:n tuomia uusia mahdollisuuksia sekä käydään esimerkkipelin pohjalta läpi miten selainpohjainen mobiilipeli toteutetaan käytännössä. Esimerkkipeli sisältää HTML5:n lisäksi JavaScript-ohjelmointia. Ohjelmointiympäristönä toimi Adoben Dreamweaver.

Idea HTML5-pelin tekemisestä tuli keväällä 2013 ja kesän 2013 ajan opiskelin itsenäisesti HTML5- ja JavaScript-ohjelmointia. Kerättyäni tarpeeksi tietoa kirjallisuudesta ja internetistä ohjelmoin esimerkkipelin syksyllä 2013.

Tässä opinnäytetyössä ohjelmoitu peli ei ole vielä valmis versio vaan kehitys jäi beta-asteelle. Pelin juoni on yksinkertainen: Sankarihahmo ampuu avaruudessa oikealta ilmestyviä vihollisia ja saa tästä pisteitä. Pelissä ei voi kuolla, eikä periaatteessa oikein voittaakaan. Pelissä on kuitenkin tyypillisen mobiilipelin pääpiirteet: sen juoni on yksinkertainen ja pelin grafiikka on kaksiulotteinen.

Opinnäytetyön alussa esittelen mobiilipelien kehitystä sekä kerron myös pelimoottoreista. Näiden jälkeen esitellään esimerkkipelissä käytetyt tekniikat ja välineet. Seuraavana on vuorossa esimerkkipelin ohjelmoinnin läpikäyminen, jossa selitetään yksityiskohtaisesti miten mikäkin pelin osio on tehty. Opinnäytetyön lopussa on vielä yhteenveto esimerkkipelin ohjelmoinnista.

2 MOBIILIPELIT, HTML5 JA ADOBE DREAMWEAVER

Mobiilipelien suosion kasvun myötä on pelien julkaisua varten markkinoilla useita kilpailevia käyttöjärjestelmiä. Näistä suurimmat ovat Applen iOS, Googlen Android ja Microsoftin Windows Phone. Jotta jokaiselle alustalle voitaisiin ohjelmoida oma natiivi pelisovellus, tulisi sama peli ohjelmoida kolmella eri ohjelmointikielellä: Apple käyttää iOS-alustassaan Objective C:tä, Google Androidissaan Javaa ja Microsoft Windows Phonessaan C#:a. Pelin julkaisijalle tämä tulisi erittäin työlääksi ja kalliiksi toimintatavaksi. Onneksi useimmat käyttöjärjestelmät tukevat avoimeen lähdekoodiin perustuvaa HTML5:ta. HTML5 on webin uusin standardi, jonka ohjelmointikielenä toimii JavaScript. Näin sama koodi toimii usealla eri alustalla samalla tavalla. Tässä luvussa kerrotaan yleisesti pelien kehityksestä, sekä esitellään HTML5:ta ja JavaScriptia tarkemmin.

2.1 Mobiilipelien kehitys

Mobiilipelien kehitys alkoi 1990-luvun lopulla, jolloin matkapuhelinyhtiö Nokia julkaisi ensimmäisen Snake-pelinsä, joka tunnetaan paremmin matopelinä. Tämä loi pohjan mobiilipelien menestymismahdollisuuksille. Vuonna 2004 Nokia julkaisi N-Gage-pelipuhelimen, jolla oli mahdollista pelata jo konsolitasoisia pelejä. N-Gage ei kuitenkaan menestynyt odotetusti ja sen valmistus lopetettiin (Immonen 2013).

Mobiilipelit kokivat seuraavan suuren mullistuksen, kun Apple julkaisi ensimmäisen iPhonen vuonna 2007. iPhonessa oli App Store -sovelluskauppa, johon kenen tahansa oli mahdollista julkaista itse tekemiään pelejä. Tällöin ei enää tarvittu suuria pelitaloja pelien julkaisemista varten. Tämä sovellusten liiketoimintamalli oli aivan uutta alallaan ja loi alalle mobiilipelien julkaisutulvan (Immonen 2013). Myöhemmin myös Google ja muut valmistajat innostuivat sovelluskauppojen tuomista mahdollisuuksista ja näin uusi liiketoimintamalli oli syntynyt.

Monet suomalaiset mobiilipelialan yritykset, kuten Rovio ja Supercell, ovat luoneet suuren omaisuuden menestymällä omilla mobiilipeleillään. Rovion vuonna 2009 julkaisema Angry Birds oli App Storen myydyin sovellus ja Clash of Clans -pelillään menestystä niittänyt Supercell teki vuonna 2013 miljoonakaupat japanilaisten peliyritysten SoftBank ja GungHo kanssa. Nämä suurimmat suomalaiset esimerkit varmasti antavat lisää nostetta suomalaiselle pelikehitykselle ja uusia peliohjelmistoyrittäjiä syntyykin kovalla tahdilla. (Immonen 2013)

2.2 Pelimoottorit (game engine)

Mobiilipelien kehitystä varten on olemassa väliohjelmistoja nimeltä pelimoottorit. Pelimoottoreilla helpotetaan huomattavasti kehittäjän työtä, sillä niissä on pelien perustoiminnallisuudet ja rakenteet valmiiksi ohjelmoituina. Pelimoottoreilla vähennetään kehittäjän työmäärää ja samalla pelien laatua pystytään parantamaan. Pelimoottoreiden avulla on peliteollisuudessa pystytty lisäämään peliohjelmien välistä uudelleenkäyttöä, joka on mahdollistanut uusien pelien tuottamisen entistä tehokkaammin. Useimmat pelimoottorit eivät kuitenkaan mahdollista äänen tai grafiikan luontia. Tällöin on käytettävä ulkoisia äänen ja grafiikan tuottamiseen tarkoitettuja ohjelmia. Avoimen lähdekoodin ratkaisuja tarjoavat audioeditointiin esimerkiksi Audacity ja kuvankäsittelyyn Gimp. Maksullisia kaupallisia ohjelmia ovat esimerkiksi Adoben audioeditointiohjelma Audition ja kuvankäsittelyohjelma Photoshop (Tukiainen 2013, 26).

Mobiilipeleille suunnatuista pelimoottoreista esimerkkejä ovat ShiVa3D Game Engine, Marmalade, Unreal Engine sekä Unity. (Tukiainen 2013, 28) Mobiilipelien pelimoottoreille erityisiä piirteitä ovat niiden mahdollisuus tuottaa pelejä usealle eri alustalle. Useat kilpailevat alustat muodostavat isomman asiakasryhmän kuin vain yhdelle alustalle natiivisti toteutettu pelisovellus.

Lisäksi eri alustojen markkinaosuuksien kehittymistä on vaikea ennustaa. Jotta markkinariskiä saadaan pienennettyä, on pelin yhteensopivuus eri alustojen kanssa tärkeää.

Pelimoottorille tärkeä ominaisuus on myös lopullisen pelin asennuspaketin koko. Mobiilipelit ladataan usein mobiilialustan sovelluskaupasta asennustiedostona. Tämä asettaa tiettyjä rajoitteita, sillä sovellukset ladataan usein mobiilidatayhteyttä käyttäen ja osa sovelluskaupoista asettaa jaettavien sovellusten koolle ylärajan. Jotkin sovellukset eivät myöskään anna ladata yli 50 megatavun tiedostoja ilman lähiverkkoyhteyttä. Jotta kuluttaja saisi hyvän käyttäjäkokemuksen, on tärkeää saada sovellusten lataaminen ja päivittäminen mahdollisimman kevyeksi ja nopeaksi toiminnoksi (Tukiainen 2013, 28).

Tämän opinnäytetyön HTML5-esimerkkipelissä ei ole käytetty mitään ulkoista pelimoottoria tai muutakaan JavaScript-kirjastoa, sillä yksinkertaisesti ei ollut aikaa opetella tai perehtyä niiden käyttöön. Niitä on kuitenkin olemassa pelien tekemisen helpottamiseksi. Näistä tunnetuimpia ovat muun muassa (HTML5 Game Engine 2014)

- ImpactJS
- EaselJS
- Cocos2d-X.

Kuten muissakin pelimoottoreissa, on näillä JavaScript-kirjastoilla mahdollisuus helpommin ja nopeammin tehdä kohtuullisen monimutkaisia HTML5-pelejä.

2.3 HTML5

HTML5 on webin uusin standardi, joka on kehitetty sen edeltäjän HTML 4:n pohjalta. Se tuo mukanaan paljon uusia nykyaikaisia elementtejä, jotka mahdollistavat esimerkiksi videoiden, musiikin ja animaatioiden suorittamisen selaimessa ilman erillisiä asennettavia lisäosia. Lisäksi sitä voidaan käyttää myös monimutkaisten Web-sovellusten rakentamiseen. Se on myös suunniteltu toimimaan useiden eri laitteiden kanssa samalla tavalla, oli käytössä sitten tietokone, tabletti, älypuhelin tai älytelevisio (W3 Schools 2014).

2.3.1 HTML5:n syntaksi

Alla esitellään yksinkertainen HTML5-dokumentti, jossa on vähin määrä tageja, mitä sen luomiseen vaaditaan:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>

<body>
Content of the document......
</body>
</html>
```

(http://www.w3schools.com/html/html5_intro.asp)

”HTML5:n HTML-syntaksi tarvitsee doctype-määrityksen, jotta selain renderöi sivun standardien mukaisesti. XML-syntaksin kanssa doctypeä ei tarvitse määrittää. Doctypen määritys on <!doctype html>. Tämä eroaa aiemmista HTML-versioista siten, että se ei tarvitse lisäksi viittausta DTD:hen (Document Type Definition), koska ne perustuivat SGML:ään (Standard Generalized Markup Language)” (Aitta-Aho 2012, 3). Uusi määritys on huomattavasti yksinkertaisempi verrattuna aiempaan HTML4 -standardin mukaiseen doctype-määritykseen:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd"> (W3 Schools 2014)
```

2.3.2 Uudet elementit

Kuten aiemmin mainittiin, HTML5 tuo mukanaan uusia elementtejä. Taulukossa 1 on luettelo näistä elementeistä.

Taulukko 1 HTML5: uudet ominaisuudet (W3 Schools 2014, Aitta-Aho 2012, 4)

<canvas>	Määrittelee graafisen piirtämisen JavaScriptin avulla.
<audio>	Käytetään äänisisällön toistamiseen.
<video>	Käytetään videoiden tai elokuvien toistamiseen, sekä äänitiedostojen joissa on tekstitys.
<source>	Käytetään muiden mediaelementtien kanssa. Mahdollistaa vaihtoehtoisten medialähteiden käytön.
<track>	Käytetään muiden mediaelementtien kanssa. Mahdollistaa ulkoisten ajastettujen tekstien, esimerkiksi tekstityksien käytön.
<embed>	Käytetään ulkoisen sovelluksen tai interaktiivisen sisällön sisällyttämiseen.
<header>	Käytetään luomaan ylätunniste.
<nav>	Käytetään määrittämään sivulta alue, jossa on navigaatiolinkkejä sivun muihin osiin tai muille sivuille.
<section>	Käytetään määrittämään dokumentin tai sovelluksen osa.
<main>	Määrittää dokumentin pääsisällön.
<article>	Käytetään muun muassa sanomalehden artikkelin tai blogikirjoituksen määritykseen.
<aside>	Määrittää sisältöä joka liittyy muun sivun sisältöön, mutta on kuitenkin erillään tästä.
<footer>	Käytetään luomaan alatunniste.
<details>	Käytetään näyttämään/piilottamaan lisätietoja.

(jatkuu)

Taulukko 1 (jatkuu).

<summary>	Käytetään näyttämään otsikko <details> -elementille.
<figure>	Määrittää omavaraista sisältöä kuten koodilistauksia ja diagrammeja.
<figcaption>	Käytetään <figure> -elementin kanssa selittämään esimerkiksi kuvateksti.
<mark>	Käytetään korostamaan tekstiä.
<time>	Käytetään määrittämään aika koneluettavaan muotoon.
<bdi>	Käytetään eristämään tekstinosa kaksisuuntaista tekstiä varten.
<wbr>	Käytetään pitkien lauseiden kanssa jotta ne vaihtaisivat riviä oikein.
<dialog>	Määrittää dialogi laatikon tai ikkunan.
<meter>	Käytetään skaalautuvana mittarina kuvaamaan jonkun tietyn rajan puitteissa, esimerkiksi kuvaamaan levynkäyttöä.
<progress>	Käytetään kuvaamaan jonkin tehtävän etenemistä.
<ruby>	Käytetään Itä-Aasian annotaatioiden kirjoituksessa.
<rt>	Käytetään kirjainten selitykseen/ääntämiseen Itä-Aasian kirjoituksissa.
<rp>	Määrittää mitä näytetään, jos selain ei tue Itä-Aasian annotaatoita.
<datalist>	Määrittää esiasetukset valintalistaan.
<keygen>	Määrittää avainpari generaattori alueen. Käytetään lomakkeissa.
<output>	Määrittää laskutoimituksen lopputuloksen.

HTML5:n myötä poistuneet HTML elementit (W3 Schools 2014):

- <acronym>
- <applet>
- <basefont>
- <big>
-
- <dir>
- <font>
- <frame>

- <frameset>
- <noframes>
- <strike>
- <tt>

2.3.3 Canvas-elementti

HTML5 -sovelluskehityksen tärkein elementti on canvas-elementti. Se on yksi keskeisimpiä uusia ominaisuuksia ja tarjoaa ”piirtoalustan” graafisille esityksille. Itse elementti on hyvin yksinkertainen, mutta sen kiinnostavuutta lisää mahdollisuus käyttää graafisia piirtokäskyjä, muotoiluja ja vuorovaikutusta käyttäjän kanssa JavaScriptin avulla (Korpela, HTML5).

```
<canvas id="canvasBg" width="800" height="500"></canvas>
```

Canvas-elementin kutsuminen on erittäin helppoa. Ensin kutsutaan canvas-metodikutsulla ja määritetään sille esimerkiksi id, width ja height-attribuutit. Ellei canvakselle anneta mitään attribuutteja, sen oletuskoko on 300x150 pikseliä. Se, mitä canvaksen sisällä tapahtuu, määritellään JavaScript-koodissa, joka on linkitetty HTML-sivulle.

2.4 JavaScript

JavaScript on olio-orientoitunut ohjelmointikieli, jota käyttämällä verkkosivulle voidaan lisätä toimintoja. HTML5:ssa JavaScript toimii ohjelmointikielenä, jolla toiminnallisuudet ohjelmoidaan.

JavaScript kielen kehitti Netscape Communications Corporation vuonna 1995. Aluksi sitä kutsuttiin LiveScript-nimellä ja se esiteltiin Netscape Navigator 2.0 -selaimen julkaisun yhteydessä. Myöhemmin sen nimi vaihtui JavaScriptiksi sen läheisen Java-yhteyden vuoksi (Moncur 1999). Nimestään huolimatta JavaScriptilla ja Javalla ei kuitenkaan ole toistensa kanssa mitään tekemistä. Java on Sun Microsystemsin kehittämä kielien C ja C++ kaltainen sukulainen, jolla voi-

daan luoda täysiverisiä ohjelmia ja hallita kuluttajaelektroniikkalaitteita(Smith & Negrino 2007, 3). Toisin kuin monet muut olio-orientoituneet ohjelmointikielet, JavaScript käyttää olioiden periytyvyyksiin luokkien sijasta prototyyppejä. JavaScript ei kärsi turhauttavista viiveistä, jotka liittyvät palvelinpuolen ohjelmointikieliin kuten PHP, joka perustuu selaimen ja palvelimen väliseen yhteyden pitoon (McFarland 2012).

JavaScriptilla internetsivuille voidaan luoda muun muassa seuraavia toiminnallisuuksia (Moncur 1999):

- Vierivän tai muuttuvan viestin liittäminen selaimen tilariville
- Lomakkeen sisällön tarkastaminen ja laskelmien tekeminen (Esimerkiksi tilauslomakkeessa päivittyvä kokonaissumma kun uusi tuote lisätään tilaukseen)
- Viestien välittäminen käyttäjille (Web –sivujen tai varoituslaatikoiden avulla)
- Animaatioiden luominen
- Selainversion ja selainohjelman määrittäminen ja ohjaaminen käytettävää selainta varten tehdylle Web –sivulle
- Selaimen käytössä olevien Plug-In –liitännäisohjelmien tarkistaminen ja käyttäjälle vaadituista selainliitännäisistä kertominen

Version	Release date	Equivalent to	Netscape Navigator	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	March 1996		2.0		3.0			
1.1	August 1996		3.0					
1.2	June 1997		4.0-4.05			3[102]		
1.3	October 1998	ECMA-262 1st + 2nd edition	4.06-4.7x		4.0	5[103]		
1.4			Netscape Server			6		
1.5	November 2000	ECMA-262 3rd edition	6.0	1.0	5.5 (JScript 5.5), 6 (JScript 5.6), 7 (JScript 5.7), 8 (JScript 5.8)	7.0	3.0-5	1.0-10.0.666
1.6	November 2005	1.5 + array extras + array and string generics + E4X		1.5				
1.7	October 2006	1.6 + Pythonic generators + iterators + let		2.0				28.0.1500.95
1.8	June 2008	1.7 + generator expressions + expression closures		3.0		11.50		
1.8.1		1.8 + native JSON support + minor updates		3.5				
1.8.2	June 22, 2009	1.8.1 + minor updates		3.6				
1.8.5	July 27, 2010	1.8.2 + ECMAScript 5 compliance		4	9	11.60		
1.8.6[*dubious – discuss*]				17				

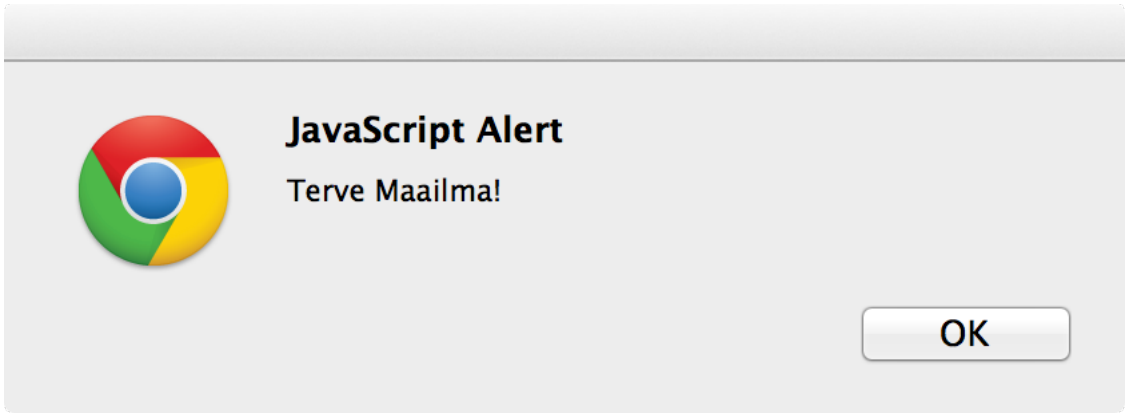
Kuva 1. JavaScriptin versiot (Wikipedia 2014.)

Kuvassa 1 on esitettynä englanninkielisestä Wikipediasta poimittu taulukko, joka esitttää javascriptin versiohistorian taulukkona.

JavaScriptin syntaksi

JavaScript koodi kirjoitetaan tagien <script> ja </script> väliin. JavaScriptia voidaan lisätä HTML:n sekaan, mutta isommissa sovelluksissa JavaScript usein linkitetään HTML-sivulle. Hello World-sovellus JavaScriptillä näyttää tältä:

```
<! doctype html>
<html>
    <head>

    </head>

    <body>

        <script>
            alert ('Terve Maailma!');
        </script>

    </body>

</html>
```



Kuva 2. Hello World

Kuvassa kaksi on Hello World sovellus. Selaimessa sovellus tulostaa Alert laatikon, jossa lukee ”Terve Maailma!”.

2.5 Adobe Dreamweaver

Adobe Dreamweaver on alunperin ohjelmistoyritys Macromedia:n kehittämä työkalu Web-sovellusten kehittämiseen. Adobe osti Macromedia:n vuonna 2005 ja samalla sovelluksen nimi vaihtui nykyiseen muotoon (Wikipedia 2014). Dreamweaver tukee seuraavia ohjelmointikieliä (Adobe 2012):

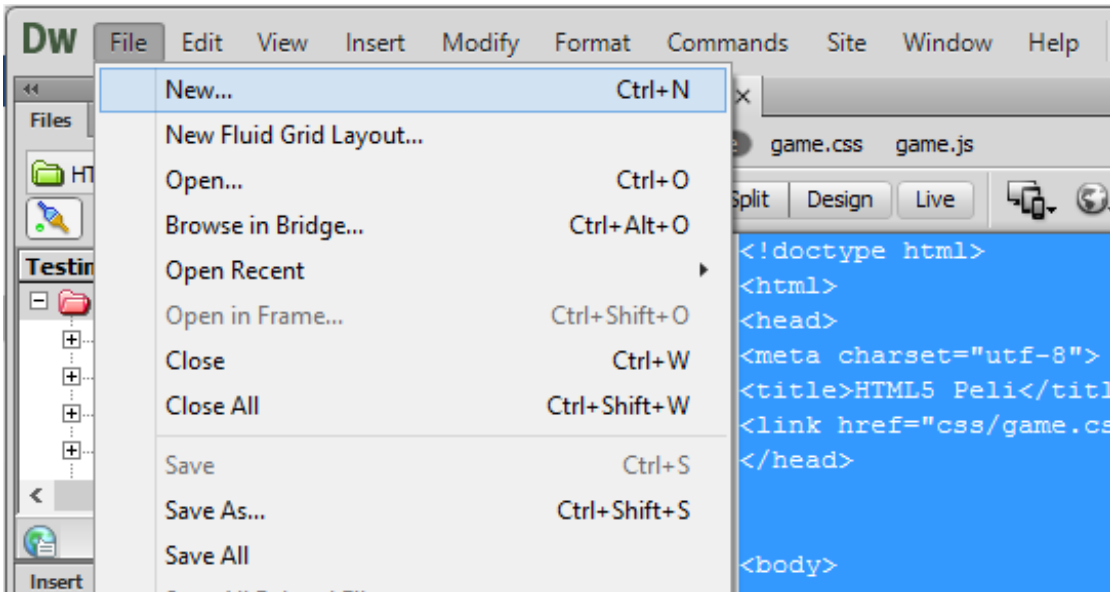
- HTML
- XHTML
- CSS
- JavaScript
- ColdFusion Markup Language (CFML)
- VBScript (for ASP)
- C# and Visual Basic (for ASP.NET)
- JSP
- PHP

3 ESIMERKKIPELIN OHJELMOINTI

Tätä opinnäytetyötä varten ohjelmoin esimerkkityön Adoben Dreamweaver CS6-ohjelmaa avuksi käyttäen. Pelissä on aluksi aloitusvalikko, josta nappia painamalla peli lähtee käyntiin. Pelissä ohjataan nuoli –tai W,A,S,D -näppäimillä sankarihahmoa, joka tuhoaa oikealta tulevia vihollisia heittelemällä niitä pullon-muotoisilla esineillä. Jokaisesta viholliseesta kertyy pisteidenlaskuun 25 pistettä. Peliä ei varsinaisesti voi voittaa eikä hävitä, sillä peli on vielä beta-vaiheessa eikä siis vielä täysin valmis.

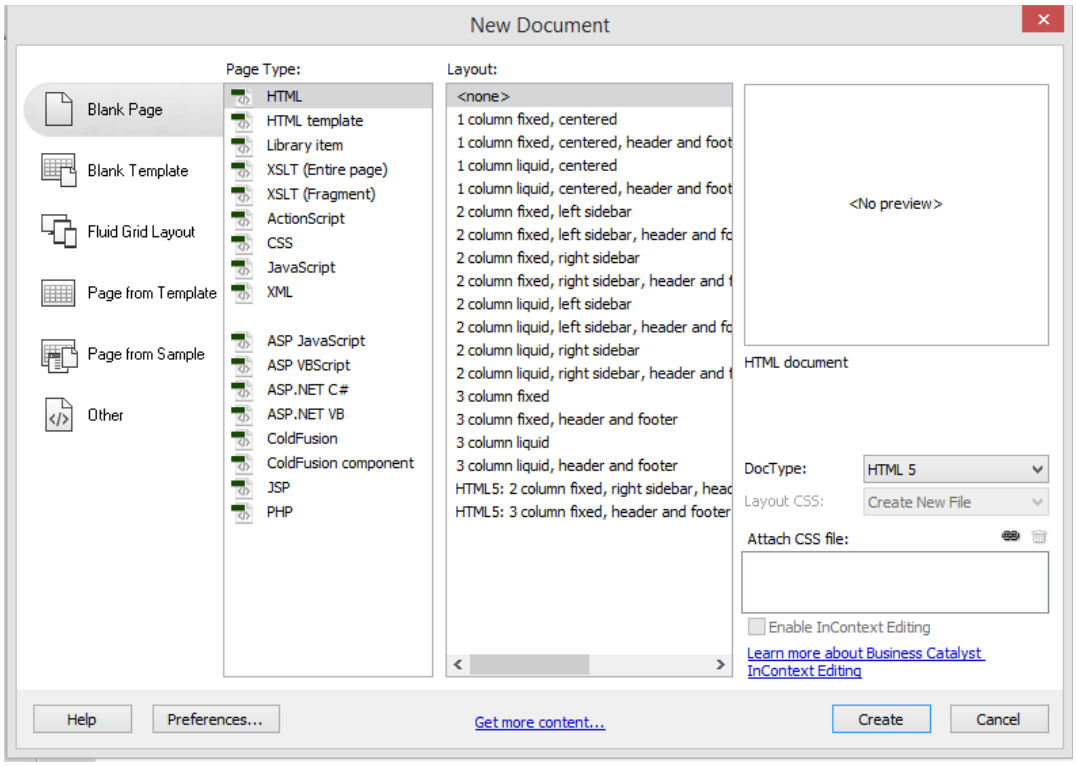
3.1 Pelin luominen Adobe Dreamweaverillä

Tässä luvussa kerrotaan, miten HTML5 peli luotiin Adobe Dreamweaver CS6 ympäristössä.



Kuva 3. Uuden projektin luominen Dreamweaverilla.

Kuva kolme havainnollistaa kuinka uusi Dreamweaver -projekti luodaan. Ensin painetaan ”file” ja sen jälkeen ”new”, jonka jälkeen aukeaa kuvan neljä mukainen näkymä.



Kuva 4. Uuden projektin luomisvalikko Dreamweaverissä.

Kuvan 4 mukaisesta valikosta voidaan valita, minkä tyylinen projekti halutaan luoda: HTML, PHP, ASP.NET tai joku muu vaihtoehto. Tässä esimerkkiprojektissa loin ensiksi HTML 5 sivun, jota valitessa valittiin HTML ja ”doctype” kohdasta piti muistaa valita HTML5 vaihtoehto. Tämän jälkeen eteen aukesi tyhjä HTML5 sivun rakenne, johon piti luoda canvas -elementit, sekä myöhemmin lisättävät javascript -ja CSS tiedostojen polut.

3.2 Canvaksen luonti

HTML5 pelikehitys perustuu lähes pelkästään uuden canvas-elementin ympärille. Tätä varten loin siis omat canvakset taustakuvalle, sankarihahmolle, viholliselle ja pisteenlaskulle. Canvas elementin ohjelmointi tapahtuu seuraavasti:

```
<canvas id="canvasBg" width="800" height="500"></canvas>
<canvas id="canvasEnemy" width="800" height="500"></canvas>
<canvas id="canvasHero" width="800" height="500"></canvas>
<canvas id="canvasScore" width="800" height="500"></canvas>
```

Jokaiselle pelin objektille eli taustakuvalle, viholliselle, sankarille ja pisteidenlaskulle luodaan oma id, joka kertoo mitä canvasta määritellään. Tässä esimerkissä jokaiselle canvakselle on määritetty sama leveys ja korkeus 800 x 500 pikseliä.

3.3 Spritet

Koska peli toteutetaan yksinkertaisella 2d-grafiikalla, on luontevaa käyttää niin kutsuttua sprite-tekniikkaa. Spritet ovat taustakuvattomia kuvia. Ne koostuvat pikseligrafiikalla toteutetuista hahmoista, jotka sijaitsevat sprite-kuvasivulla.

Kuva 5. Spritejen luominen.

Kuva 5 havainnollistaa, kuinka pelin taustakuva, sankarihahmo, vihollinen ja ammus sijaitsevat samassa kuvatiedostossa. Kuvasta voi myös huomata, että sprite -sivulla ei ole taustaväriä. Jos taustaväri olisi esimerkiksi valkoinen, hahmot eivät näyttäisi niin aidoilta, vaan lentäviltä neliöitä, joissa on kuva. Photoshopilla spriteja luodessa on myös hyvä lisätä PSD, eli Photoshop-dokumentti, tiedostoon kuva 3:n mukaiset apuviivat, jotka auttavat spriten koordinaattien löytämisessä kuvan piirtovaiheessa. Kuvan piirtämisestä kerrotaan luvussa 3.4.1.

3.4 Aloitusvalikon ja taustakuvan luominen

3.4.1 Aloitusvalikko

Pelin alkaessa ruutuun ilmestyy ensin kuvan kuusi mukainen aloitusvalikko, joka piirtyy canvakselle erillisestä sprite-kuvatiedostosta. Tässä kappaleessa esitetään miten, JavaScript koodilla voidaan piirtää canvakselle sprite-grafiikkaa.

Kuva 6. Aloitusvalikko

```
var canvasBg = document.getElementById('canvasBg');
var ctxBg = canvasBg.getContext('2d');//määrittelee missä for-
maatissa canvasta manipuloidaan (2d tai 3d)
```

Yläpuolella olevassa koodin pätkässä luodaan kaksi muuttujaa, joille annetaan metodit.

Muuttujalle ”canvasBg” annettu metodi palauttaa canvas-elementin, joka vastaa kysyttyä id-attribuuttia. Tässä tapauksessa id-attribuutti on luotu muuttujan kanssa samannimisiksi. Muuttuja ”ctxBg” kertoo canvaksen kontekstille manipuloitavan kuvaformaatin olevan 2d.

```
function init(){
    spawnEnemy(spawnAmount);
    drawMenu ();
    document.addEventListener('click',mouseClicked,false);
}
```

Yläpuolella oleva funktio nimeltä ”init” on pelin ensimmäinen funktio ja käynnistää koko pelin käynnistä-valikon. Funktio ”init” kutsuu useita funktioita, joista

”spawnEnemy” ampuu vihollisia ”spawnAmount”-muuttujan määrittämän määrän. Funktio ”drawMenu” piirtää pelin päävalikon, jota tutkimme seuraavaksi.

Kuva 7. Valikon piirtäminen sprite-sivulta.

```
var imgSprite = new Image();
var gameWidth = canvasBg.width;
var gameHeight = canvasBg.height;

imgSprite.src = 'images/sprites.png';

function drawMenu (){
    var srcX = 0 ; //spriten aloituspiste x-akselilla
    var srcY = 650 ; //spriten aloituspiste y-akselilla
    var drawX = 0 ; //piirron aloituspiste x
    var drawY = 0 ; //piirron aloituspiste y
    ctxBg.drawImage(imgSprite,srcX,srcY,gameWidth,gameHeight,dra
wX,drawY,gameWidth,gameHeight); //piirtää menun
}
```

Yllä oleva koodi piirtää päävalikon. Kuva 7 selventää, miltä sprite-sivu näyttää. Muuttujat ”gameWidth” ja ”gameHeight” saavat arvoksi canvaksen leveys- ja korkeusattribuutit. Itse ”drawMenu” funktion pystyisi luomaan lyhyemmin, mutta

selvennyksen vuoksi nimetään koordinaateille muuttujat, jolloin ”drawImage” metodin sulkujen sisällä olevat määritelmät ovat helpommin ymmärrettävissä.

Taustakuvan kontekstille annetaan metodi ”drawImage”. Muuttuja ”imgSprite” antaa sprites.png kuvan polun. ”srcX” antaa kuvan x-akselin, eli vaakasuuntaisen aloituspisteen ja vastaavasti ”srcY” y-akselin, eli pystysuuntaisen aloituspisteen ”sprites.png” kuvatiedostossa. ”gameWidth” kertoo kuvan leveyden ja ”gameHeight” kuvan korkeuden.

”drawX” kertoo kuvan x-akselin aloituspisteeksi canvaksella 0 ja vastaavasti ”drawY” y-akselin aloituspisteeksi 0. Kuten src-muuttujien kohdalla, pelin piirtoalue on yhtä leveä ja korkea, eli ”gameWidth” ja ”gameHeight”.

3.4.2 Taustakuva ja pelin käynnistäminen

```
function init(){
    spawnEnemy(spawnAmount);
    drawMenu ();
    document.addEventListener('click',mouseClicked,false);
}
```

Yllä oleva funktio on siis pelin ensimmäinen läpi käytävä funktio, joka alkaa lähettää vihollisia ja piirtää menun. Tämän lisäksi funktio käy läpi ”addEventListener”-metodin, joka käynnistyy ”click”-metodista. Lisäksi funktio kutsuu ”mouseClicked”-funktiota ja on oletuksena pois päältä, eli false.

```
function mouseClicked(e){
    mouseX = e.pageX - canvasBg.offsetLeft;
    mouseY = e.pageY - canvasBg.offsetTop;
    if(btnPlay.checkClicked())playGame();
          }
```

“mouseClicked”-funktio sisältää event-parametrin, joka on lyhennetty ”e”-kirjaimeksi. Se sisältää kaksi aiemmin määriteltyä muuttujaa mouseX – ja mouseY, jotka ”tarkastavat”, että event tapahtuu määritellyn canvas-elementin koordinaattien sisällä. Viimeisenä asetetaan ”if”-ehto, joka kysyy ”checkClicked”-funktiolta onko nappia painettu. Jos on, käynnistyy ”playGame”-funktio.

```
var btnPlay = new Button(196,611,208,339);

//button objekti

function Button(xL, xR, yT, yB){
    this.xLeft = xL;
    this.xRight = xR;
    this.yTop = yT;
    this.yBottom = yB;
    }

Button.prototype.checkClicked = function(){
    if (this.xLeft <= mouseX && mouseX <= this.xRight &&
this.yTop <= mouseY && mouseY <= this.yBottom) return true;
};

// button objekti loppu
```

Yläpuolella kerrotaan “Button”, eli käynnistysnappi-objektille määrittelyt. ”Button”-objektille annetaan neljä parametria, eli muuttujaa, joita käytetään funktion sisällä. Aivan ylimpänä luodaan muuttuja btnPlay. ”btnPlay”-muuttuja on uusi objekti, nimeltään ”Button”, jonka sulkujen sisällä on annettu napin x-y-koordinaatit pikseleinä. Objektifunktio ”Button”:n sulkeissa olevat parametrit tai argumentit saavat samat arvot kuin ylimmäisenä luotu ”btnPlay” muuttuja.

”Button”-objektimuuttuja konstruktoi neljä uutta muuttujaa, jotka kutsuttaessa luovat instanssin objektista ”Button”.

Lopuksi ”checkClicked”-funtio käy läpi if-lauseen avulla onko hiiren klikkaus tapahtunut napin koordinaattien sisällä. Jos on, se palauttaa arvon ”true”.

```
function playGame(){
    drawBg();
    startLoop();
    updateScore();
    document.addEventListener('keydown',checkKeyDown,false);
    document.addEventListener('keyup',checkKeyUp,false);
          }
function drawBg (){
    var srcX = 0 ; //spriten aloituspiste x-akselilla
    var srcY = 0 ; //spriten aloituspiste y-akselilla
    var drawX = 0 ; //piirron aloituspiste x
    var drawY = 0 ; //piirron aloituspiste y
```

```
ctxBg.drawImage(imgSprite,srcX,srcY,gameWidth,gameHeight,drawX,d
rawY,gameWidth,gameHeight); //piirtää taustakuvan
}
```

”playGame”-funktio kutsuu monia pelin käynnistämiseen liittyviä funktioita kuten ”drawBg”, joka piirtää pelin taustakuvan.

Kuten päävalikon piirrossa, logiikka on samanlainen. Taustakuvan kontekstille annetaan metodi ”drawImage”. ”imgSprite” on muuttujan nimi, joka antaa sprites.png kuvan polun. ”srcX” antaa kuvan x-akselin, eli vaakasuuntaisen aloituspisteen ja vastaavasti ”srcY” y-akselin, eli pystysuuntaisen aloituspisteen ”sprites.png” kuvatiedostossa. ”gameWidth” kertoo kuvan leveyden ja ”gameHeight” kuvan korkeuden.

3.5 Päähenkilön luominen

Verrattuna aloitusvalikkoon ja taustakuvaan, on pelin päähenkilöllä huomattavasti enemmän ominaisuuksia, kuten nuolinäppäimillä ohjaaminen ja vihollisten ampuminen. Tässä luvussa kerrotaan päähenkilöntoiminnot perusteellisesti.

```
var canvasHero = document.getElementById('canvasHero');
var ctxHero = canvasHero.getContext('2d');//määrittelee missä
formaatissa canvasta manipuloidaan (2d tai 3d)

var hero1 = new Hero;
```

Ensiksi luodaan muutama muuttuja. ”canvasHero” hakee html elementin nimeltään ’canvasHero’ muuttujan arvoksi. ”ctxHero” kertoo canvaksen kontekstille manipuloitavan kuvaformaatin olevan 2d. Muuttuja ”hero1” luo uuden objektin ”Hero”.

```
function Hero(){
    this.srcX = 0;
    this.srcY = 500;
    this.width = 150;
    this.height = 116;
    this.speed = 2;
    this.drawX = 150;
    this.drawY = 116;
```

```
    this.handX =  this.drawX +140;
    this.handY = this.drawY +20;
    this.isUpKey = false;
    this.isRightKey = false;
    this.isDownKey = false;
    this.iLeftKey = false;
    this.isSpacebar = false;
    this.isShooting = false;
    this.bullets =[];
    this.currentBullet = 0;
    for (var i = 0; i < 35; i++){
          this.bullets[this.bullets.length] = new Bullet(this);

          }
    this.score = 0;
          }
```

“Hero”-objektilla on useita ominaisuuksia, joiden muuttujan edessä on ”this”-avainsana. Tämä tarkoittaa, että muuttuja viittaa nykyiseen objektiin, eikä esimerkiksi objektiin ”Enemy”. Koska JavaScript on objektiorientoitunut kieli, se ei perustu luokkiin, kuten esimerkiksi C#. JavaScript perustuu prototyyppeihin, eli uusia objekteja luodaan kopioimalla ja laajentamalla vanhoja.

”Hero”-objektin ominaisuuksia ovat: spriten alkupisteiden koordinaatit sprite-sivulla, objektin leveys ja korkeus, liikkumisnopeus, piirron aloituspisteet, käsien koordinaatit (joista bulletit lähtevät), heron rajat, ohjaukseen liittyvät näppäininformaatiot, ampuminen, luotitaulukkko, ”for”-toistorakenne luotien piirtoa varten sekä aloituspisteluvuksi annettu arvo 0.

```
function checkKeyDown(e){
          var keyID = e.keyCode || e.which; //jos selaimesta ei
löydy e.keycode standardia käytetään e.which
          if (keyID === 38 || keyID === 87){ //38= ylänuolinäp-
päin tai 87 = W-näppäin
                    hero1.isUpKey = true;
                    e.preventDefault(); //estää nuolinäppäimiä
scrollaamasta websivua ylös ja alas kesken pelin
                                   }
```

Yllä oleva funktio tarkistaa onko näppäin painettu alas. Kuten olen koodissa kommentoinutkin, ”keyID”- muuttujan arvoksi annetaan tapahtuma ”keyCode” tai ”which” riippuen kumpaa metodia käytettävä selain tukee.

Jos tietokoneen näppäimistössä painetaan ylänuoli – tai W-näppäintä, piirtyvän sankarihahmon ”isUpKey”-arvoksi tulee tosi.

```
function checkKeyUp(e){
          var keyID = e.keyCode || e.which; //kertoo yleisimmil-
le eri selamille mitä nappia painetaan
          if (keyID === 38 || keyID === 87){ //38= ylänuolinäp-
päin tai 87 = W-näppäin
                    hero1.isUpKey = false;
                    e.preventDefault(); //estää nuolinäppäimiä
scrollaamasta websivua ylös ja alas kesken pelin
                                   }
```

Vastaavasti jos ylänuolinäppäin ei ole painettuna tulee ”isUpKey” muuttujan arvoksi false.

```
Hero.prototype.draw = function(){
    clearCtxHero(); // estää piirtämästä sankaria kokoajan  uu-
delleen
    this.checkDirection();
    this.handX = this.drawX +140;
    this.handY = this.drawY +20;
    this.checkShooting();
    this.drawAllBullets();
    ctxHe-
ro.drawImage(imgSprite,this.srcX,this.srcY,this.width,this.heigh
t,this.drawX,this.drawY,this.width,this.height); //piirtää
sankarin
};
```

Yläpuolella on “Hero”-objektin “draw”-funktio. Koska JavaScript ei perustu luokkiin vaan prototyyppeihin, luodaan sankarille uusi prototyyppi, eli aiemmin luodulle objektille luodaan prototyyppi ja annetaan sille periytyvä ominaisuus ”draw”.

Funktion ominaisuuksia ovat hahmon kontekstin poistaminen ennen uudelleen piirtämistä, jotta peli ei täyttyisi sankarihahmoista kun hahmoa liikutetaan. updateCoordinates-funktio päivittää objektin koordinaatit. ”checkDirection” tarkastaa, että objekti pysyy canvaksen rajojen sisällä. Lisäksi ominaisuuksiin kuuluu käden koordinaatit, ampuminen, luodit sekä hahmon piirtyminen.

```
Hero.prototype.updateCoordinates = function () {
    this.handX =  this.drawX +140;
    this.handY = this.drawY +20;
    this.leftX = this.drawX;
    this.rightX = this.drawX + this.width;
    this.topY = this.drawY;
    this.bottomY = this.drawY + this.height;
}
```

Yläpuolella on ”Hero”-objektin protoyyppifunktio, joka päivittää hahmon koordinaatit canvaksella.

```
Hero.prototype.checkDirection = function(){
    if (this.isUpKey && this.topY > 0) {
        this.drawY -= this.speed;
    }
    if (this.isRightKey && this.rightX < gameWidth) {
        this.drawX += this.speed;
    }
    if (this.isDownKey && this.bottomY < gameHeight) {
        this.drawY += this.speed;
    }
    if (this.isLeftKey && this.leftX > 0) {
        this.drawX -= this.speed;
    }

};
```

”checkDirection”-prototyyppifunktio antaa hahmon liikkua, jos se sijaitsee canvaksen reunojen sisäpuolella. Jos suuntaa osoittava näppäin on pohjassa ja x- tai y-akselin arvo on suurempi kuin nolla, siirtyy hahmo nuolen osoittamaan suuntaan objektin funktion sisällä ilmoitetulla nopeudella.

```
function clearCtxHero(){
    ctxHero.clearRect(0,0,gameWidth,gameHeight);
          }
```

Lopuksi ”clearCtxHero”-funktio tyhjentää sankarin kontekstin. Näin pystytään puhdistamaan sankari pois canvakselta ja seuraavan canvas-piirron aikana ei näytölle ilmesty montaa sankaria yhtenä vanana.

3.6 Vihollisen luonti ja sen päätoiminnot

Kuten päähenkilö, myös pelin vihollinen piirtyy erilliseltä sprite-sivulta. Viholliselle ei ole luotu niin paljon toimintoja kuin päähenkilölle. Tärkeimpinä ovat satunnaisesti uudelleen ”spawnautuminen”, eli ilmestyminen sekä vihollisen nopeus ja tuhoutuminen, ja mitä tuhoutumisen jälkeen tapahtuu. Tässä luvussa avataankin vihollisen ominaisuuksia syvemmin.

```
function Enemy(){
    this.srcX = 150;
    this.srcY = 500;
    this.width = 80;
    this.height = 130;
    this.speed = 2;
    this.drawX = Math.floor(Math.random() * 1000) + gameWidth;
// Math.floor lyhentää Math.random tuloksen alimpaan desimaaliin
ja antaa jonkin random luvun 0-800px välillä
    this.drawY = Math.floor(Math.random() * 360);
    this.rewardPoints = 25;
}
```

Kuten ”Hero”-objektilla, myös ”Enemy”-objektilla on x- ja y-akseleiden alkupisteet sprite-sivulla sekä leveys, korkeus ja nopeus. Lisäksi objektin piirron alkamiskohta on luotu satunnaiseksi JavaScriptin ”Math.floor(Math.random())”-metodilla. ”Enemy”-objektiin on myös määritelty vihollisen tuhoamisesta saatavaksi 25 pistettä, jotka lisääntyvät pistelaskuriin.

```
Enemy.prototype.draw = function(){
    this.drawX -= this.speed;// saa vihollisen tulemaan oikealta
vrt: this.drawX += this.speed, jolloin vihollinen tulisi vasem-
malta
    ctxEne-
my.drawImage(imgSprite,this.srcX,this.srcY,this.width,this.heigh
t,this.drawX,this.drawY,this.width,this.height); //piirtää vi-
hollisen
    this.checkEscaped();
};
```

Toisin kuin “Hero”:lla, ”Enemy”:lle ei tarvitse luoda kuin yksi liikkumissuunta, oikealta vasemmalle. Vihollisen piirto canvaksen kontekstille hoituu samaan tyyliin kuin muidenkin objektien.

```
Enemy.prototype.checkEscaped = function(){
    if (this.drawX + this.width <= 0){
          this.recycleEnemy();
          }
};
```

Vihollisobjektilla on myös prototyyppifunktio, joka tarkistaa onko vihollinen jäänyt tuhoutumatta ja päässyt karkuun. Käytännössä tämä toimii if-lauseella, joka katsoo jos vihollisen piirtopiste + vihollisen leveys pikseleinä, on yhteensä pienempi kuin 0, suoritetaan ”recycleEnemy”-funktio.

```
Enemy.prototype.recycleEnemy = function(){
    this.drawX = Math.floor(Math.random() * 1000) + gameWidth;
// Math.floor lyhentää Math.random tuloksen alimpaan desimaaliin
ja antaa jonkin random luvun 0-800px välillä
    this.drawY = Math.floor(Math.random() * 360);
};
```

Vihollisen kierrätys funktio suorittaa vastaavanlaisen vihollisen ”spawnaamisen”, kuin ”Enemy”-objektifunktio, eli luo satunnaisen uudelleen ilmestymispisteen.

```
function clearCtxEnemy(){
    ctxEnemy.clearRect(0,0,gameWidth,gameHeight);
          }
```

”clearCtxenemy”- tyhjentää vihollisen kontekstin jokaisen piirtokerran jälkeen ja näin estää vihollisen piirtymisen usean kertaan peräkkäin, kuten Hero-objektissakin.

3.7 Vihollisten tuhoaminen ja pisteiden lasku

3.7.1 Vihollisten tuhoaminen

Vihollisten tuhoaminen on yksi pelin tärkeimmistä ominaisuuksista. Tässä luvussa käydään läpi, miten tuo kaikki toimii.

```
function Hero(){
this.bullets =[];
this.currentBullet = 0;

for (var i = 0; i < 35; i++){
          this.bullets[this.bullets.length] = new Bullet(this);

          }
```

Kuten aiemmin ”Hero”-objektin ominaisuuksien käsittelyssä kävi ilmi, sillä on for-toistorakenne luotien piirtoa varten. Toistorakenteessa luodaan muuttuja ”i”, jonka arvo on nolla. Silmukassa luodaan uusia luoteja maksimissaan 35 kappaletta. Tämän määrän ylitettyä ei enää uusia luoteja synny.

```
Hero.prototype.drawAllBullets = function(){
```

```
    for (var i = 0; i < this.bullets.length; i++){
          if(this.bullets[i].drawX >=0)
this.bullets[i].draw();// piirtää kaikki bulletit joiden drawX
arvo >0
          if (this.bullets[i].explosion.hasHit)
this.bullets[i].explosion.draw();
          }
          };
```

”Hero”-objektille luodaan prototyyppi ”drawAllBullets”. Ensin for-toistorakenteeseen asetetaan ehto, jossa muuttuja ”i” on 0. Luotien määrä lisääntyy toistorakenteen mukaan aina yhdellä niin kauan kuin ”i”-muuttujan arvo on pienempi kuin ”bullets”-taulukon pituus. Tässä tapauksessa luoteja voi olla näkyvissä maksimissaan 35 kappaletta. Tämän jälkeen luodaan ”if”-lause. Jos luotien ”drawX”-arvo on suurempi tai yhtä suuri kuin nolla, kutsutaan kyseisen luodin piirtofunktiota ja luoti piirtyy näkyviin. Tämän jälkeen suoritetaan vielä if-lause , joka käy läpi kyseisen luodin funktiot ”explosion” ja ”hasHit”. Jos ne toteutuvat, ”explosion”- ja ”draw”-funktiot piirtävät peliin räjähdyksen.

```
Hero.prototype.checkShooting = function(){
    if (this.isSpacebar && !this.isShooting){ //!this.isShooting
on sama kuin this.isShooting === false
          this.isShooting = true;
          this.bullets[this.currentBullet].fire(this.handX,
this.handY);
          this.currentBullet++;
          if(this.currentBullet >= this.bullets.length)
this.currentBullet = 0;
          }else if (!this.isSpacebar){ //!this.isSpacebar on sa-
ma kuin this.isSpacebar === false
                    this.isShooting = false;
                    }
          };
```

Yläpuolella “Hero”-objektin prototyyppifunktio “checkShooting” suorittaa ensimmäisenä if-lauseen, joka tarkastaa, onko “isSpacebar”-funktio tosi ja “isShooting” epätosi. Jos ehto toteutuu, muuttuu ”Hero”-objektin ”isShooting” todeksi. Seuraavaksi nykyinen luoti lähtee liikkeelle ”fire”-funktiolla nykyisen objektin koordinaateista, jotka on määritetty muuttujille ”handX” ja ”handY”. Tämän lisäksi nykyisen luodin muuttujan ”currentBullet” -arvo päivittyy yhdellä seuraavaan. Tämän jälkeen seuraa if-lause jossa tarkastetaan, onko nykyinen luoti suurempi tai yhtä suuri kuin ”bullets”-listan pituus eli 35. Jos on, alkaa lista alus-

ta ”currentBullet”-muuttujan arvon ollessa taas nolla. Viitaten ensimmäiseen if-lauseeseen, on seuraavaksi vuorossa else if-ehtolause. Yksinkertaisuudessaan jos nykyisen objektin muuttuja ”isSpacebar” on epätosi, muuttuu myös nykyisen objektin ”isShooting” epätodeksi ja tällöin ammunta lakkaa.

```
//Bullet funktiot

function Bullet(h){
    this.hero = h;
    this.srcX = 230;
    this.srcY = 500;
    this.drawX = -20;
    this.drawY = 0;
    this.width = 6;
    this.height = 18;
    this.explosion = new Explosion();
    }
```

Ensin luodaan ”bullet”-objekti, jossa on parametrina kirjain ”h”, jota ”hero”-muuttuja referoi. Tämä on sitä varten, että luoti tietää, mille sankarille tämä kyseinen luoti kuuluu. Kuten “Hero”-objektissa on kyseinen for-toistorakenne:

```
for (var i = 0; i < 35; i++){
          this.bullets[this.bullets.length] = new Bullet(this);

          }
```

Toistorakenteesta löytyy kohta `new Bullet(this)`. Näin kyseinen luoti ja kyseinen sankari linkittyvät toisiinsa. Tämä on tärkeä ominaisuus pisteiden laskun kannalta. Muuten luodilla on samat muuttujat kuin muillakin pelin objekteilla, eli x- ja y-akseleiden lähdepisteet sprite-sivulla. Muuttujat ”this.width” ja ”this.height” kertovat luodin koon pikseleinä. Luodit piirtyvät canvaksen ulkopuolelle, eli ovat näkymättömissä alkuun. Tämä ilmenee ”this.drawX” -muuttujan arvolla -20, jolloin luodit sijoittuvat 20 pikseliä canvaksen vasemman yläreunan vasemmalle puolelle. Y-akselin arvo on nolla. ”Bullet”-objektilla on myös muuttuja ”this.explosion”, joka luo uuden ”Explosion”-objektin, joka on siis vihollisen räjähdysobjekti.

```
Bullet.prototype.draw = function(){
    this.drawX +=3;
    ctxHe-
ro.drawImage(imgSprite,this.srcX,this.srcY,this.width,this.heigh
```

```
t,this.drawX,this.drawY,this.width,this.height); //piirtää
luodin
    this.checkHitEnemy();
    if (this.drawX > gameWidth) this.recycle;
};
```

Yläpuolella oleva ”Bullet”-objektin prototyyppifunktio ”draw” ilmoittaa ”drawX”-muuttujan arvoksi +=3. Tämän ansiosta luoti piirtyy jokaisen canvasin piirtokerran jälkeen kolme x-koordinaattia oikealle. Tämän jälkeen luodaan normaalisti konteksti, joka piirtää luodin. Seuraavaksi prototyyppi tarkistaa ”checkHitEnemy” -funktiolla, onko luoti osunut viholliseen. Funktio checkHitEnemy osaa kierrättää luodin takaisin alkupisteeseen canvasin ulkopuolelle. Jos taas luoti ei osukaan viholliseen vaan liikkuu canvasin rajojen ulkopuolelle, if-lause tarkastaa tämän ja käy läpi ”recycle”-funktion, joka palauttaa luodin alkupisteeseen.

```
Bullet.prototype.fire = function(startX, startY){
    this.drawX = startX;
    this.drawY = startY;
};
Hero.prototype.checkShooting = function(){
if (this.isSpacebar && !this.isShooting){
this.bullets[this.currentBullet].fire(this.handX, this.handY);
    }
};
```

”Bullet”-prototyypin ”fire”-metodi vaihtaa ammuttaessa alkuperäisen piirtopisteen halutun objektin piirtopisteen aloituskohtaan. Tässä tapauksessa piirron aloituskohtana on ”handX” ja ”handY”, eli sankarin käsi, kuten toiseksi ylimpänä olevasta ”checkShooting”- funktiosta voidaan havaita.

```
Bullet.prototype.checkHitEnemy = function() {
    for (var i = 0; i < enemies.length; i++) {
        if (this.drawX >= enemies[i].drawX &&
            this.drawX <= enemies[i].drawX + enemies[i].width &&
            this.drawY >= enemies[i].drawY &&
            this.drawY <= enemies[i].drawY + enemies[i].height)
{
                this.explosion.drawX = enemies[i].drawX;
                this.explosion.drawY = enemies[i].drawY;
                this.explosion.hasHit = true;
                this.recycle();
                enemies[i].recycleEnemy();

    this.hero.updateScore(enemies[i].rewardPoints);
        }
```

```
    }
};
```

Yläpuolella on Bulletin ”checkHitEnemy” -prototyyppifunktio. Se alkaa for-toistorakenteella, joka toimii niin kauan kuin muuttuja ”i” on pienempi kuin ”enemies”-listan pituus. Alussa muuttujan ”i” arvo on nolla ja jokaisella suoritus-kerralla sen arvo kasvaa yhdellä. Tämän jälkeen seuraa pitkä if-lause, joka käy läpi, onko luodin ”drawX”-arvo suurempi tai yhtä suuri kuin vihollisen ”drawX”-arvo ja onko ”drawX” pienempi kuin vihollisen ”drawX” + vihollisen leveys. Li-säksi se käy läpi, onko ”drawY”-arvo suurempi tai yhtä suuri kuin vihollisen ”drawY”-arvo ja onko ”drawY” pienempi kuin vihollisen ”drawY” + vihollisen kor-keus. Näin saadaan selville, onko luoti osunut vihollisen koordinaatteihin. Jos kaikki on tähän mennessä toteutunut, funktio piirtää räjähdyksen vihollisen ”drawX”- ja ”drawY” -koordinaattien osoittamaan kohtaan. Tämän jälkeen funk-tio asettaa ”hasHit”-muuttujan arvoksi ”true”, sillä näiden perusteella viholliseen on osuttu. Seuraavaksi suoritetaan ”recycle”-funktiot, joissa luoti ja viholliset palaavat takaisin lähtöpisteisiinsä. Lopuksi vielä viimeinen koodirivi antaa ”re-wardPoints” -muuttujan määrittelemät pisteet pisteidenlaskuun.

```
Bullet.prototype.recycle = function() {
    this.drawX = -20;
};
//Bullet funktiot loppu
```

Yläpuolella vielä ”Bullet”-objektin ”recyle”-funktio, joka palauttaa luodin lähtöpis-teeseensä -20 x-koordinaattia.

```
function Bullet(h){
this.explosion = new Explosion();
    }
// räjähdys funktiot

function Explosion() {
    this.srcX = 305;
    this.srcY = 500;
    this.drawX = 0;
    this.drawY = 0;
    this.width = 27;
    this.height = 22;
    this.hasHit = false;
    this.currentFrame = 0;
    this.totalFrames = 10;
}
```

Kun “Bullet”-funktioissa viitattiin “explosion”-muuttujaan, joka luo uuden “Explosion”-objektin, tässä näemme miten uusi objekti ja sen attribuutit luodaan. Samaan tapaan kuin kaikki muutkin objektit, määritellään ensin sen lähdepisteet sprite-sivulla. Muuttujat ”drawX” ja ”drawY” on asetettu nollaksi, koska räjähdyksen piirtoalue on vaihteleva ja riippuvainen siitä, missä koordinaateissa räjähdys tapahtuu. Seuraavaksi on spriten koko pikseleinä. Muuttuja ”hasHit” on oletuksena asetettu epätodeksi. Muuttuja ”currentFrame” on 0 ja ”totalFrames” on 10. Tämä tarkoittaa, että kun räjähdys piirtyy, on sen aloitus canvasframe 0 ja räjähdyksen piirto kestää 10 kuvaruudun päivitystä. Tuo 10 kuvaruudun kesto on ihan sopiva aika tässä tapauksessa räjähdyksen näkyvissä olemiseen. Jos ”totalFramesin” arvoksi olisi annettu esimerkiksi 1, olisi räjähdyksen kesto vain pieni silmänräpäys.

```
Explosion.prototype.draw = function() {
    if (this.currentFrame <= this.totalFrames) {
        ctxHero.drawImage(imgSprite, this.srcX, this.srcY,
this.width, this.height, this.drawX, this.drawY, this.width,
this.height);
        this.currentFrame++;
    } else {
        this.hasHit = false;
        this.currentFrame = 0;
    }
};

// räjähdysfunktiot loppu
```

Yläpuolella on esitelty “Explosion.prototype.draw” -funktio, joka piirtää räjähdyksen. Funktio alkaa if-lauseella, jossa ehtona on, että ”currentFrame” -muuttujan arvo on pienempi tai yhtä suuri kuin ”totalFrames” -muuttujan arvo. Jos ehto toteutuu, piirtää funktio räjähdyksen sprite-kuvasta ”drawImage” -metodilla. Kuvan piirtämisen jälkeen ”currentFramen” arvo kasvaa yhdellä. Jos if-lauseen ehto ei täyty, ”hasHit”-muuttujan arvo on epätosi ja ”currentFramen” arvoksi palautuu nolla.

3.7.2 Pisteiden lasku

Tässä luvussa käsitellään, miten pelin pisteiden lasku toimii.

```
var ctxScore = canvasScore.getContext('2d');//määrittelee missä
formaatissa canvasta manipuloidaan (2d tai 3d)
ctxScore.fillStyle = "hsla(0,0%,0%,0.5)"; //(Hue, Saturation,
Lightness, opacity via Alpha)
ctxScore.font = "bold 20px Arial";
```

Ennen funktioita luodaan muuttuja, joka hakee pistelaskurin kontekstin HTML-sivun canvas-elementistä samaan tapaan kuin sankari, vihollinen ja taustakuvakin. Se mitä muilla pelin canvaksilla ei ole, on attribuutit ”fillStyle” ja ”font”. Näistä ensimmäisen, eli ”fillStylen” arvoksi on annettu `hsla(0,0%,0%,0.5)`. Hsla-lyhenne tarkoittaa sisällön värisävyn, saturaation, valoisuuden ja läpinäkyvyyden säätöominaisuuksia. Seuraavana oleva ”font” taas yksinkertaisesti määrittelee halutun fontin, jota sisältö käyttää. Tässä tapauksessa se on vahvennettu, kooltaan 20 pikseliä ja fontin tyyli on Arial.

```
function playGame(){
    drawBg();
    startLoop();
    updateScore();
    document.addEventListener('keydown',checkKeyDown,false);
    document.addEventListener('keyup',checkKeyUp,false);
          }
```

Pelin aloitusfunktiossa “playGame()” kutsutaan ”updateScore” -funktiota, jotta näkisimme heti pelin alkaessa pisteidenlaskun peliruudun oikeassa ylälaidassa.

```
function updateScore(){
```

```
ctxScore.clearRect(0,0,gameWidth,gameHeight);
ctxScore.fillText("Score: " + hero1.score, 680, 30);
        }
```

Funktio ”updateScore()” tyhjentää pisteenlaskualueen ennen pisteiden päivitystä. Muuten numerot piirtyisivät ruudulle päällekkäin ja lopputulos näyttäisi sekavalta. Pisteiden lukumäärän näyttäminen tapahtuu ”fillText” -funktiossa, jossa on kirjoitettu teksti ”Score: ” ja tämän jälkeen perään lisätään ”hero1”-muuttujan, eli ”Hero”-objektin pistemäärä ”score”-muuttujasta. Tulostaulu sijaitsee canvaksella kohdassa x= 680 ja y= 30 pikseliä.

```
function Hero(){
this.score = 0;
        }
```

”Hero”-objektin score-muuttujan alkuarvoksi on annettu nolla. Tällöin pelin alkaessa pisteitä on nolla.

```
// Pisteiden lasku


Hero.prototype.updateScore = function(points){
    this.score += points;
    updateScore();
    }
```

Yllä on ”Hero”-objektin prototyyppifunktio ”updateScore”, jossa on annettuna parametrinä ”points”. Funktio lisää ”score”-muuttujan arvoon pisteet, jotka ”updateScore” -funktio näyttää pelaajalle ruudulla.

```
Bullet.prototype.checkHitEnemy = function() {
    for (var i = 0; i < enemies.length; i++) {
        if (this.drawX >= enemies[i].drawX &&
            this.drawX <= enemies[i].drawX + enemies[i].width &&
            this.drawY >= enemies[i].drawY &&
            this.drawY <= enemies[i].drawY + enemies[i].height)
{
            this.explosion.drawX = enemies[i].drawX;
                this.explosion.drawY = enemies[i].drawY;
                this.explosion.hasHit = true;
                this.recycle();
                enemies[i].recycleEnemy();

    this.hero.updateScore(enemies[i].rewardPoints);
        }
    }
};
```

Kertauksen vuoksi tässä näytetään, kuinka ”checkHitEnemy” -funktion lopussa oleva koodirivi lähettää ”updateScore” -funktioon vihollisesta saatavan pistemäärän.

```
function Enemy(){
this.rewardPoints = 25;
}
```

”Enemy”-objektissa on annettu ”rewardPoints” -muuttujalle arvoksi 25. Tämä tarkoittaa, että jokaisesta tuhotusta vihollisesta saatava pistemäärä on 25. Näin olemme luoneet kaikki pelin funktiot ja toiminnallisuudet mitä pelaamiseen vähintään tarvitaan.

4 YHTEENVETO

Tämän opinnäytetyön tarkoituksena on tuoda lukija lähemmäs HTML5-pelinkehitystä ja havainnollistaa yksinkertaisen pelin rakennetta esimerkkipelin avulla. Esimerkkipelinä ohjelmoin yksinkertaisen selaimessa toimivan ammuntapelin, jossa tuhotaan vihollisia ja lasketaan pisteitä. Pelissä ei voi voittaa tai hävitä.

Työn tekeminen oli haastavaa, sillä kyseessä oli ensimmäinen peliohjelmointini. Prosessin aikana selvisi pelin tekemisestä sellaisia asioita, joita tavallinen pelinkuluttaja ei osaa pelatessaan ajatella. Olisin voinut peliä tehdessäni ottaa avuksi jonkin JavaScript -pelimoottorin, joka luultavasti olisi helpottanut joidenkin asioiden tekemistä, kuten vihollisten uudelleen ilmestymistä tuhoutumisen jälkeen tai muuta sellaista. Päädyin kuitenkin tekemään pelin ilman pelimoottoria ja mielestäni lopputuloksesta tuli alkuodotuksiin nähden ihan toimiva ja kehityskelpoinen. Sain ohjelmointityötä tehdessäni uuden kipinän koodaamiseen ja olenkin suunnitellut pelin varalle jatkokehitysideoita mahdollista julkaisua varten, kuten vaikeustason kasvamisen ja sankarihahmon tuhoutumisen mahdollisuuden. Koska pelissä esiintyvään Peräsmies-hahmoon tai muihinkaan pelissä esiintyviin sprite-objekteihin minulla ei ole tekijänoikeuksia, joutuisi ainakin ne suunnittelemaan uusiksi. Pelin aloitusvalikko on kuitenkin omaa käsialaani ja sitä voisi jatkossakin hyödyntää.

HTML5 toimi pelintekotarkoituksessa erittäin hyvin. Pelikehityksessä ei natiivin järjestelmän toimintalogiikoilla ole niinkään väliä vaan peleillä on yleensä omanlaisensa käyttöliittymät. Natiivikoodilla ohjelmoitavat ei peli-sovellukset joutuvat usien noudattamaan järjestelmässä vaadittavia toimintoja ja eleitä, jotta ne toimisivat sulavasti järjestelmän kanssa. Peleillä ei tällaista vaadita. Ja koska HTML5 toimii kaikilla sitä tukevilla alustoilla samalla tavalla on HTML5 todennäköisesti paras tekniikka mobiilipelien tekemiselle. Opinnäytetyössä tehty pelisovellus ei myöskään kasvanut hirveän isoksi ja paljon tilaa vieväksi, vain muutama megabitti, ja siitäkin suurin osa on kuvatiedostojen takia.

Adobe Dreamweaver ei ollut minulle aiemmin tuttu ennen tätä opinnäytetyötä, mutta se oli helppo omaksua ja muokata omien mieltymysten mukaiseksi. Havaitsin Dreamweaverin erittäin hyväksi työkaluksi projektini toteuttamista varten. Toisaalta jatkossa en välttämättä tekisi peliä Dreamweaverilla, sillä markkinoilla on varta vasten pelien tekemiseen suunniteltuja ohjelmistoja, jotka voisivat palvella käyttötarkoituksessa paremmin ja helpottaen joidenkin pelin osien tekemistä.

LÄHTEET

Adobe 2012. General information about coding in Dreamweaver. Viitattu 8.4.2014 http://help.adobe.com/en_US/dreamweaver/cs/using/WSc78c5058ca073340dcda9110b1f693f21-7bf1a.html.

Adobe Dreamweaver 2014. Wikipedia. Viitattu 8.4.2014 http://fi.wikipedia.org/wiki/Adobe_Dreamweaver.

HTML5 Game Engine 2013. HTML5 Game Engines. Viitattu 8.4.2014 www.html5gameengine.com.

JavaScript 2014. Wikipedia. Viitattu 7.4.2014 http://en.wikipedia.org/wiki/JavaScript.

Korpela, J.K. 2011. HTML5 Uudet Ominaisuudet. Jyväskylä: WSOYpro.

Mikael Immonen 2013. Case Universomo – Tampereen peliteollisuuden kehitys 2000 -luvulla. Kandidaatin tutkielma. Viitattu 7.4.2014

Moncur, M. 2000. JavaScript Trainer. Suom. Virpi, P. Jyväskylä: Gummerus.

Negrino, T. & Smith, D. 2007. JavaScript – Tehokas hallinta. Suom. Kamppila, M. Jyväskylä: Gummerus.

Sprite-grafiikka 2014. Wikipedia. Viitattu 13.4.2014 http://fi.wikipedia.org/wiki/Sprite-grafiikka.

Tero Aitta-Aho 2012. HTML5 ja julkaisujärjestelmät. Ammattikorkeakoulun opinnäytetyö. Viitattu 7.4.2014 Hämeen ammattikorkeakoulu.

Tomi Tukiainen 2013. Mobiilipelit ja pelimoottorit. Pro gradu -tutkielma. Viitattu 7.4.2014 Helsingin yliopisto.

W3Schools. HTML5 Introduction. Viitattu 8.4.2014 www.w3schools.com > HTML > HTML5 Intro.

W3Schools. HTML5 New Elements. Viitattu 8.4.2014 www.w3schools.com > HTML > HTML5 New Elements.

Pelin koodi

Liitteenä pelin ohjelmallinen JavaScript-koodi.

```
// JavaScript Document

var canvasBg = document.getElementById('canvasBg');
var ctxBg = canvasBg.getContext('2d');//määrittelee missä for-
maatissa canvasta manipuloidaan (2d tai 3d)
var canvasHero = document.getElementById('canvasHero');
var ctxHero = canvasHero.getContext('2d');//määrittelee missä
formaatissa canvasta manipuloidaan (2d tai 3d)
var canvasEnemy = document.getElementById('canvasEnemy');
var ctxEnemy = canvasEnemy.getContext('2d');//määrittelee missä
formaatissa canvasta manipuloidaan (2d tai 3d)
var canvasScore = document.getElementById('canvasScore');
var ctxScore = canvasScore.getContext('2d');//määrittelee missä
formaatissa canvasta manipuloidaan (2d tai 3d)
ctxScore.fillStyle = "hsla(0,0%,0%,0.5)"; //(Hue, Saturation,
Lightness, Alpha)
ctxScore.font = "bold 20px Arial";

var hero1 = new Hero;
var btnPlay = new Button(196,611,208,339);
var gameWidth = canvasBg.width;
var gameHeight = canvasBg.height;
var mouseX = 0;
var mouseY = 0;
var isPlaying = false;
var requestAnimFrame =  window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.msRequestAnimationFrame ||
window.oRequestAnimationFrame;

var spawnInterval;
var totalEnemies = 0;
var enemies = [];
var spawnRate = 2000;
var spawnAmount = 5;
var imgSprite = new Image();
imgSprite.src = 'images/sprites.png';
imgSprite.addEventListener('load',init,false); //lataa alapuo-
lella olevan init-funktion, joka määrittää spritejen piirtämisen

// pää funktiot

function init(){
    spawnEnemy(spawnAmount);
    drawMenu ();
```

```
    document.addEventListener('click',mouseClicked,false);
}

function playGame(){
    drawBg();
    startLoop();
    updateScore();
    document.addEventListener('keydown',checkKeyDown,false);
    document.addEventListener('keyup',checkKeyUp,false);
}

function spawnEnemy(number){
    for (var i = 0; i < number; i++){
          enemies[enemies.length] = new Enemy(); //luo uuden vi-
hollisen enemy listaan (enemies [totalEnemies]), kun for loopin
ehdot toteutuvat
    }
}
function drawAllEnemies(){
    clearCtxEnemy(); // estää piirtämästä vihollista kokoajan
uudelleen
    for (var i = 0; i < enemies.length; i++){
          enemies[i].draw();
    }
}


function loop(){
    if (isPlaying){
        hero1.draw();
        drawAllEnemies();
        requestAnimFrame(loop);
    }
}

function startLoop(){
    isPlaying = true;
    loop();
}

function stopLoop(){
    isPlaying = false;
}

function drawMenu (){
    var srcX = 0 ; //spriten aloituspiste x-akselilla
    var srcY = 650 ; //spriten aloituspiste y-akselilla
    var drawX = 0 ; //piirron aloituspiste x
    var drawY = 0 ; //piirron aloituspiste y
    ctxBg.drawImage(imgSprite,srcX,srcY,gameWidth,gameHeight,dra
wX,drawY,gameWidth,gameHeight); //piirtää taustakuvan
}
```

```
function drawBg (){
    var srcX = 0 ; //spriten aloituspiste x-akselilla
    var srcY = 0 ; //spriten aloituspiste y-akselilla
    var drawX = 0 ; //piirron aloituspiste x
    var drawY = 0 ; //piirron aloituspiste y
    ctxBg.drawImage(imgSprite,srcX,srcY,gameWidth,gameHeight,dra
wX,drawY,gameWidth,gameHeight); //piirtää taustakuvan
}

function clearCtxBg(){
    ctxBg.clearRect(0,0,gameWidth,gameHeight);
}

function updateScore(){
    ctxScore.clearRect(0,0,gameWidth,gameHeight);
    ctxScore.fillText("Score: " + hero1.score, 680, 30);
}
// pääfunktioiden loppu


//hero funktiot


function Hero(){
    this.srcX = 0;
    this.srcY = 500;
    this.width = 150;
    this.height = 116;
    this.speed = 2;
    this.drawX = 150;
    this.drawY = 116;
    this.handX =  this.drawX +140;
    this.handY = this.drawY +20;
    this.isUpKey = false;
    this.isRightKey = false;
    this.isDownKey = false;
    this.iLeftKey = false;
    this.isSpacebar = false;
    this.isShooting = false;
    this.bullets =[];
    this.currentBullet = 0;
    for (var i = 0; i < 35; i++){
          this.bullets[this.bullets.length] = new Bullet(this);
    }
    this.score = 0;
}



Hero.prototype.draw = function(){
    clearCtxHero(); // estää piirtämästä sankaria kokoajan  uu-
delleen
```

```
    this.updateCoordinates();
    this.checkDirection();
    this.handX = this.drawX +140;
    this.handY = this.drawY +20;
    this.checkShooting();
    this.drawAllBullets();
    ctxHe-
ro.drawImage(imgSprite,this.srcX,this.srcY,this.width,this.heigh
t,this.drawX,this.drawY,this.width,this.height); //piirtää
sankarin
};

Hero.prototype.updateCoordinates = function () {
    this.handX =  this.drawX +140;
    this.handY = this.drawY +20;
    this.leftX = this.drawX;
    this.rightX = this.drawX + this.width;
    this.topY = this.drawY;
    this.bottomY = this.drawY + this.height;
}


Hero.prototype.checkDirection = function(){
    if (this.isUpKey && this.topY > 0) {
        this.drawY -= this.speed;
    }
    if (this.isRightKey && this.rightX < gameWidth) {
        this.drawX += this.speed;
    }
    if (this.isDownKey && this.bottomY < gameHeight) {
        this.drawY += this.speed;
    }
    if (this.isLeftKey && this.leftX > 0) {
        this.drawX -= this.speed;
    }

};



Hero.prototype.drawAllBullets = function(){
    for (var i = 0; i < this.bullets.length; i++){
          if(this.bullets[i].drawX >=0)
this.bullets[i].draw();// piirtää kaikki bulletit joiden drawX
arvo >0
          if (this.bullets[i].explosion.hasHit)
this.bullets[i].explosion.draw();
    }
};

Hero.prototype.checkShooting = function(){
    if (this.isSpacebar && !this.isShooting){ //!this.isShootin
```

```
on sama kuin this.isShooting === false
          this.isShooting = true;
          this.bullets[this.currentBullet].fire(this.handX,
this.handY);
          this.currentBullet++;
          if(this.currentBullet >= this.bullets.length)
this.currentBullet = 0;
    }else if (!this.isSpacebar){ //!this.isSpacebar on sama kuin
this.isSpacebar === false
        this.isShooting = false;
    }
};

Hero.prototype.updateScore = function(points){
    this.score += points;
    updateScore();
}


function clearCtxHero(){
    ctxHero.clearRect(0,0,gameWidth,gameHeight);
}

// hero funktioiden loppu


//Bullet funktiot

function Bullet(h){
    this.hero = h;
    this.srcX = 230;
    this.srcY = 500;
    this.drawX = -20;
    this.drawY = 0;
    this.width = 6;
    this.height = 18;
    this.explosion = new Explosion();
}

Bullet.prototype.draw = function(){
    this.drawX +=3;
    ctxHe-
ro.drawImage(imgSprite,this.srcX,this.srcY,this.width,this.heigh
t,this.drawX,this.drawY,this.width,this.height); //piirtää
sankarin
    this.checkHitEnemy();
    if (this.drawX > gameWidth) this.recycle;
};

Bullet.prototype.fire = function(startX, startY){
    this.drawX = startX;
    this.drawY = startY;
```

```
};

Bullet.prototype.checkHitEnemy = function() {
    for (var i = 0; i < enemies.length; i++) {
        if (this.drawX >= enemies[i].drawX &&
            this.drawX <= enemies[i].drawX + enemies[i].width &&
            this.drawY >= enemies[i].drawY &&
            this.drawY <= enemies[i].drawY + enemies[i].height)
{
            this.explosion.drawX = enemies[i].drawX;
            this.explosion.drawY = enemies[i].drawY;
            this.explosion.hasHit = true;
            this.recycle();
            enemies[i].recycleEnemy();
            this.hero.updateScore(enemies[i].rewardPoints);
        }
    }
};
Bullet.prototype.recycle = function() {
    this.drawX = -20;
};
//Bullet funtiot loppu

// räjähdys funktiot

function Explosion() {
    this.srcX = 305;
    this.srcY = 500;
    this.drawX = 0;
    this.drawY = 0;
    this.width = 27;
    this.height = 22;
    this.hasHit = false;
    this.currentFrame = 0;
    this.totalFrames = 10;
}

Explosion.prototype.draw = function() {
    if (this.currentFrame <= this.totalFrames) {
        ctxHero.drawImage(imgSprite, this.srcX, this.srcY,
this.width, this.height, this.drawX, this.drawY, this.width,
this.height);
        this.currentFrame++;
    } else {
        this.hasHit = false;
        this.currentFrame = 0;
    }
};

// räjähdysfunktiot loppu

//enemy funktiot
```

```
function Enemy(){
    this.srcX = 150;
    this.srcY = 500;
    this.width = 80;
    this.height = 130;
    this.speed = 2;
    this.drawX = Math.floor(Math.random() * 1000) + gameWidth;
// Math.floor lyhentää Math.random tuloksen alimpaan desimaaliin
ja antaa jonkin random luvun 0-800px välillä
    this.drawY = Math.floor(Math.random() * 360);
    this.rewardPoints = 25;
}

Enemy.prototype.draw = function(){
    this.drawX -= this.speed;// saa vihollisen tulemaan oikealta
vrt: this.drawX += this.speed, jolloin vihollinen tulisi vasem-
malta
    ctxEne-
my.drawImage(imgSprite,this.srcX,this.srcY,this.width,this.heigh
t,this.drawX,this.drawY,this.width,this.height); //piirtää vi-
hollisen
    this.checkEscaped();
};

Enemy.prototype.checkEscaped = function(){
    if (this.drawX + this.width <= 0){
          this.recycleEnemy();
    }
};

Enemy.prototype.recycleEnemy = function(){
    this.drawX = Math.floor(Math.random() * 1000) + gameWidth;
// Math.floor lyhentää Math.random tuloksen alimpaan desimaaliin
ja antaa jonkin random luvun 0-800px välillä
    this.drawY = Math.floor(Math.random() * 360);
};

function clearCtxEnemy(){
    ctxEnemy.clearRect(0,0,gameWidth,gameHeight);
}


//enemy funtiot loppu

//button objekti

function Button(xL, xR, yT, yB){
    this.xLeft = xL;
    this.xRight = xR;
    this.yTop = yT;
    this.yBottom = yB;
```

```
}

Button.prototype.checkClicked = function(){
    if (this.xLeft <= mouseX && mouseX <= this.xRight &&
this.yTop <= mouseY && mouseY <= this.yBottom) return true;
};

// button objekti loppu

// tapahtumafunktiot
function mouseClicked(e){
    mouseX = e.pageX - canvasBg.offsetLeft;
    mouseY = e.pageY - canvasBg.offsetTop;
    if (!isPlaying){ //korjaa bugin, jolla pelin nopeus lähti
käsistä canvasta klikattaessa
        if(btnPlay.checkClicked())playGame();
    }
}

function checkKeyDown(e){
    var keyID = e.keyCode || e.which; //jos selaimesta ei löydy
e.keycode standardia käytetään e.which
    if (keyID === 38 || keyID === 87){ //38= ylänuolinäppäin tai
87 = W-näppäin
        hero1.isUpKey = true;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 39 || keyID === 68){ //39= oikeanuolinäppäin
tai 68 = D-näppäin
        hero1.isRightKey = true;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 40 || keyID === 83){ //40= alanuolinäppäin tai
83 = S-näppäin
        hero1.isDownKey = true;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 37 || keyID === 65){ //37= vasennuolinäppäin
tai 65 = A-näppäin
        hero1.isLeftKey = true;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 32){ // =välilyönti
        hero1.isSpacebar = true;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
}
```

```
function checkKeyUp(e){
    var keyID = e.keyCode || e.which; //kertoo yleisimmille eri
selamille mitä nappia painetaan
    if (keyID === 38 || keyID === 87){ //38= ylänuolinäppäin tai
87 = W-näppäin
        hero1.isUpKey = false;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 39 || keyID === 68){ //39= oikeanuolinäppäin
tai 68 = D-näppäin
        hero1.isRightKey = false;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 40 || keyID === 83){ //40= alanuolinäppäin tai
83 = S-näppäin
        hero1.isDownKey = false;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 37 || keyID === 65){ //37= vasennuolinäppäin
tai 65 = A-näppäin
        hero1.isLeftKey = false;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }
    if (keyID === 32){ // =välilyönti
        hero1.isSpacebar = false;
        e.preventDefault(); //estää nuolinäppäimiä scrollaamasta
websivua ylös ja alas kesken pelin
    }


}
```