Krista Laiho

HTML5 Canvas

Canvasin soveltuminen banneriin

Metropolia Ammattikorkeakoulu

Medianomi

Viestinnän koulutusohjelma

Opinnäytetyö

22.11.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Krista Laiho HTML5 Canvas – Canvasin soveltuminen banneriin 54 sivua + 2 liitettä 22.11.2013
Tutkinto	Medianomi
Koulutusohjelma	Viestinnän koulutusohjelma
Suuntautumisvaihtoehto	Digitaalinen viestintä
Ohjaaja(t)	Yliopettaja Pauli Laine

Opinnäytetyön pääasiallisena tavoitteena oli tutkia, mikä HTML5 Canvas on ja miten sillä piirretään kaksiulotteista grafiikkaa. Canvas on melko uusi ja tehokas HTML5-elementti grafiikan, animaatioiden ja pelien toteuttamiseen suoraan selaimeen ilman kolmansien osapuolten liitännäisiä. Työn toisena tavoitteena oli tehdä banneri canvasilla ja tutkia canvasin soveltuvuutta banneriin.

Opinnäytetyön teoriaosuus perustuu laajaan lähdekirjallisuuteen, omiin havaintoihin ja käytännön kokeiluun. Teoriaosuuden ensimmäinen osa käsittelee HTML5 Canvasin taustaa, canvas-elementtiä ja 2d-kontekstia, canvasin mahdollisuuksia sekä canvasin toimivuutta. Teoriaosuuden toinen osa keskittyy canvasille piirtämiseen koodin kautta. Luku sisältää tietoa canvasin käyttöönotosta, koordinaatistojärjestelmästä, 2d-kontekstin ohjelmointirajapinnasta ja erilaisista JavaScript-kirjastoista canvasille.

Opinnäytetyön toiminnallisessa osassa tehdään banneri canvasilla ja KineticJS -JavaScript-kirjastolla. Banneri toimi sulavasti useimmilla selaimilla ja testatuilla mobiililaitteilla. Bannerin grafiikka toistui joissain selaimissa hieman epätarkasti ja bannerin tiedostokoko oli isompi KineticJS-kirjaston vuoksi kuin useimmissa teknisissä määrityksissä sallitaan.

Opinnäytetyö on tehty itsenäisenä tutkimustyönä. Työ toimii hyvänä oppaana canvasin perusteisiin opiskelijoille ja muille verkkoteknologioista kiinnostuneille. Käytännöntyönosuus voi kiinnostaa erityisesti yrityksiä tai muita, jotka etsivät vaihtoehtoisia menetelmiä bannereiden toteuttamiseen.

Avainsanat	HTML5 Canvas, 2d-konteksti, animaatio, banneri, KineticJS

Author(s) Title Number of Pages Date	Krista Laiho HTML5 Canvas - Canvas' suitability for banner 54 pages + 2 appendices 22 October 2013
Degree	Bachelor of Arts and Culture
Degree Programme	Media
Specialisation option	Digital Media
Instructor(s)	Pauli Laine, Principal Lecturer

The main aim of the Thesis was to study what HTML5 Canvas is and how it applies to drawn two-dimensional graphics. Canvas is a new powerful HTML5 element that provides API for graphics, animations and games. Canvas does not need any additional plugins to function and it is ready to use on the fly. The second aim of the Thesis was to create a banner with Canvas and study its suitability for the banner.

The theoretical part of the Thesis is based on extensive literary references, personal observations and practical experiments. The first part of the theory handles background of the HTML5 Canvas, Canvas element and the 2d context, possibilities of the Canvas and how the Canvas operates. The second theoretical part of the Thesis focuses on how to draw with the Canvas using code. The chapter includes information about how to use Canvas, the coordinate system, 2d context API and a range of JavaScript libraries for Canvas.

The operational part of the Thesis is about creating banner with Canvas and KineticJS JavaScript library. The banner played smoothly in most browsers and mobile devices that were tested. However, in the some browsers banner's graphic rendered slightly inaccurately and the banner's file size was bigger than a general recommendation is for file size.

The Thesis is an independently made study. The Thesis serves as a good guide to the basics of Canvas for students and other readers who are interested in web technologies. The operational part may be of particular interest for companies or others who are looking for alternative methods of creating banners.

Keywords	HTML5 Canvas, 2D context, animation, banner, KineticJS

Sisällys

Liitteet

Liite 1. Työssä käytettyjä termejä

Liite 2. Bannerin JavaScript-lähdekoodi

1 Johdanto

Opinnäytetyöni aiheena on HTML5 Canvas ja toiminnallinen banneri canvasia käyttäen. Canvas on HTML5:n elementti, jota voidaan käyttää muun muassa grafiikan, animaatioiden ja selaimessa toistettavien pelien tekemiseen. Bannerit taas ovat verkkosivuilla olevia suorakulmionmallisia mainoksia, joiden tekemiseen käytetään tavallisesti Flash-animaatioita tai kuvia.

HTML5 Canvas on vielä melko uutta tekniikkaa ja sen vuoksi vielä tuntematon teknologia monelle alan opiskelijalle ja ammattilaiselle. Opinnäytetyö on tarkoitettu verkkoteknologioiden opiskelijoille ja muille alasta kiinnostuneille. Opinnäytetyön pääasiallisena tavoitteena on johdattaa lukija HTML5 Canvasiin ja se toimii hyvin oppaana canvasin perusteisiin. Opinnäytetyön toisena tavoitteena on toteuttaa banneri käytännön työnä canvasilla ja tutkia canvasin soveltuvuutta banneriin. Opinnäytetyö saattaa erityisesti kiinnostaa alan ammattilaisia, jotka etsivät vaihtoehtoisia menetelmiä bannereiden tekemiseen. Omana tavoitteenani on oppia käyttämään canvasia ja tekemään canvasin avulla suurempia käytännöntöitä, jotta voin hyödyntää osaamistani tulevaisuudessa työelämässä.

Työn aihe syntyi omasta tarpeesta ja kiinnostuksesta canvasia kohtaan. Olen valinnut verkkoteknologiat syventäviksi opinnoikseni ja aikaisempi kokemukseni canvasista on hyvin vähäinen. Olen kokeillut canvasia kerran aikaisemmin ja minua kiinnostaa, minkälaisiin töihin canvas käytännössä taipuu ja miten se toimii. Opinnäytetyön käytännöntyön aihe muodostui siitä, että canvasia ei käytetä vielä bannereihin juuri ollenkaan. Mainostoimistossa työskennellessä huomasin, että bannereihin käytetään vielä Flashia, jonka ongelma on sen toimimattomuus esimerkiksi mobiililaitteissa. Flash-animaatio tarvitsee myös erillisen liitännäisen toimiakseen. Tämän pohjalta haluan tutkia canvasin soveltuvuutta bannereihin uuden tekniikan näkökulmasta. HTML5 Canvasin oppimisen kannalta pidän tärkeänä myös tehdä suuremman käytännöntyön teoriaa soveltaen, joten banneri toimii myös hyvänä esimerkkityönä opitusta.

HTML5 Canvas on aihealueeltaan laaja canvasin monikäyttöisyyden vuoksi. Opinnäytetyöni keskittyy kaksiulotteiseen grafiikkaan ja canvasille piirtämiseen koodin kautta. Aiheeseen on otettu mukaan JavaScript-kirjastot, koska niiden avulla on helpompi animoida canvasia, kun omat JavaScript taidot eivät ole tarpeeksi kehittyneet. Kirjasto

myös laajentaa canvasin omia sisäänrakennettuja ominaisuuksia. Työssä ei käsitellä paljon interaktiivisia animaatioita niiden laajuuden vuoksi. Opinnäytetyön lähestymistapa on tekninen ja käytännönläheinen, joten opinnäytetyöstä rajautuu pois muun muassa käytännöntyön ulkonäkö ja bannerin viesti. Bannereihin ei myöskään syvennytä liiemmin vaan käytännöntyössä korostuu bannerin tekeminen canvasilla teknisesti.

Opinnäytetyöni on luonteeltaan kvalitatiivinen eli laadullinen ja työtä on tehty vuoden 2013 helmikuusta marraskuun loppuun asti. Tiedonhakumenetelmänä on tutustuttu laajaan ja luotettavanoloiseen lähdekirjallisuuteen sekä useisiin canvasilla tehtyihin esimerkkeihin. Lähdekirjallisuuden lisäksi työn teoreettinen puoli perustuu omiin kokeiluihin ja havaintoihin. Monet opinnäytetyön canvasilla tehdyt esimerkit on tehty soveltamalla valmiita canvasilla tehtyjä esimerkkejä. Myös suuri osa canvasin mahdollisuuksista on havainnoitu itse verkosta olevista canvasilla tehdyistä töistä. Opinnäytetyön käytännönosuus perustuu pääasiallisesti lähdeaineistosta opittuun sekä omaan käytännön kokeiluun ja soveltamiseen. Luvun teksti on kirjoitettu oppimispäiväkirjan perusteella ja se perustuu pääosin omiin havaintoihin ja kokeiluihin.

Opinnäytetyöni ymmärtäminen edellyttää lukijalta perustiedot HTML5:sta ja JavaScriptistä sekä perustietämystä CSS:stä. Työhön on myös sisälletty englanninkielisiä termejä selkeyden lisäämiseksi, koska osa termeistä on tunnetumpia englanniksi. Työssä käsiteltyihin verkkoteknologioiden perustermeihin voi tutustua liitteessä 1.

Luvussa 2 luodaan katsaus HTML5 Canvasiin. Aluksi tarkastellaan lyhyesti canvasin taustaa sekä määritetään, mitä HTML5 Canvas pitää sisällään. Luvussa kartoitetaan myös, mitä canvasilla voidaan tehdä ja esitetään käytännön esimerkkejä. Lopuksi käydään läpi canvasin toimimista selaimissa ja mobiililaitteissa.

Luvussa 3 käsitellään piirtämistä HTML5 Canvasille koodin kautta. Aluksi esitetään, miten canvas voidaan ottaa käyttöön ja miten canvasin käyttämä koordinaatisto toimii. Luvussa käsitellään pääosin piirtämistä canvasille 2d-kontekstin omia sisäänrakennettuja metodeja ja ominaisuuksia käyttäen. Lopuksi tarkastellaan erilaisia JavaScript-kirjastoja, jotka laajentavat canvasin käytön mahdollisuuksia.

Luvussa 4 käsitellään bannerin toteuttamista HTML5 Canvasilla ja KineticJS -JavaScript-kirjastolla. Aluksi luodaan lyhyt katsaus bannereihin, jonka pohjalta käsitellään tämän opinnäytetyön bannerin suunnittelua. Luku käsittelee pääosin bannerin

toteuttamisprosessia, joka sisältää canvasin ja KineticJS-kirjaston käyttöönottoa, elementtien ja liikeanimaatioiden luomista KineticJS-kirjastolla sekä bannerin lopputuloksen esittelyä. Lopuksi mietitään canvasin soveltumista käytännöntyönä tehtyyn banneriin.

2 Katsaus HTML5 Canvasiin

Tässä luvussa luodaan katsaus HTML5 Canvasiin ja sen käyttömahdollisuuksiin. Luvussa kartoitetaan lyhyesti canvasin ja HTML5:n taustaa. Seuraavaksi selvitetään canvas-elementtiä ja 2d-kontekstia sekä esitetään käytännön esimerkkejä canvasilla tehdyistä töistä. Luvun lopuksi kartoitetaan canvasin toimivuutta selaimissa ja mobiililaitteissa. Aiheet on valittu yleiskäsityksen saamiseksi HTML5 Canvasista.

2.1 HTML5 Canvasin taustaa

Canvasin on kehittänyt alun perin Apple vuonna 2004. Canvasia oli tarkoitus käyttää työpöydän widgeteissä ja tehostamassa grafiikkaa Safari-selaimessa. Myöhemmin myös muut selaimet, kuten Firefox, Google Chrome ja Opera, alkoivat tukea Canvasia. Applen pelättiin hakevan oikeuksia Canvasista, mutta myöhemmin Apple julkaisi Canvasin W3C:n maksuttoman patenttisopimuksen alle. (Albers & Lubbers & Salim 2010, 25; Rowell 2011, 3.)

Canvas on uusi html-elementti ja se on osa uusinta html-versiota eli HTML5:ta. Vuonna 2008 W3C esitteli ensimmäisen luonnoksensa HTML5:sta. HTML5 toi mukanaan uusia tehokkaita ominaisuuksia ja parannuksia, joilla saadaan lisättyä dynaamisuutta verkkosivuille. Entiset versiot html:stä toimivat hyvin dokumenttipohjaisessa verkossa. Dynaamisten ja mediarikkaiden verkkosivujen lisäännyttyä jouduttiin käyttämään verkkosivuissa kolmansien osapuolten liitännäisiä (plugin) interaktiivisuuden luomiseen. HTML5 tuo paljon uusia elementtejä sisällön rakenteeseen, kuten navigaatio-, ala- ja ylätunniste-elementit sekä mediaelementtejä videon, äänen ja grafiikan esittämiseen. HTML5 tuo myös parannuksia muun muassa lomakkeiden validointiin. (Hawkes 2011, 1-4, 8-10.)

HTML5:ta ei ole vielä standardoitu. W3C:n olisi tarkoitus esitellä valmis HTML5-standardi vuoden 2014 loppuun mennessä ja HTML5.1-versio vuonna 2016. Monet selaimet ja web-kehittäjät ovat jo pitkään tukeneet HTML5:ta, vaikka se onkin vielä keskeneräinen. (Bright 2012.)

2.2 HTML5 Canvas -elementti ja 2d-konteksti

Canvasia käytettäessä on tärkeää ymmärtää canvas-elementin ja canvasin kontekstin (context) ero (Rowell 2012a). Canvas-elementti tarjoaa säiliön kontekstille ja konteksti taas mahdollistaa grafiikan piirtämisen (Geary 2012, 694). Canvas-elementti on dokumenttioliomallin (DOM) solmu (node), joka upotetaan html-sivuun. Canvasin konteksti on taas olio (object), jolla on erilaisia ominaisuuksia ja metodeja. Ominaisuuksia ja metodeja käytetään grafiikan kuvantamiseen (render) canvas-elementin sisällä. (Rowell 2012a.)

Canvas-elementti on html-tägi, jota käytetään piirtämään grafiikkaa verkkosivuille useimmiten JavaScriptin avulla. HTML5 Canvas on bittikarttagrafiikkaa, joten canvas-alue peittää kaikki canvas-elementin alla olevat pikselit. Canvas-elementissä itsessään ei ole sisältöä tai reunoja, vaan se toimii ikään kuin säiliönä grafiikalle. (Geary 2012, 524; Mozilla Developer Network 2013; Rowell 2011, 3; W3C 2013.) Canvas-elementti luo läpinäkyvän suorakulmaisen alueen verkkosivulle. Alue on leveydeltään 300 pikseliä ja korkeudeltaan 150 pikseliä, kun leveyttä ja korkeutta ei ole määritelty erikseen. (Albers & Lubbers & Salim 2010, 26.) Kuviossa 1 on esitetty canvas-elementin alku- ja loppu-tägit yksinkertaisessa muodossa.

```
<canvas></canvas>
```

Kuvio 1. Canvas-elementti ilman määritettyä pituutta ja leveyttä sekä id-arvoa.

JavaScript-koodissa ei ole paljon käyttöä canvas-elementille itselleen, vaan canvasin mielenkiinto piilee kontekstissa. 2d-kontekstilla piirretään kaksiulotteista grafiikkaa canvasille yleensä JavaScriptillä. 2d-konteksti tarjoaa ohjelmointirajapinnan (API) muun muassa erilaisten muotojen piirtämiseen sekä tekstin ja kuvien esittämiseen. 2d-kontekstilla tehdyn grafiikan voi kuvantaa canvas-elementin kautta. (Geary 2012, 694; Hawkes 2011, 57–58; W3C 2012.) Kontekstista löytyy kuitenkin joitain puutteita ja rajoi-

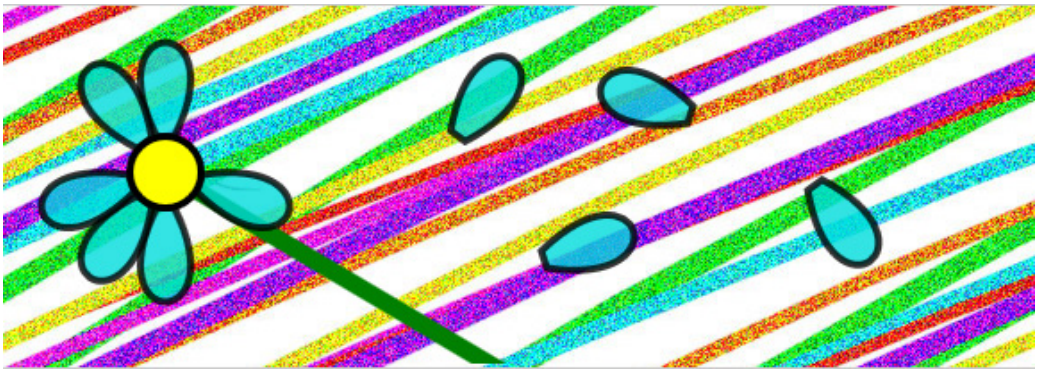
tuksia. 2d-konteksti ei esimerkiksi tue katkoviivoilla (dashed line) piirtämistä. JavaScript on kuitenkin dynaaminen kieli, jonka avulla voidaan lisätä itse 2d-kontekstin mahdollisuuksia ja tehdä omia uusia metodeja. (Geary 2012, 709.) 2d-kontekstia tarkastellaan tarkemmin luvussa 3.3.

Canvasille ei ole olemassa ainoastaan yhtä määritelmää. Canvasin on määritellyt W3C ja WHATWG. Kummatkin määritelmät canvasista ovat hyvin lähellä toisiaan, mutta erojakin löytyy. WHATWG sisällyttää canvasin kontekstin määrityksiin, kun taas W3C on erottanut canvasin kontekstista. (Geary 2012, 791, 815.) W3C:n määrityksessä kontekstista käytetään nimityksiä HTML Canvas 2D Context ja CanvasRenderingContext2D. WHATWG:n sivuilla taas käytetään nimitystä The 2D rendering context. Sisällöltään nämä tarkoittavat samaa. (WHATWG 2013; W3C 2012.) Myös internetissä ja kirjallisuudessa käsitellään canvasia erilaisin termein. Tässä työssä käytetään termiä canvas tai HTML5 Canvas kuvaamaan koko kokonaisuutta, johon sisältyy canvas-elementti ja konteksti. Elementtiä kuvataan nimellä canvas-elementti ja kontekstista käytetään termiä 2d-konteksti tai konteksti.

2.3 HTML5 Canvasin käyttömahdollisuuksia

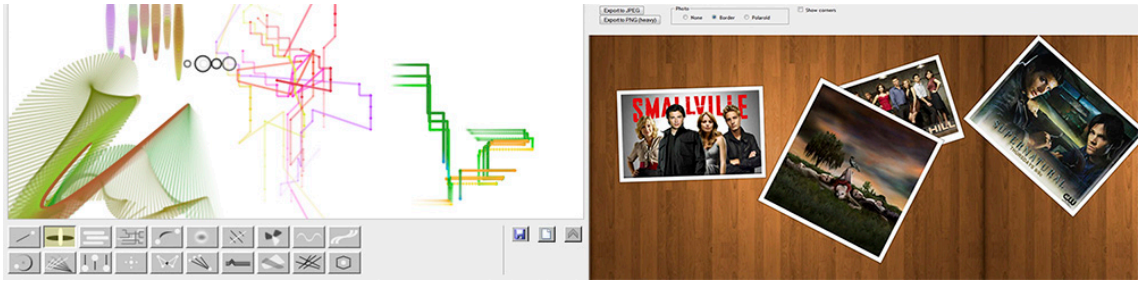
HTML5 Canvas konteksteineen mahdollistaa monenlaisen grafiikan kuvantamisen. Se soveltuu muun muassa erilaisten kuvioiden ja kuvien piirtämiseen, eritasoisiin animaatioihin, selaimessa toimiviin peleihin, interaktiivisiin animaatioihin, -gallerioihin ja muihin applikaatioihin, kuten piirtoapplikaatioihin. 2d-konteksti tarjoaa myös erilaisia metodeja ja ominaisuuksia kuvien manipuloimiseen ja muokkaamiseen, kuten värien muokkaamisen ja rajaamistyökalun, sekä erilaisia efektejä, kuten varjostuksen ja siirtymiä. (Geary 2012, 508; Hawkes 2011, 13; W3 Schools 2013a.) Canvas mahdollistaa myös videon manipuloimisen, kun HTML5:n video-elementti asetetaan canvas-elementin sisälle (Hawkes 2011, 152–156).

Kun tarkastelin canvasilla tehtyjä töitä verkosta, huomasin animaatioiden olevan eritasoisia. Animaatio voi olla yksinkertaisimpana pomppiva pallo, sivuttain kulkeva suorakulmio tai välkkyvä tähti. Monet canvasilla tehdyt animaatiot ovat interaktiivisia eli ne reagoivat esimerkiksi käyttäjän kosketukseen tai hiireen. Esimerkiksi kuviossa 2 on interaktiivinen kukka, jota käyttäjä pystyy liikuttamaan, niin että kukan alavarsi pysyy samassa paikassa. Kukan terälehdet ovat myös irrotettavissa yksi kerrallaan.



Kuvio 2. Interaktiivinen Canvasilla tehty animaatio, jota pystyy liikuttamaan ja josta saa irrotettua terälehtiä (Rowell 2013a).

Canvasia käytetään myös pienempiin applikaatioihin, kuten piirto-ohjelmien ja kuvagallerioiden tekemiseen. Kuvion 3 vasemmalla puolella on esimerkkinä piirto-ohjelma, johon käyttäjä voi piirtää kuvioita valitsemillaan animoiduilla piirtotyökaluilla. Oikealla puolella on esimerkki kuvagalleriasta, jonka kuvia voi kääntää ja skaalata.



Kuvio 3. Canvasilla tehty piirtoalusta ja kuvagalleria (Bonomo 2013; Osmani 2012).

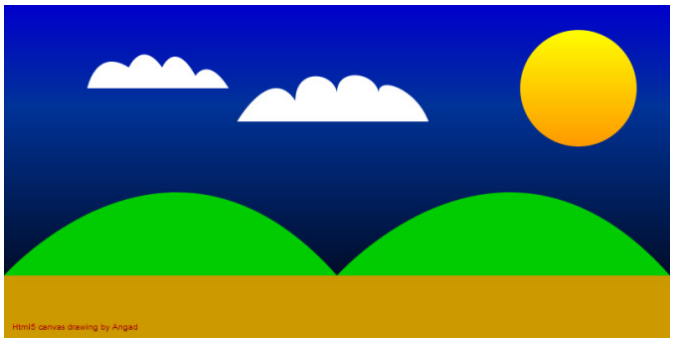
Canvasia käytetään paljon selaimessa toistettavien pelien tekemiseen. 2d-kontekstilla tehdyt pelit vaihtelevat pulmapeleistä (puzzle game) erilaisiin tasohyppely- ja ampumapeleihin. Kuvion 4 vasemmalla puolella on esimerkkinä ampumapeli, jossa liikutaan pysty- ja vaakasuunnassa nuolinäppäimillä sekä ammutaan vastaantulevia vihollisia. Oikealla puolella on esimerkki matemaattisesta pulmapelistä, jossa klikataan hiirellä numeroita tietyn summan verran.

Kuvio 4. Vasemmalla puolella on ampumapeli ja oikealla matemaattinen pulmapeli (Wagner 2009; Hyperandroid 2010).

Videoihin voi lisätä erilaisia efektejä. Selaamiini videoihin on lisätty muun muassa valoefektiä videon reunoille, vaihdettu videon tausta ja videon päälle lisätty kokoa muuttava kuva. Kuvion 5 videoesimerkissä käyttäjä pystyy räjäyttämään osan videokuvasta hiiren avulla. Videon palaset yhdistyvät takaisin videoon hetken kuluttua.

Kuvio 5. Video, jonka voi hajottaa palasiksi hiiren klikkauksella (Christmann 2010).

Canvasin 2d-konteksti tarjoaa erilaisia kurveja ja muotoja, joita yhdistelemällä voidaan piirtää erilaisia kuvioita. Canvasiin voidaan yhdistää myös valmiita kuvia. Kuviossa 6 on piirretty maisemakuva koodin kautta käyttämällä 2d-kontekstin ohjelmointirajapintaa. Yksinkertaisempien muotojen, viivojen ja kaarien piirtämistä 2d-kontekstin ohjelmointirajapinnalla käsitellään tarkemmin luvussa 3.3.



Kuvio 6. 2d-konteksilla piirretty kuva (Rajput 2012).

2.4 HTML5 Canvasin toimivuus selaimissa ja mobiililaitteissa

Canvas toimii hyvin muun muassa pöytäkoneissa ja monet merkittävimmät selaimet, kuten Firefox, Chrome, Safari, Opera ja Internet Explorer, tukevat pääosin HTML5 Canvasia. Firefox, Chrome Safari ja Opera ovat tukeneet HTML5:ta kauemmin, kun taas Internet Explorer on alkanut tukea sitä versioissa yhdeksän ja kymmenen. (Geary 2012, 823; Hawkes 2011, 58.)

Jos halutaan canvasin toimivan vanhemmilla Internet Explorer -selaimilla, kuten versioilla seitsemän ja kahdeksan, on tehtävä erityistoimenpiteitä. Yksi vaihtoehto on käyttää Googlen kehittämää Explorercanvasia. Explorercanvasia käytetään lisäämällä pieni pätkä koodia tiedostoon, jossa käytetään canvasia. Explorercanvasia käytettäessä canvasin pitäisi toimia samalla tavalla kuin Internet Explorerin yhdeksännessä versiossa. Toinen vaihtoehto on käyttää Google Chrome Framea, joka korvaa Internet Explorer-selaimen Chrome-selaimen moottorilla. Jos vanhempia Internet Explorer-versioita ei tarvitse tukea, niin voidaan käyttää koodissa varasisältöä, kuten tekstiä. (Geary 2012, 823; Hawkes 2011, 58.)

Gearyn (2012, 24504) ja Rowellin (2011a, 6129) mukaan canvas toimii mobiililaitteissa vaihtelevasti. Mobiililaitteiden suorituskyky canvasin pyörittämiseen on monesti heikko, koska laitteissa ei ole tarpeeksi hyviä prosessoreja ja selaimet käyttävät prosessoria canvasin kuvantamiseen. Tableteissa on paremmat prosessorit kuin älypuhelimissa, joten canvas toimii niissä paremmin. Huonompi suorituskyky voi näkyä esimerkiksi monimutkaisempien animaatioiden pätkimisenä. Uudemmat mobiililaitteet toistavat canvasia kokoajan paremmin ja sulavammin. (Geary 2012, 24504; Hawkes 2011, 277; Rowell 2011, 6129.)

Testatessani eritasoisia canvasilla toteutettuja piirroksia ja animaatioita internetistä iPad3-tabletilla (iOS) ja Sonyn Xperia Z -älypuhelimella (Android) havaitsin suorituskyvyn olevan heikompi monimutkaisempien- ja interaktiivisten animaatioiden kohdalla. Monet animaatiot pätkivät, reagoivat kosketukseen jäljessä tai interaktiivisuus ei muuten toiminut. Osa animaatioista ei myöskään piirtynyt oikein näytölle. Kuten edellisessä kappaleessa mainittiin, havaitsin myös itse animaatioiden toimivan hieman paremmin ja sulavammin tabletilla kuin älypuhelimella. Muun muassa interaktiivisten elementtien liikuttaminen sujui nopeammin tabletilla, kun taas älypuhelimella elementit eivät liikkuneet aina reaaliaikaisesti. Samat animaatiot toimivat pöytäkoneella katsottuna moitteettomasti. Staattiset canvasilla piirretyt kuvat toimivat melko hyvin myös mobiililaitteilla.

3 Piirtäminen HTML5 Canvasille koodin kautta

Luvussa käsiteltävien aiheiden tarkoituksena on johdattaa HTML5 Canvasin piirtämisen perusteisiin. Luvun alussa käydään läpi esimerkki canvasin käyttöönotosta ja tutustutaan canvasin käyttämään koordinaatistoon. Tämän jälkeen syvennytään 2d-kontekstin ohjelmointirajapintaan käytännön esimerkein. Lopuksi käydään läpi erilaisia JavaScript-kirjastoja, jotka on tehty helpottamaan canvasin käyttöä sekä laajentamaan 2d-kontekstin omia metodeja ja ominaisuuksia.

3.1 HTML5 Canvasin käyttöönotto

Canvasin voi ottaa monella eri tavalla käyttöön, mutta jotkin perusasiat säilyvät samoina. Kuviossa 7 on esitetty yksi malli, jossa on hyödynnetty Rowellin (2012a) tekemää esimerkkiä. Koodissa ei ole mitään ylimääräistä, joten se sopii hyvin esimerkiksi sen yksinkertaisuutensa vuoksi. Esimerkissä on tavallinen HTML5-pohja, johon on lisätty canvas-elementti ja JavaScriptiä. Canvas-tägi laitetaan haluttuun kohtaan body-tägin sisälle. Canvas-elementille määritetään jokin id-arvo sekä leveys (width) ja korkeus (height). (Rowell 2011, 3; Rowell 2012a.) Kuviossa 7 canvas-elementille on annettu id-arvoksi minunCanvas. On hyvä huomioida, että JavaScript erottaa isot ja pienet kirjaimet. Esimerkiksi minunCanvas on eri asia kuin minuncanvas. Canvasin leveydeksi on määritetty 600 pikseliä ja korkeudeksi 300 pikseliä. Korkeus ja leveys määrittävät piir-

toalueen koon. Jos leveyttä ja korkeutta ei määritetä, automaattinen leveys on 300 pikseliä ja korkeus 150 pikseliä, kuten mainittiin luvussa 2.2.

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML5 Canvasin käyttöönotto</title>
</head>
<body>
    <canvas id="minunCanvas" width="600" height="300"></canvas>

    <script>
    var canvas = document.getElementById('minunCanvas');
    var context = canvas.getContext('2d');

    // Piirrä tähän

    </script>

</body>
</html>
```

Kuvio 7. HTML5 Canvasin käyttöönottoesimerkki.

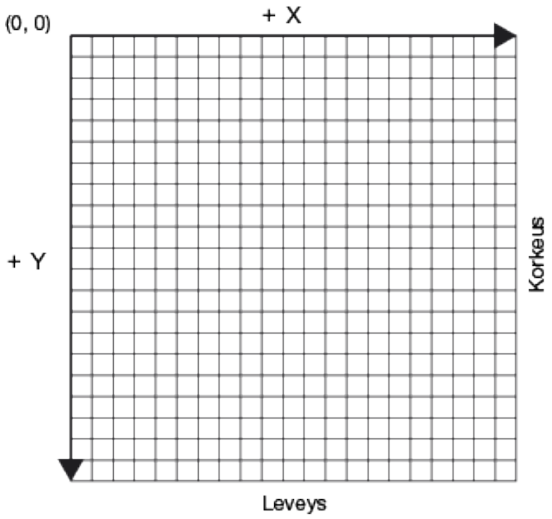
Kuviossa 7 script-tägit on laitettu body-osan loppuun. Muita vaihtoehtoja on esimerkiksi laittaa script-tägit head-osioon tai tehdä ulkoinen JavaScript-tiedosto. Canvas saadaan käyttöön getElementById()-metodilla. Document.getElemenById etsii canvas-elementin dokumenttioliomallista (DOM). (Pilgrim 2013; Rowell 2011, 3.) Id on yksilöllinen tunniste, joten metodin on helppo löytää elementti, joka on tässä tapauksessa minunCanvas. Var tarkoittaa muuttujaa ja var canvas muuttujaa, jonka nimi on canvas.

Kun canvas-elementti on löydetty dokumenttioliomallista, kutsutaan canvasin kontekstia getContext()-metodilla. 2d-kontekstia kutsuessa sulkuihin kirjoitetaan 2d-merkkijono kuvion 7 mukaisesti. (Pilgrim 2013.) Merkkijonojen, kuten 2d ja minunCanvas, molemmin puolin voidaan käyttää puolilainausmerkkiä (’ ’) tai lainausmerkkiä (” ”).

Kuvion 7 koodi ei näytä verkkosivuilla vielä mitään. Jos halutaan canvas-elementille reunat, ne voidaan määritellä normaaliin tapaan Css-tyylillä. Canvasille piirtäminen sen sijaan tapahtuu JavaScriptin kautta, kuten on aiemmin mainittu luvussa 2.2. (Pilgrim 2013.)

3.2 HTML5 Canvasin käyttämä koordinaatisto

2d-konteksti käyttää karteesista koordinaatistoa (Hawkes 2011, 58; WHATWG 2013). Karteesinen koordinaatisto on tavallisimmin käytetty koordinaatisto (Jyväskylä 2013). Canvasin kohdalla koordinaatiston origo eli nollapiste (0, 0) sijaitsee koordinaatiston vasemmassa yläreunassa. Koordinaatisto sisältää x- ja y-akselit. Liikuttaessa oikealle x-akselin arvoa kasvatetaan. Kun halutaan liikkua koordinaatiossa alaspäin, kasvatetaan y-akselin arvoa. (Hawkes 2011, 58; Pilgrim 2013; WHATWG 2013.) Canvasin käyttämä koordinaatisto poikkeaa normaalista karteesista koordinaatiosta siten, että y-akselin arvot ovat käänteisesti positiivisia arvoja. Normaalisti alaspäin liikuttaessa arvot ovat negatiivisia. (Bradford & Haine 2011, 245.) Canvasin 2d-kontekstin käyttämää koordinaatistoa on havainnoitu kuviossa 8.



Kuvio 8. Canvasin käyttämä koordinaatisto.

```
context.fillRect(x, y, leveys, korkeus);
```

Kuvio 9. Esimerkki koordinaation hyödyntämisestä piirtämisessä. ent3

Koordinaatteja käytetään kuvioiden asemoimiseen. Koordinaatistossa oleva yksikkö vastaa useimmiten yhtä ruudulla olevaa pikseliä. Esimerkiksi koordinaatit (50, 40) sijaitsevat 50 pikseliä oikealla ja 40 pikseliä alhaalla origosta. Teräväpiirtonäytöissä yksikkö vastaa yleensä kahta yksikköä korkean laadun säilyttämiseksi. Esimerkkinä

koordinaatiston hyödyntämisestä on täytetty suorakulmio, joka voidaan piirtää koodilla context.fillRect(10, 10, 50, 20). Esimerkkikoodia havainnoidaan kuviossa 9. Kaksi ensimmäistä sulussa olevaa lukua tarkoittavat x- ja y-arvoja, eli suorakulmion yläreuna sijaitsee oikealla ja alhaalla 10 pikseliä origosta. Seuraavat kaksi arvoa tarkoittavat suorakulmion leveyttä ja korkeutta. Leveys siirtyy leveyssuunnassa x- ja y-sijainnista x-akselin mukaisesti, ja korkeus vastaavasti alaspäin pystysuunnassa y-akselin mukaisesti. (Hawkes 2011, 58–60; WHATWG 2013.)

3.3 2d-kontekstin ohjelmointirajapinta

Kun lähdetään käyttämään HTML5 Canvasia, on hyvä tuntea piirtämisen perusteet (Rowell 2011, 8). Luvussa 2.2 kerrottiin, että piirtäminen canvasille tapahtuu 2d-kontekstin kautta. Tässä luvussa käsitellään tarkemmin 2d-kontekstin sisäänrakennettuja ominaisuuksia ja metodeja käytännön esimerkein. Kaikista metodeista ja ominaisuuksista ei löydy esimerkkejä, vaan tarkoituksena on esittää 2d-kontekstin käytön logiikka. Luvusta ei myöskään löydy kaikkien metodien ja ominaisuuksien tarkkoja syntakseja, mutta ne ovat saatavilla internetistä esimerkiksi W3C:n HTML Canvas 2D Context -dokumentaatiosta. Luvun esimerkit on tehty pääasiallisesti valmista aineistoa soveltaen ja kokeilemalla.

3.3.1 Polut ja suorakulmiot

2d-kontekstilla pystytään toteuttamaan erilaisia viivoja, polkuja, kurveja ja suorakulmioita. Viivoja, polkuja ja kaaria yhdistämällä voidaan piirtää erilaisia kuvioita. Suorakulmion piirtämiseen on oma JavaScript-käskynsä. (Rowell 2011, 7, 28; Rowell 2012b.) Taulukossa 1 on suorakulmioiden ja neliöiden piirtämiseen tarkoitettuja metodeja. Taulukossa 2 on esitetty erilaisia polkujen ja kurvien piirtämiseen tarkoitettuja metodeja. Viivojen erilaisia tyylejä käsitellään luvussa 3.3.2.

Taulukko 1. Metodeja täytettyjen ja ääriviivallisten suorakulmioiden piirtämiseen.

Suorakulmiot	
Metodi	**Kuvaus**
strokeRect()	Piirtää ääriviivallisen suorakulmion.
fillRect()	Piirtää täytetyn suorakulmion.

rect()	Luo suorakulmion. Vaatii stroke()- tai fill()-metodin piirtämiseen.
clearRect()	Poistaa pikselit suorakulmionmalliselta alueelta.

Taulukko 2. Metodeja polkujen ja kurvien piirtämiseen.

Polut	
Metodi	**Kuvaus**
fill()	Täyttää kuvion määritetyllä tyylillä.
stroke()	Piirtää varsinaisen ääriviivan määritetyllä tyylillä.
clip()	Leikkaa määritetyn alueen canvasista. Leikkaamisen jälkeen voidaan piirtää ainoastaan leikatulle alueelle.
beginPath()	Aloittaa uuden polun.
lineTo()	Lisää uuden pisteen ja luo linjan tästä pisteestä aiemmin määritettyyn pisteeseen.
moveTo()	Siirtää polun määrättyyn pisteeseen canvasilla.
closePath()	Lopettaa polun ja luo polun aloituspisteeseen.
arc()	Luo kaaren. Tämän avulla piirretään esimerkiksi ympyröitä.
arcTo()	Luo kaaren kahden tangentin väliin.
bezierCurveTo()	Luo neliöllisen Bézier-käyrän.
quadraticCurveTo()	Luo kuutiollisen Bézier-käyrän.
isPointInPath()	Metodi palauttaa arvon tosi, jos määritetty piste on polulla.

Luvussa 3.2 oli esimerkki täytetyn suorakulmion piirtämisestä. Samalla tavalla voidaan piirtää myös ääriviivallinen suorakulmio tai neliö käyttämällä strokeRect()-metodia fillRect()-metodin sijaan. Käytettäessä rect()-metodia tarvitaan stroke()- tai fill()-metodia piirtämään suorakulmio näkyviin canvasille. Fill()-metodi täyttää kuvion määritetyllä tyylillä ja stroke()-metodi piirtää kuviolle ääriviivat. (Rowell 2011, 28–29, 32.) Tyylejä käsitellään luvussa 3.3.2. Kuviossa 10 on koodiesimerkki ääriviivallisen neliön piirtämisestä. Kuvio 11 havainnollistaa koodin lopputulosta kuvana. StrokeRect()-metodi saa parametreiksi eli arvoiksi suorakulmion vasemman yläreunan pisteen x- ja y-akselin arvot sekä neliön leveyden ja korkeuden.

Viivan piirtäminen muodostuu useammasta metodista. BeginPath()-metodi määrittää uuden polun alkavaksi. MoveTo()-metodi sijoittaa viivan alkupisteen haluttuun kohtaan ja lineTo()-metodi piirtää suoran viivan alkupisteestä uuteen pisteeseen. MoveTo()- ja

lineTo()-metodit ottavat parametreiksi koordinaatiston x- ja y-arvot. Nämä kaikki edellä mainitut metodit eivät vielä näytä viivaa canvasilla. Tarvitaan stroke()-metodi, jolla viiva saadaan varsinaisesti näkyväksi. (Rowell 2013b.) Kuviossa 10 on koodiesimerkkejä viivojen ja polkujen piirtämisestä, mitä on havainnoitu kuvallisesti kuviossa 11. Esimerkin kolmio muodostuu poluista, joille on luotu ensimmäinen piste moveTo()-metodilla. Kolmiolle on tehty kaksi viivaa käyttämällä LineTo()-metodeita ja polku on suljettu käyttämällä closePath()-metodia. ClosePath()-metodi lopettaa polun ja luo polun takaisin aloituspisteeseen, joka luotiin moveTo()-metodilla.

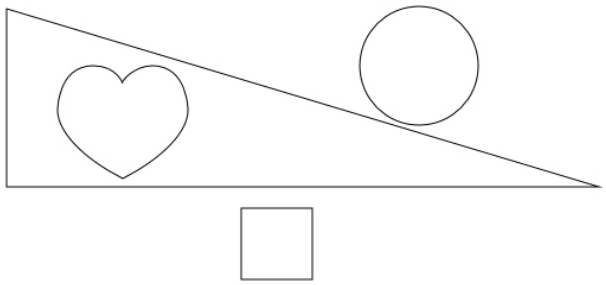
```
//Kolmio
context.beginPath();
context.moveTo(52,52);
context.lineTo(52,202);
context.lineTo(552,202);
context.closePath();
context.stroke();

//Ympyrä
context.beginPath();
context.arc(400,100,50,0,2*Math.PI);
context.stroke();

//Sydän
context.beginPath();
context.moveTo(150,115);
context.bezierCurveTo(150,115,145,100,125,100);
context.bezierCurveTo(95,100,95,137.5,95,137.5);
context.bezierCurveTo(95,155,115,177,150,195);
context.bezierCurveTo(185,177,205,155,205,137.5);
context.bezierCurveTo(205,137.5,205,100,175,100);
context.bezierCurveTo(160,100,150,112,150,113.5);
context.stroke();

//Neliö
context.strokeRect(250,220,60,60);
```

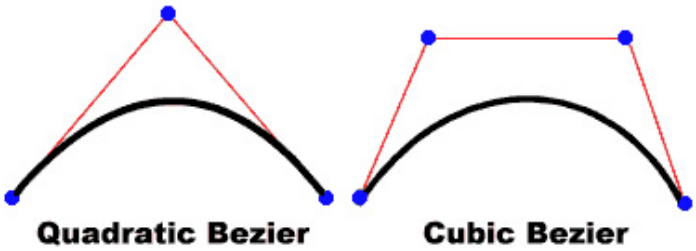
Kuvio 10. Esimerkkejä muotojen piirtämisestä.



Kuvio 11. Kuvion 10 koodin lopputulos piirroksina.

2d-konteksti tarjoaa metodeja kaarien toteuttamiseen. Arc()-metodin avulla voidaan piirtää muun muassa ympyröitä ja arcTo()-metodi soveltuu käytettäväksi esimerkiksi pyöristettyyn kulmaan kahden kohtisuoran viivan välillä. Kuvioon 10 on piirretty esimerkkiympyrä. Arc()-metodi saa parametreiksi x- ja y-arvot koordinaatistossa, ympyrän säteen, aloituskulman ja viimeiseksi lopetuskulman. JavaScriptissä oleva Math-olio mahdollistaa matemaattisten laskutoimitusten suorittamisen. Se sisältää matemaattisia vakioita, kuten esimerkissä olevan piin (π), ja metodeja. Lopetuskulman arvo 2 luo kokonaisen ympyrän, kun taas arvo 1.5 kolmasosaympyrän ja 1 puolikkaan ympyrän (kehän pituus lasketaan kaavalla 2 * π r, jossa r on ympyrän säde). (Rowell 2011, 11–12; W3 Schools 2013a; W3 Schools 2013b; W3 Schools 2013c.) BeginPath()- ja stroke()-metodit toimivat samalla tavalla, kuten mainittiin edellisessä kappaleessa. BeginPath()-metodi aloittaa polun ja stroke()-metodi piirtää kuvion näkyväksi.

QuadraticCurveTo() ja BezierCurveTo() -metodeita voidaan käyttää Bézier-käyrien piirtämiseen. Bézier-käyrät määritetään pisteillä, jotka sijaitsevat kuvitteellisissa tangenteissa. QuadraticCurveTo()-metodi luo kuutiollisen Bézier-käyrän (quaratic Bézier), joka määritetään kahdella tukipisteellä ja yhdellä kontrollipisteellä. BezierCurveTo()-metodilla taas luodaan neliöllinen Bézier-käyrä (cubic Bézier), joka määritetään kahdella tukipisteellä ja kahdella kontrollipisteellä. (Geary 2012, 4654; Hwkes 2011, 118–120.) Kuvio 12 havainnollistaa kuutiollisen- ja neliöllisen Bézier-käyrän eroa.



Kuvio 12. Kuutiollinen ja neliöllinen Bézier-käyrä (Tondeur 2008).

Kuvion 10 koodissa on esimerkki sydämen piirtämisestä neliöllistä Bézier-käyrää käyttäen. Sydämen muoto muodostuu useista pienistä Bézier-käyristä. MoveTo()-metodilla luodaan ensimmäinen piste koordinaatistossa. Tämän jälkeen BezierCurveTo()-metodeille annetaan parametreiksi kaksi kontrollipistettä x- ja y-akselilla. Viimeiset pa-

rametrit ovat viimeisen pisteen koordinaatit x- ja y-akselilla. Koodin lopputulos näkyy kuvasta 11.

3.3.2 Tyylit

2d-konteksti sisältää kuvioiden ja polkujen tyylittelyyn tarkoitettuja metodeja ja ominaisuuksia. Kuviolle voi määrittää esimerkiksi värin, liukuvärin, kuvioinnin (pattern), varjostuksen, läpinäkyvyyden ja kuvioiden asettelun. Viivoille voi värien lisäksi määrittää leveyden ja muodon. (W3 Schools 2013a.) Taulukossa 3 on esitetty ominaisuuksia ja metodeja värien, kuviointien ja varjostuksien luomiseen. Taulukossa 4 on ominaisuuksia viivojen tyylille ja taulukossa 5 ominaisuuksia läpinäkyvyydelle ja kuvioiden asettelulle.

Taulukko 3. Ominaisuuksia ja metodeja kuvioiden tyyleihin.

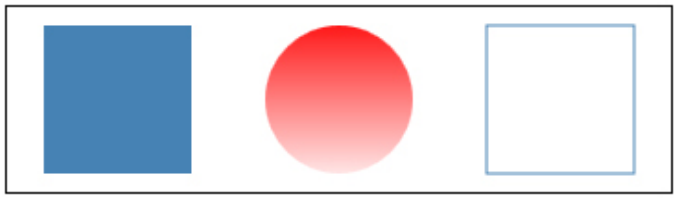
Värit, tyylit ja varjot	
Ominaisuus	**Kuvaus**
fillstyle	Määrittää värin, kuvioinnin (pattern) tai liukuvärin, jota käytetään kuvion tai tekstin täyttämiseen.
strokeStyle	Määrittää värin, kuvioinnin tai liukuvärin viivalle, ääriviivalle tai tekstin ääriviivoille.
shadowBlur	Määrittää varjoalueen laajuuden. Varjon oletusarvona on nolla. Varjon kokoa voidaan lisätä arvoa suurentamalla.
shadowOffsetX	Määrittää varjon horisontaalisen etäisyyden kuviosta.
shadowOffsetY	Määrittää varjon vertikaalisen etäisyyden kuviosta.
shadowColor	Määrittää varjon värin. Väri on useimmiten läpikuultava.
Metodi	**Kuvaus**
createLinearGradient()	Luo lineaarisen liukuvärin.
createRadialGradient()	Luo radiaalisen liukuvärin.
addColorStop()	Määrittelee liukuvärin värin ja sijainnin, johon väri loppuu.
createPattern()	Luo kuvioinnin toistamalla määritettyä kuviota määritetyssä suunnassa.

FillStyle ja strokeStyle ovat keskeisimpiä ominaisuuksia värien määrittämisessä. FillStyle-ominaisuus määrittää kuvioiden täytteen, joka voi olla väri, liukuväri tai kuviointi. Viivalle tai kuvion ääriviivoille voidaan määrittää väri, liukuväri tai kuviointi strokeStyle-ominaisuudella. (Rowell 2011, 32; W3 Schools 2013d; W3 Schools 2013e.) Kuvion 13

koodissa ensimmäisen neliön väri on määritetty fillStyle-ominaisuudella siniseksi heksadesimaali väriarvolla. Toisen neliön ääriviivat on määritetty samanväriseksi kuin ensimmäisen neliön täyteväri strokeStyle-ominaisuudella. Kuvion 13 koodin lopputulosta on havainnoitu kuviossa 14.

```
18
19          //Neliö
20          context.fillStyle="#4682b4";
21          context.fillRect(20,10,80,80);
22
23          //Ympyrä
24          var gradient = context.createLinearGradient(0,0,0,100);
25          gradient.addColorStop(0, "red");
26          gradient.addColorStop(1, "white");
27
28          context.fillStyle = gradient;
29          context.beginPath();
30          context.arc(180,50,40,0,2*Math.PI);
31          context.fill();
32
33          //Neliö
34          context.strokeStyle="#4682b4";
35          context.strokeRect(260,10,80,80);
36
```

Kuvio 13. Värillä täytetty neliö, lineaarisella liukuvärillä täytetty ympyrä ja ääriviivallinen neliö.

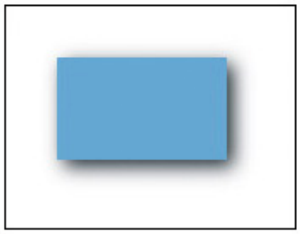


Kuvio 14. Kuvion 13 koodin lopputulos.

Lineaarinen liukuvärjäys luodaan käyttämällä createLinearGradient()-metodia ja radiaalinen liukuvärjäys createRadialGradian()-metodilla. Kuvion 13 ympyrä on täytetty lineaarisella liukuvärillä (kuvion 13 ympyrä on havainnoitu kuviossa 14). Esimerkissä on luotu muuttuja nimeltään gradient, joka saa arvokseen context-muuttujaan liittyvän createLinearGradient()-metodin. CreateLinearGradient-metodi saa parametreiksi liukuvärin alku- ja loppupisteet x- ja y-akselilla. AddColorStop()-metodilla määritetään liukuvärin väri sekä liukuvärin aloitus- ja loppumiskohdat. Aloitus- ja loppumiskohtien arvo on nollan ja yhden välillä. AddColorStop()-metodeja voi käyttää useampia, mikäli halutaan luoda monisävyisempi liukuvärjäys. FillStyle-ominaisuudeksi eli kuvion täytön tyy-

liksi määritetään edellä luotu gradient-muuttuja. Fill()-metodilla täytetään kuvio näkyväksi. (Hawkes 2011, 113–114; Rowell 2013c.)

```
19          //Suorakulmio + varjostus
20          context.rect(30, 30, 100, 60);
21          context.fillStyle = '#65a7d2';
22          context.shadowColor = '#14212a';
23          context.shadowBlur = 10;
24          context.shadowOffsetX = 4;
25          context.shadowOffsetY = 4;
26          context.fill();
```



Kuvio 15. Varjostuksen luominen.

2d-konteksti mahdollistaa varjojen luomisen erilaisilla ominaisuuksilla. ShadowColor-ominaisuudella määritetään varjon väri ja shadowBlur-ominaisuudella varjon sumeuden (blur) laajuus. ShadowOffsetX- ja shadowOffsetY -ominaisuudet määrittävät varjon etäisyyden x- ja y-akselin suunnassa pikseleinä kuvion vasemmalta puolelta. ShadowOffsetX- ja shadowOffsetY -ominaisuudet voivat saada myös negatiivisia arvoja, jolloin varjo liikkuu oikealta vasemmalle ja arvot nolla, jolloin varjo jää kuvion taakse. (Hawkes 2011, 109–111; Jenkov 2013.) Kuviossa 15 on esimerkki varjon käyttämisestä.

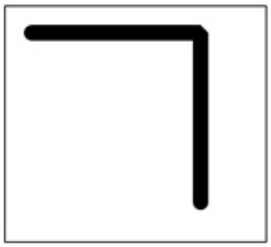
Taulukko 4. Ominaisuuksia viivojen tyylittelyyn.

Viivojen tyylit	
Ominaisuus	**Kuvaus**
lineWidth	Määrittää viivan leveyden.
lineJoin	Määrittää kahden viivan muodostaman kulman muodon. Muoto voi olla terävä (miter), pyöreä (round) tai viisto (bevel).
lineCap	Määrittää viivan pään muodon. Muoto voi olla pyöreä (round), suora (butt) tai neliö (square), joka lisätään viivan päähän.
miterLimit	Määrittää kahden viivan muodostaman kulman etäisyyden.

Viivoille voidaan määrittää ominaisuuksilla viivan leveys, viivojen muodostaman kulman muoto, kulman etäisyys ja viivanpään muoto. Kuvion 16 esimerkissä viivojen luominen on aloitettu beginPath()-metodilla, jonka jälkeen on määritetty viivan leveydeksi 12 pikseliä lineWidth-ominaisuudella, viivanpään muoto pyöreäksi lineCap-ominaisuudella ja kahden viivan muodostama kulma viistoksi lineJoin-ominaisuudella. (Geary 2012, 4062, 4081, 4099; W3 Schools 2013a.) Tyylien jälkeen on määritetty viivan alkamispis-

te moveTo()-metodilla, kaksi viivaa koordinaatistossa lineTo()-metodeilla ja piirretty viivat näkyviksi stroke()-metodilla.

```
19      context.beginPath();
20      context.lineWidth=12;
21      context.lineCap="round";
22      context.lineJoin="bevel";
23      context.moveTo(20,20);
24      context.lineTo(150,20);
25      context.lineTo(150, 150);
26      context.stroke();
```

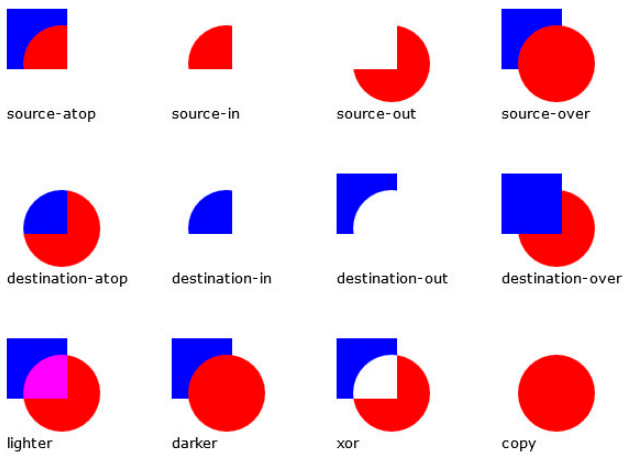


Kuvio 16. Esimerkki viivojen luomisesta.

Taulukko 5. Ominaisuuksia läpinäkyvyyteen ja sommitteluun.

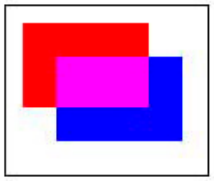
Läpinäkyvyys ja sommittelu	
Ominaisuus	**Kuvaus**
globalCompositeOperation	Määrittää kuinka uusi kuvio asettuu olemassa olevan kuvan kanssa.
globalAlpha	Määrittää Kuvion läpinäkyvyyden. Arvo on nollan ja yhden välillä. Nolla on täysin läpinäkyvä ja yksi täysin näkyvä.

GlobalCompositeOperation-ominaisuus määrittää, miten uusi sisältö piirretään canvasille. Erilaisia vaihtoehtoja on 12, mutta normaalisti uudet elementit asettuvat vanhojen elementtien päälle (kuvion 17 source-over). Jo olemassa olevaa kuvaa kutsutaan nimellä source ja uutta kuvaa nimellä destination. (Rowell 2013d; W3 Schools 2013f.) Kuvion 18 esimerkissä siniselle suorakulmiolle on annettu globalCompositeOperation-ominaisuus lighter.



Kuvio 17. GlobalCompositeOperation-ominaisuuden 12 eri vaihtoehtoa (Rowell 2013d).

```
19      context.fillStyle="red";
20      context.fillRect(10,10,75,50);
21      context.fillStyle="blue";
22      context.globalCompositeOperation="lighter";
23      context.fillRect(30,30,75,50);
```



Kuvio 18. Esimerkki globalCompositeOperation-ominaisuudesta.

GlobalAlpha-ominaisuudella voidaan määrittää elementtien läpinäkyvyyttä. Läpinäkyvyyttä säädetään arvolla, joka sijoittuu nollan ja yhden väliin. Nolla on täysin läpinäkyvä ja arvo yksi näkyvä. (Rowell 2013e.) Jos esimerkiksi halutaan elementin olevan 70 % läpinäkyvä, niin globalAlphalle määritetään arvoksi 0.7.

3.3.3 Teksti

Canvas tarjoaa erilaisia metodeja ja ominaisuuksia tekstin kirjoittamiseen. Tekstille voidaan määrittää kirjasintyyppi, koko, tyyli ja väri. Tekstille löytyy myös muita ominaisuuksia ja metodeja, kuten ääriviivallinen teksti, keskittäminen ja tekstin leveyden mittaaminen pikseleinä. (Rowell 2011, 20–21; W3 Schools 2013a.) Tekstin ominaisuudet ja metodit on lueteltu taulukossa 6.

Taulukko 6. Tekstin piirtämiseen ja muotoiluun tarkoitettuja metodeja ja ominaisuuksia.

Teksti	
Ominaisuus	**Kuvaus**
font	Määrittelee fontin ja muita fontin ominaisuuksia, kuten koon ja tyylin (esimerkiksi bold tai italic).
textBaseline	Määrittää tekstin vertikaalisen sijoittamisen, esimerkiksi ylhäällä, keskellä tai alhaalla.
textAlign	Määrittää tekstin horisontaalisen sijoittamisen, esimerkiksi vasemmalla, keskellä tai oikealla.
Metodi	**Kuvaus**
fillText()	Piirtää tekstin annettuihin koordinaatteihin.
strokeText()	Piirtää ääriviivallisen tekstin annettuihin koordinaatteihin.
measureText()	Mittaa määrätyn tekstin pituuden pikseleinä.

Tekstin kirjoittamiseen käytetään pääasiallisesti fillText()-metodia ja font-ominaisuutta. Ensimmäisenä määritetään kirjasintyypin tyyli (normal, italic, bold), kirjasintyypin koko

ja kirjasintyyppi font-ominaisuudella. Jos kirjasintyypin tyyliä ei määritetä, tyylin oletusarvo on normaali (normal). FillText()-metodille annetaan parametreiksi kirjoitettava teksti sekä x- ja y-akselin arvot tekstin aloituspisteelle. Parametriksi voi asettaa myös halutessaan tekstin maksimileveyden pikseleinä. (Rowell 2013f; W3 Schools 2013g.) Kuviossa 19 on esimerkki tekstin piirtämisestä canvasille.

```
18
19      context.font = "bold 30px georgia";
20      context.fillStyle = "#0355fa";
21      context.fillText("Lorem ipsum",20,60);
22
```

Kuvio 19. Esimerkki tekstin piirtämisestä canvasille.

Tekstin oletusvärinä on musta. Tekstin väriä voi muuttaa fillStyle-ominaisuudella antamalla arvoksi värin nimen, heksadesimaalivärikoodin tai rgb-arvot. (W3 Schools 2013e.) Värin lisäksi fillStyle-ominaisuudella voi asettaa tekstille liukuvärin tai kuvioinnin, kuten mainittiin luvussa 3.3.2.

3.3.4 Siirtymät

Taulukko 7. Metodeja siirtymiin.

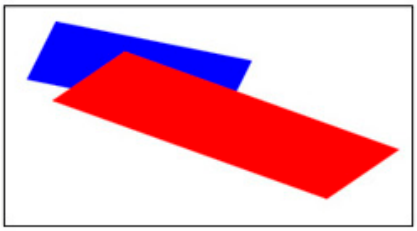
Siirtymät	
Metodi	**Kuvaus**
transform()	Korvaa mahdollisen edellisen transformaatio-matriisin uudella. Mahdollistaa kuvioiden skaalaamisen (scale), pyörittämisen (rotate), vääristämisen (skew) ja siirtämisen (move).
rotate()	Pyörittää kuviota.
scale()	Skaalaa kuviota pienemmäksi tai isommaksi.
translate()	Muuttaa origon paikkaa canvasilla.
setTransform()	Luo uuden transformaatio-matriisin. Mahdollistaa kuvioiden skaalaamisen (scale), pyörittämisen (rotate), vääristämisen (skew) ja siirtämisen (move).

2d-konteksti sisältää erilaisia metodeja siirtymiin (transformation). Metodeilla on mahdollista muuttaa elementin paikkaa, skaalata ja pyörittää elementtiä sekä tehdä omia transformaatio-yhdistelmiä. Transform()-metodi korvaa aina edellisen siirtymän uudella, mutta säilyttää myös edelliset asetukset eli lisää uudet parametrit vanhojen päälle. Uu-

den siirtymän voi luoda käyttämällä setTransform()-metodilla, jolloin edellinen mahdollinen siirtymä ei vaikuta uuden elementin siirtymään. (W3 Schools 2013a; W3 Schools 2013h; W3 Schools 2013i.)

Rotate()-, scale()- ja translate() -metodeja käytetään sellaisenaan lisäämään kuviolle tietty transformaatio. Rotate()-metodi saa parametriksi pyörimiskulman, esimerkiksi viisi astetta kirjoitetaan context.rotate(5*Math.PI/180). Scale()-metodi saa parametreiksi skaalattavan leveyden ja korkeuden, esimerkiksi context.scale(2, 2) skaalaa kuvion kaksi kertaa isommaksi (0.5 = 50 %, 1 = 100 %, 2 = 200 % jne.). Translate()-metodi saa parametreiksi uuden sijainnin x- ja y-arvot koordinaatistolla, esimerkiksi context.translate(20, 0) siirtää kuviota horisontaalisesti 20 pikseliä kuvion alkuperäisestä x-koordinaatista. (Hawkes 2011, 92–98; W3 Schools 2013j; W3 Schools 2013k; W3 Schools 2013l.)

```
19      context.transform(1.5,0.3,-0.5,1,35,10);
20      context.fillStyle="blue";
21      context.fillRect(0,0,90,40);
22
23      context.transform(1.5,0.3,-0.5,1,35,10);
24      context.fillStyle="red";
25      context.fillRect(0,0,90,40);
```



Kuvio 20. Esimerkki transform()-metodin käytöstä.

Kuviossa 20 on esimerkki transform()-metodin käytöstä. Transform()-metodin ensimmäinen parametri skaalaa ja toinen vääristää kuviota horisontaalisesti, kolmas parametri vääristää ja neljäs skaalaa kuviota vertikaalisesti, toiseksi viimeinen parametri siirtää kuviota horisontaalisesti ja viimeinen vertikaalisesti (W3 Schools 2013h). Koska koodissa on käytetty transform()-metodia, seuraava elementti perii edellisen elementin siirtymät. Tämän vuoksi toinen elementti on erilainen kuin ensimmäinen, vaikka koodit ovat väriä lukuun ottamatta samat. SetTransform()-metodia käytetään koodissa saman lailla kuin transform()-metodia, mutta setTransform() ei peri edellisen elementin siirtymiä vaan aloittaa uuden siirtymämatriisin, kuten aiemmin mainittiinkin.

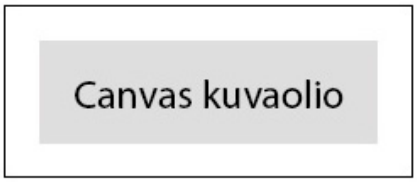
3.3.5 Kuvan piirtäminen

Taulukko 8. Metodi kuvan piirtämiseen canvasille.

Kuvan piirtäminen	
Metodi	**Kuvaus**
drawImage()	Piirtää kuvan, canvasin tai videon canvasiin.

Kuvan, videon ja toisen canvasin piirtäminen canvasille on mahdollista drawImage()-metodilla. Metodin avulla kuvaa voi rajata, säätää kuvan kokoa ja asemoida haluttuun kohtaan canvasille. Kuviossa 21 on piirretty kuva canvasille drawImage()-metodia käyttäen. DrawImage()-metodi vaatii kuva-olion, joka on luotu kuvion 21 koodin ensimmäisellä rivillä. Onload-funktiolla varmistetaan, että kuva latautuu ennen kuin se piirretään canvasille. Funktion sisälle kirjoitetaan drawImage()-metodi, joka saa parametreiksi minimissään kuva-olion sekä piirretyn kuvan vasemman yläreunan x- ja y-pisteen koordinaatit. DrawImage()-metodin muita valinnaisia parametreja ovat leikattavan alueen x- ja y-koordinaatit, leikatun kuvan leveys ja korkeus sekä leveys ja korkeus kuvan skaalaamista varten. Koodin viimeisessä rivissä määritetään kuvan lähde. (Hawkes 2011, 125–129; Rowell 2013g.) Esimerkissä kuva on samassa kansiossa html-tiedoston kanssa.

```
18
19      var kuvaOlio = new Image();
20      kuvaOlio.onload = function() {
21          context.drawImage(kuvaOlio, 20, 20);
22      };
23      kuvaOlio.src = 'kuva.png';
```



Kuvio 21. Esimerkki DrawImage()-metodin käytöstä.

3.3.6 Pikselimanipulaatio

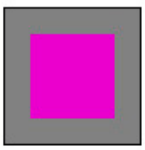
Pikselimanipulaatio (pixel manipulation) mahdollistaa pääsyn jokaiseen yksittäiseen canvasilla olevaan pikseliin. Jokaisesta pikselistä on saatavilla tietoa väristä ja alfa-arvosta (alpha). Pikselimanipulaation kautta canvasilla olevia kuvia tai kuvioita pystyy manipuloimaan muun muassa vaihtamalla kuvan värit, esimerkiksi käänteisiksi tai mustavalkoiseksi. Kuvioista voi myös kopioida tietyn kokoisia osia ja toistaa niitä canvasille. (Hawkes 2011, 136–137; Rowell 2013h; W3 Schools 2013m; W3 Schools 2013n.) Taulukossa 9 on lueteltu pikselimanipulaatioon tarkoitettuja ominaisuuksia ja metodeja.

Taulukko 9. Ominaisuuksia ja metodeja pikselimanipulaatioon.

Pikselimanipulaatio	
Ominaisuus	**Kuvaus**
width	Palauttaa ImageData-olion leveyden.
height	Palauttaa ImageData-olion korkeuden.
data	Palauttaa olion, joka sisältää kuvatietoa määritetystä ImageData-oliosta.
Metodi	**Kuvaus**
createImageData()	Luo uuden tyhjän ImageData-olion.
getImageData()	Palauttaa ImageData-olion, joka kopioi pikselien tiedot määritellyltä suorakulmionmuotoiselta alueelta.
putImageData()	Asettaa määritetyn ImageData-olion kuvan tiedot takaisin canvasille.

CreateImageData()-metodi luo uuden tyhjän ImageData-olion. ImageData-olio tarkoittaa määritettyä suorakulmionmallista aluetta canvasilla, mikä sisältää tietoa jokaisesta yksittäisestä pikselistä alueen sisällä. Jokainen pikseli sisältää RGBA-arvot eli punaisen, vihreän, sinisen ja alfakanavan. RGBA-arvot vaihtelevat 0-255 välillä. Väri-informaatio muodostaa taulukon (array) ja se on tallennettu ImageData-olion data-ominaisuuteen. GetImageData()-metodilla voidaan kopioida määritetty alue canvasilta. Väri-informaation muokkaamisen jälkeen muutokset voidaan asettaa takaisin canvasille putImageData()-metodilla. (Hawkes 2011, 137–138; W3 Schools 2013m; W3 Schools 2013n.)

```
19      context.fillStyle = 'grey';
20      context.fillRect(0, 0, 80, 80);
21      var imgData=context.createImageData(50,50);
22      for (var i=0;i<imgData.data.length;i+=4)
23        {
24        imgData.data[i+0]=200; //punainen - Red
25        imgData.data[i+1]=0; //vihreä - Green
26        imgData.data[i+2]=200; //sininen - Blue
27        imgData.data[i+3]=255; //alfa - Alpha
28        }
29      context.putImageData(imgData,15,15);
```



Kuvio 22. Esimerkki yksinkertaisesta pikselimanipulaatiosta.

Kuviossa 22 näkyy esimerkki yksinkertaisesta pikselimanipulaatiosta. Harmaanväriseen 80 x 80 pikselin canvasiin on muokattu 50 x 50 pikselin värillinen alue createImageData()-metodia käyttäen. RGBA-arvot muodostavat taulukon ja ne voi käydä läpi for-

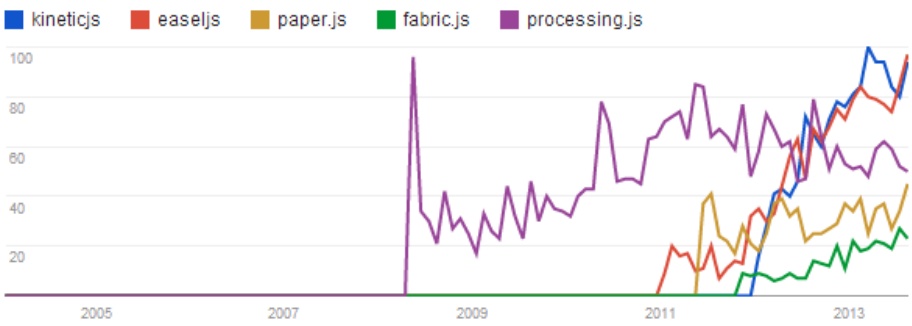
silmukalla (W3 Schools 2013m). PutImageData()-metodi lisää muutetut arvot canvasille haluttuun kohtaan x- ja y-akselille (W3 Schools 2013n).

3.4 JavaScript-kirjastoja HTML5 Canvasille

HTML5 Canvasille on kehitetty useita JavaScript-kirjastoja laajentamaan 2d-kontekstin ominaisuuksia ja metodeja sekä helpottamaan canvasin käyttöä. 2d-kontekstin valmiit metodit ja ominaisuudet sopivat hyvin yksinkertaisen grafiikan piirtämiseen, mutta canvasin käyttö hankaloituu käsitellessä useampia elementtejä samanaikaisesti ja suorituskyky heikkenee muun muassa interaktiivisten animaatioiden kohdalla. JavaScript-kirjastot tehostavat canvasin suorituskykyä ja helpottavat erityisesti canvasin elementtien animoimisessa. (Design your way 2013; Vepsäläinen 2013.)

Mielestäni JavaScript-kirjastojen käytössä on myös haasteellisia puolia. Kirjastoja käytettäessä täytyy usein opetella uudenlainen syntaksi, joka vaihtelee kirjaston mukaan. HTML5 Canvasille on olemassa monia kirjastoja, joten eri kirjastojen syntaksien läpikäyminen ja kirjaston etsiminen omiin tarpeisiin saattaa viedä paljon aikaa. Kirjastot eivät myöskään käytä aina tavallista JavaScriptiä. Muita vaihtoehtoja ovat muun muassa jQuery ja CoffeeScript. Esimerkiksi paper.js niminen JavaScript-kirjasto käyttää omaa JavaScriptistä johdettua PaperScript-kieltä (Paper.js 2013).

Rowellin (2013i) mukaan tämän hetken suosituimmat JavaScript-kirjastot HTML5 Canvasille olisivat KineticJS, EaselJS, Paper.js, Fabric.js ja Processing.js. Kuvion 23 mukaan KineticJS- ja EaselJS-kirjastojen suosio on kasvussa. Nämä kaksi kirjastoa ovat myös Googlen hakukoneessa korkeimmalla sijalla sekä käytettyjä muun muassa alan artikkeleissa ja keskustelufoorumeilla, kuten Stack Overflowssa. On kuitenkin huomioitava, että Rowell on yksi KineticJS-kirjaston kehittäjistä (KineticJS 2013a), kun mietitään lähteen luotettavuutta. Seuraavissa alaluvuissa käydään läpi tarkemmin KineticJS ja EaselJS -kirjastot sekä esitellään muita suosittuja kirjastoja lyhyesti.

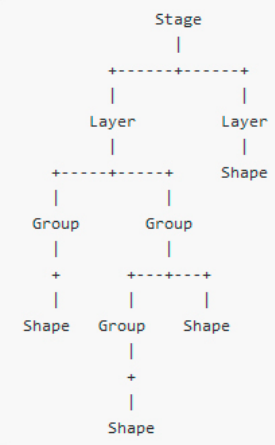


Kuvio 23. Käyttäjien kiinnostus JavaScript-kirjastoja kohtaan (Rowell 2013i).

3.4.1 KineticJS

KinecticJS -JavaScript-kirjasto tuo lisää ominaisuuksia ja metodeja canvasin käyttöön sekä lisää korkean suorituskyvyn animaatioihin. Ominaisuuksiin kuuluu muun muassa oliopohjainen ohjelmointirajapinta, kerrosten (layer) hyödyntäminen, animaatiot ja siirtymät, kustomoidut kuviot, raahaus- ja pudotus (drag and drop) toiminnot sekä tapahtumien käsittely mobiiliapplikaatioihin ja verkkosivuille. KineticJS-kirjasto sisältää myös paljon valmiita kuvioita, kuten suorakulmioita, ympyröitä, viivoja, polygoneja, tekstiä, kuvia ja SVG-polkuja. Kirjastoon kuuluu valmiita tapahtumia muun muassa hiirenkäsittelyyn, kosketusnäyttöön ja piirtämiseen. KineticJS on helppokäyttöinen henkilöille, jotka hallitsevat JavaScriptin, jQueryn, HTML:n ja CSS:n. (Rowell 2013j.)

KineticJS-kirjastoa käytettäessä muodostetaan stage-olio, johon määritetään kerroksia. Jokainen kerros voi sisältää esimerkiksi kuvion, ryhmän kuvioita tai ryhmän kuvioita ryhmän sisällä. Jokaiseen kerrokseen sisältyy kaksi canvasia. Toinen canvasista on näkyvä eli siinä voi olla esimerkiksi piirroksia ja tekstiä. Toinen canvasista on näkymätön ja se sisältää tapahtumia (events). (Rowell 2013j.) Kuvio 24 havainnollistaa KineticJS-kirjaston rakennetta.



Kuvio 24. KineticJS-kirjaston käyttämä rakenne (Rowell 2013j).

Kuviossa 25 on esimerkki täytetyn ääriviivallisen suorakulmion piirtämisestä KineticJS-kirjaston avulla. Koodiin on lisätty KineticJS-kirjasto script-tägin sisään. Kirjaston sisällä määritetään canvas-elementti ja 2d-konteksti, joten niitä ei tarvitse kirjoittaa koodiin erikseen. (Rowell 2013k.) Canvas-tägin sijaan html-tiedostoon määritetään div-elementti, joka saa id-arvokseen container. JavaScriptissä muodostetaan stage-olio, joka sisältää container divin sekä canvas alueen leveyden ja korkeuden. Tämän jälkeen luodaan kerros-olio (Kinetic.Layer). Kerroksen luomisen jälkeen luodaan suorakulmio-olio (Kinetic.Rect), jonka sisään määritetään x- ja y-akselin pisteet, suorakulmion leveys ja korkeus, täytteen ja ääriviivan värit sekä ääriviivan leveys. Lopuksi lisätään suorakulmio-olio kerrokseen ja kerros stage-olioon. Elementtien luomista KineticJS-kirjastolla käsitellään tarkemmin luvussa 4.3.2.

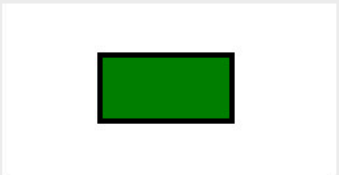
```
<body>
  <div id="container"></div>
  <script src="http://www.html5canvastutorials.com/libraries/kinetic-v4.7.3-beta.js"></script>
  <script defer="defer">
    var stage = new Kinetic.Stage({
      container: 'container',
      width: 578,
      height: 200
    });

    var layer = new Kinetic.Layer();

    var rect = new Kinetic.Rect({
      x: 239,
      y: 75,
      width: 100,
      height: 50,
      fill: 'green',
      stroke: 'black',
      strokeWidth: 4
    });

    // add the shape to the layer
    layer.add(rect);

    // add the layer to the stage
    stage.add(layer);
  </script>
</body>
```



Kuvio 25. Täytetty ääriviivallinen suorakulmio, joka on toteutettu KineticJS-kirjastolla.

3.4.2 EaselJS

EaselJS on avoimeen lähdekoodiin perustuva JavaScript-kirjasto, joka yksinkertaistaa monimutkaisemman grafiikan tekemistä ja interaktiivisuuden lisäämistä canvasille. EaselJS sisältää monia ominaisuuksia ja metodeja muun muassa erilaisten kuvioiden piirtämiseen, animaatioiden tekoon, efektien käyttöön ja kosketusnäytön hallintaan. EaselJS kuuluu osaksi CreateJs-kirjastokokoelmaa. CreateJS:n kirjastot toimivat erikseen tai niitä voidaan käyttää yhdessä. EaselJS-kirjaston lisäksi CreateJS sisältää muun muassa TweenJS:n monimutkaisempiin animaatioihin ja SoundJS:n helpottamaan audio-elementin käyttöä. CreateJS:n sponsoreina toimii muun muassa Adobe ja Microsoft. (CreateJS 2013a; CreateJS 2013b; CreateJS 2013c; CreateJS 2013d; CreateJS 2013e.)

EaselJS -JavaScript-kirjasto tarjoaa tutun ohjelmointirajapinnan Flashin tunteville kehittäjille, mutta käyttää kielenä JavaScriptiä. EaselJS sisältää muun muassa Flashista tutun hierarkkisen puurakenteen (hierarchical display list) ja apuluokkia (helper classes). EaselJS-kirjastoa käytettäessä luodaan stage niminen säiliö canvasille KineticJS-kirjaston tavoin (luvussa 3.4.1), minkä jälkeen voidaan lisätä erilaisia elementtejä canvasille. (CreateJS 2013e.) Flash on vektorigrafiikkapohjainen multimediaformaatti, jota käytetään muun muassa animaatioiden, käyttöliittymien ja bannerien tekemiseen (Paananen, Petteri 2011, 7). Toisin kuin HTML5 Canvas, Flash tarvitsee erillisen liitännäisen toistamaan Flash-animaatioita.

```
<!DOCTYPE html>
<html>
<head>
    <title>EaselJS demo: Getting started</title>
    <link href="../shared/demo.css" rel="stylesheet" type="text/css">
    <script src="../../lib/easeljs-0.5.0.min.js"></script>
    <script>
        function init() {
            var stage = new createjs.Stage("demoCanvas");
            var circle = new createjs.Shape();
            circle.graphics.beginFill("red").drawCircle(0, 0, 50);
            circle.x = 100;
            circle.y = 100;
            stage.addChild(circle);
            // stage.addChild(new createjs.Shape()).setTransform(100,100).graphics.f("red").dc(0,0,50);
            stage.update();
        }
    </script>
</head>
<body onLoad="init();">
    <canvas id="demoCanvas" width="500" height="200">
        alternate content
    </canvas>
</body>
</html>
```

Kuvio 26. Täytetty ympyrä, joka on toteutettu EaselJS-kirjastolla (Overscan 2013).

Kuviossa 26 on esimerkki EaselJS-kirjaston käytöstä. Script-tägin sisälle lisätään EaselJS-kirjasto. Funktion sisään on määritetty stage-olio, joka viittaa canvas-tägin id-arvoon. Tämän jälkeen luodaan kuvio, jonka alapuolella määritetään kuvion muoto, väri, ympyrän säde, keskipiste sekä x- ja y pisteet canvasilla. Ympyrän piirtämiseen käytetään drawCircle()-metodia. Stage.addChild(circle) lisää kuvion stage-säiliöön ja stage.update()-metodi tekee kuviosta näkyvän. Rivien 9-14 koodit voidaan kirjoittaa myös lyhyempään muotoon, kuten rivillä 15 on tehty. (Overscan 2013.)

3.4.3 Muita JavaScript-kirjastoja canvasille

HTML5 Canvasille on olemassa paljon erilaisia JavaScript-kirjastoja eri tarkoituksiin. Osa kirjastoista sopii monipuolisesti piirtämiseen ja animaatioiden tekoon ja osa taas on erikoistunut johonkin alueeseen, kuten kaavioiden luomiseen, vektorigrafiikkaan tai pelikehitykseen. Useimmissa kirjastoissa on sisäänrakennettuja metodeja ja ominaisuuksia erilaisten kuvioiden piirtämiseen, animaatioihin, efekteihin, interaktiivisten elementtien tekemiseen ja erilaisille tapahtumille. Taulukossa 10 esitetään esimerkkejä canvasille tarkoitetuista JavaScript-kirjastoista.

Taulukko 10. JavaScript-kirjastoja HTML5 Canvasille.

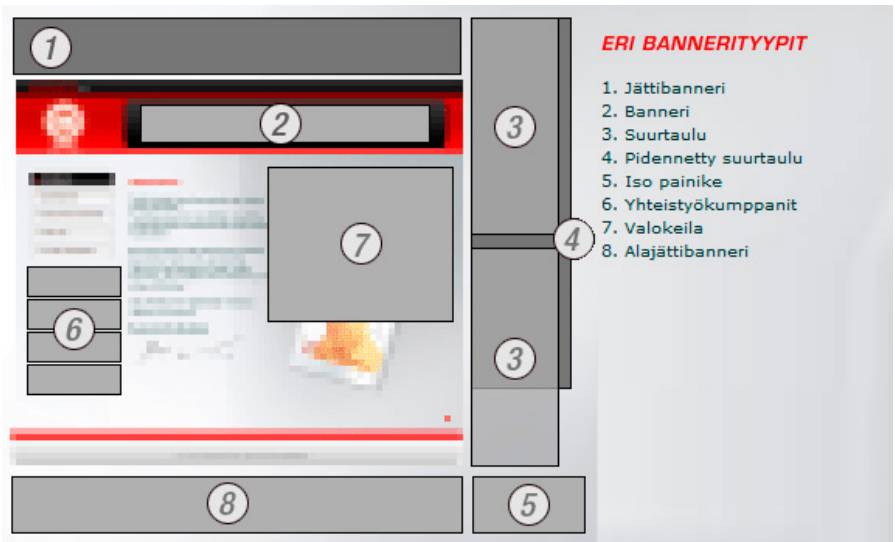
JavaScript-kirjastoja		
Kirjasto	**Kieli**	**Kuvaus**
Paper.js	PaperScript / JavaScript	Vektorigrafiikka-kirjasto, joka hyödyntää muun muassa kerroksia (layer), ryhmiä (group) ja polkuja (path) kuvioiden piirtämisessä (Paper.js 2013).
Fabric.js	JavaScript	Tarjoaa interaktiivisen oliomallin, joka mahdollistaa kuvioiden muokkauksen suoraan hiiren avulla. Canvasilla tehdyn piirroksen voi muuttaa myös SVG muotoon. (Fabric.js 2013.)
Processing.js	JavaScript	Tarjoaa pohjan muun muassa digitaalisen taiteen, pelien, interaktiivisten animaatioiden ja infografiikan tekemiseen (Processing.js 2013).
Canvas_library	CoffeeScript	Flashin kaltainen ohjelmointirajapinta piirtämiseen ja animaatioiden tekemiseen (Lawson 2011).
bHive	JavaScript	Kevyt oliopohjainen kirjasto piirtämiseen, animaatioihin ja interaktiivisten elementtien luomiseen (Becker 2011).
RGraph	JavaScript	Kaavioiden luomiseen tarkoitettu JavaScript-kirjasto (Heyes 2008).
Canvas Query	jQueryn kaltainen	Pelikehitykseen tarkoitettu JavaScript-kirjasto (Canvas Query 2013).
jCanvas	jQuery	Tarjoaa perus piirtämisen lisäksi lisäominaisuuksia, kuten kerrosten käyttöä ja animaation luomista, canvasille jQueryn syntaksilla (Evans 2013).
jCanvaScript	JavaScript	Oliopohjainen JavaScript-kirjasto, joka tuo erilaisia lisäominaisuuksia canvasiin (jCanvaScript 2013).

4 Bannerin toteutus canvasilla ja KineticJS-kirjastolla

Tässä luvussa käsitellään käytännöntyönä toteutettua banneria ja sen työvaiheita. Aluksi selvitetään lyhyesti, mitä bannerit ovat. Tämän jälkeen käsitellään bannerin suunnitteluvaiheita, joihin sisältyy muun muassa bannerityypin ja JavaScript-kirjaston valinta sekä animaation suunnittelu. Suurimmaksi osaksi luvussa käydään läpi bannerin toteutusprosessia. Toteutuksen pääkohtina ovat muun muassa elementtien ja liikeanimaatioiden luominen. Lopuksi mietitään canvasin soveltumista banneriin.

4.1 Lyhyt katsaus bannereihin

Bannerit ovat verkkosivuilla olevia useimmiten suorakulmion mallisia mainoksia. Bannerimainontaa kutsutaan myös display-mainonnaksi. Formaatiltaan banneri on useimmiten staattinen kuva, kuten jpg tai png, tai Flash-animaatio. Bannerin koko ja tyyppi vaihtelee paljon. Erilaisia bannerityyppejä ovat muun muassa banneri, jättibanneri, suurtaulu, pidennetty suurtaulu, alajättibanneri ja valokeila. (Banneri.info 2013a; Banneri.info 2013b; Jansson 2011; KWD Digital 2013.) Tyyppien nimet vaihtelevat osittain lähteittäin. Kuvio 27 esittää keskeisiä bannerityyppejä ja niiden sijainteja verkkosivulla.



Kuvio 27. Erilaisia bannerityyppejä (Banneri.info 2013).

Tehokas banneri herättää kävijän huomion ja kiinnostuksen sekä tuo mainonnan kohteen nopeasti esille ja on riittävän lyhyt. Bannerin tarkoituksena on ohjata verkkosivuilla vieraileva kävijä mainostajan sivulle bannerissa olevan linkin kautta. Hyvin toimivasta

bannerista välittyy oikeanlainen viesti ja se on myös visuaalisesti näyttävä, jotta banneri erottuu muiden joukosta. Bannerit voivat myös olla sivuston ylläpitäjän omia mainoksia esimerkiksi omalle tuotesivulle. (Banneri.info 2013a; Banneri.info 2013b; Jansson 2011; KWD Digital 2013.)

Tarkastelin verkosta löytyviä perinteisillä menetelmillä, kuten Flashilla, tehtyjä bannereita. Bannerit vaihtelivat yksinkertaisista monimutkaisiin. Suurin osa bannerien animaatioista ovat vaihtuvia kuvia ja tekstejä. Monimutkaisemmissa bannereissa oli animoituja interaktiivisia elementtejä tai näyttäviä animaatioita. Bannerien animaatiot pyörivät vähintään kerran.

4.2 Bannerin suunnittelu

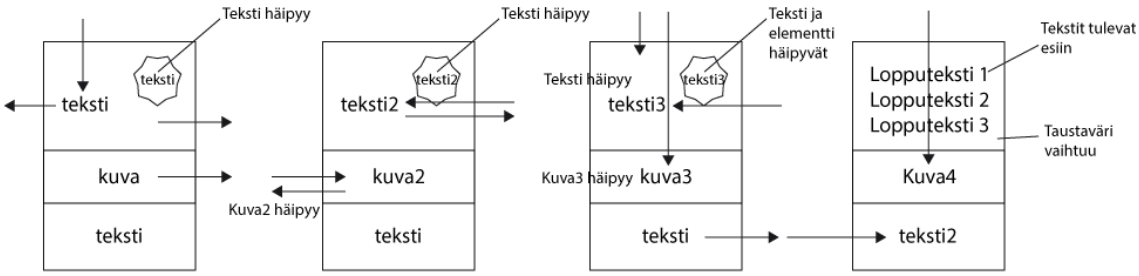
Keskityn opinnäytetyössäni bannerin tekniseen toteutukseen canvasilla. Työssä ei keskitytä bannerin visuaaliseen puoleen, eikä bannerin viestiin. Tärkeintä on canvasin soveltuvuus banneriin, bannerin toimivuus sekä kuvioiden ja animaatioiden tekninen toteutus. Lähdin myös suunnittelemaan banneria teknisestä ja käytännönläheisestä näkökulmasta, missä otan huomioon ainakin osittain bannerien yleisiä teknisiä vaatimuksia.

Valitsin työhön bannerityypeistä suurtaulun. Suurtaulun standardikoko on useamman lähteen mukaan 140 x 350 pikseliä. Animaation pituudeksi suositellaan maksimissaan 15 sekuntia, mikä otetaan huomioon tässä työssä. Tiedostokoot vaihtelevat formaatin mukaan. Esimerkiksi videomainosten suositeltu koko on maksimissaan noin kaksi megatavua, Flash- tai gif-animaation koko 30 kilotavua ja staattisen kuvan koko 20 kilotavua. (Banneri.info 2013b; IAB Finland 2006; IAB Finland 2009.) Tiedostokoot saattavat vaihdella paikasta riippuen.

Bannerin animaatioiden vuoksi on järkevää ottaa käyttöön JavaScript-kirjasto lisäämään kontekstin ominaisuuksia. Valitsin KineticJS-kirjaston, koska se on yksi tämän hetken suosituimmista kirjastoista, hyvin dokumentoitu ja sen syntaksi vaikuttaa helppokäyttöiseltä 2d-kontekstin ohjelmointirajapinnan oppimisen jälkeen. Minulla ei ole paljon kokemusta Flashista, joten EaselJS-kirjaston käyttäminen vaikutti hankalalta.

Työn ideana on toteuttaa liikeanimaatioita (tween animation) sisältävä neljän näkymän banneri. Bannerin tekstit, kuvat ja taustat liikkuvat ajastettuina ja ne sisältävät erilaisia

kulkunopeuksia (easing). Kuviossa 28 on suunnitelma animaation kulusta. Kuvion nuolet kertovat elementtien siirtymäsuunnat. Kulkunopeudet ja ajastukset syntyvät vasta toteutusvaiheessa kokeilemalla. Ideana on myös, että bannerin animaatio pyörii läpi vain kerran ja jää sen jälkeen loppunäkymään.



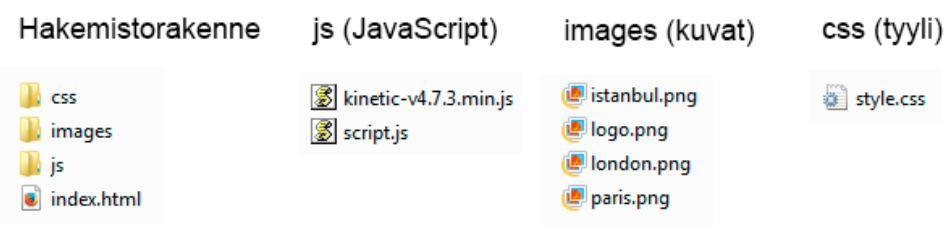
Kuvio 28. Animaatiosuunnitelma bannerille.

4.3 Bannerin toteutus

Lähdin toteuttamaan banneria animaatiosuunnitelman pohjalta. Tutustuin KineticJS-kirjaston dokumentaatioon (http://kineticjs.com/docs/index.html), josta otin mallia koodin syntaksiin eli kirjoitusmuotoon sekä tutkin kirjaston sisäänrakennettuja ominaisuuksia. Bannerin JavaScriptin lähdekoodi löytyy liitteestä 2. Tämän luvun alaluvuissa käsitellään bannerin teon keskeisimpiä asioita, kuten prosessin aloittamista, elementtien ja liikeanimaatioiden luomista sekä bannerin lopputulosta.

4.3.1 Hakemistorakenne ja tiedostojen esivalmistelut

Aloitin bannerin tekemisen luomalla tarvittavat tiedostot ja kansiot. Kuviossa 29 esitetään bannerin hakemistorakenne ja kansioiden sisältämät tiedostot. Edellisistä esimerkeistä poiketen canvasille piirtämiseen tarkoitetulle JavaScriptille on luotu oma tiedosto (script.js). Tämä helpottaa koodin hallintaa, kun koodia on paljon sekä pitää koodin järjestyksessä. Samaan js-kansioon on lisätty KineticJS-kirjaston tiedosto. Kuvat ja css-tyylitiedosto on laitettu omiin kansioihin selkeyden vuoksi. Css-tiedostoa ei välttämättä tarvita bannerin tekemiseen, koska bannerin tyylit määritellään JavaScriptin kautta. Tyylitiedostossa canvas-alue on asemoitu keskelle html-sivua. Html-tiedosto sijaitsee juurihakemistossa.



Kuvio 29. Bannerin tekemiseen käytetyt tiedostot ja hakemistorakenne.

Luvussa 3.4.1 mainittiin KineticJS-kirjaston käyttämisen eroavan hieman canvasin normaalista käyttöönotosta (vertaa luku 3.1). Html-tiedostoon ei merkitä canvas-tägiä ja 2d-kontekstia, vaan ne löytyvät KineticJS:n omasta kirjastosta. Canvas-tägin sijaan html-tiedostoon määritetään div-tägi, jolle annetaan jokin id-arvo. Kuviossa 30 on banneriin tarvittava html-koodi. Html-tiedostoon on annettu id-arvoksi container. JavaScript-tiedostot voidaan lisätä head-osioon tai ennen body-tägin lopetusta.

```
1  <!DOCTYPE html>
2  <html lang="fi">
3    <head>
4      <meta charset="utf-8">
5      <title>HTML5 Canvas banneri</title>
6      <link rel="stylesheet" href="css/style.css">
7      <script src="js/kinetic-v4.7.3.min.js"></script>
8      <script src="js/script.js"></script>
9    </head>
10   <body>
11     <div id="wrapper">
12         <a href="https://www.google.fi/" target="_blank">
13             <div id="container"></div>
14         </a>
15     </div>
16   </body>
17 </html>
```

Kuvio 30. Bannerin tekemiseen tarkoitettu html-tiedosto.

Kuvion 30 koodissa on elementtejä, jotka eivät ole välttämättömiä bannerin kannalta. Div-elementtiin nimeltään wrapper on lisätty tyyli, joka keskittää canvasin keskelle html-sivua. Monissa verkossa olevissa bannereissa koko bannerialue on linkki mainostajan sivulle. Kuviossa 30 koko canvas-alue (div id=”container”) on määritetty linkiksi a-tägien sisälle. Mikäli canvas-alueeseen halutaan interaktiivisia elementtejä, kuten hiirellä siirrettäviä kuvia, koko canvas-aluetta ei voi käyttää linkkinä. Tällöin voidaan luoda esimerkiksi erillinen linkki html-tiedostossa ja asemoida se canvasin päälle css-tyylillä.

4.3.2 Elementtien luominen

Kuten luvussa 3.4.1 mainittiin, KineticJS-kirjasto sisältää erilaisia sisäänrakennettuja ominaisuuksia ja metodeja, joita on enemmän kuin canvasin omassa 2d-kontekstissa. Ominaisuudet muistuttavat nimeltään melko paljon 2d-kontekstin ominaisuuksia, joita esitettiin luvussa 3.3, mutta ovat useimmiten lyhyempiä. Esimerkiksi 2d-kontekstin fillStyle ja strokeStyle ominaisuudet kirjoitetaan KineticJS-kirjastoa käytettäessä nimillä fill ja stroke. Osa ominaisuuksien nimistä on samoja, esimerkiksi lineCap ja lineJoin. Kaikki metodit ja ominaisuudet löytyvät KineticJS-kirjaston dokumentaatiosta.

Luvussa 3.4.1 kerrottiin KineticJS-kirjastolla olevan oliopohjainen ohjelmointirajapinta. Kuviossa 31 esitetään KineticJS:n käyttämä kirjoitusmuoto olioiden luomiseen. Ensin määritetään muuttuja, esimerkissä var suorakulmio. Seuraavaksi valitaan haluttu metodi, esimerkiksi kuviossa 31 new Kinetic.Rect tarkoittaa uutta suorakulmiota. Sulkujen sisään lisätään haluttu määrä ominaisuuksia ja niiden arvoja. Ominaisuuden jälkeen kirjoitetaan kaksoispiste, arvo ja pilkku. Viimeisen ominaisuuden arvon jälkeen ei tule pilkkua ja olio suljetaan puolipisteellä.

```
var suorakulmio = new Kinetic.Rect ({
      ominaisuus1: arvo,
      ominaisuus2: arvo,
      ominaisuus3: arvo
});
```

Kuvio 31. KineticJS-kirjaston käyttämä kirjoitusmuoto olioiden luomiseen.

Banneri piirretään canvasille script.js-tiedoston kautta. Canvas otetaan käyttöön luomalla stage-olio (kuvio 32). Olioon määritetään canvas-alueen leveys, korkeus ja html-tiedostossa määritetty id-arvo (kuviossa 32 ’container’). Kuvion olio on muodostettu samalla syntaksilla kuin kuvion 31 esimerkki. New Kinetic.Rect sijaan kirjoitetaan new Kinetic.Stage.

```
7   // Luodaan stage-olio ja määritetään canvasin koko
8   var stage = new Kinetic.Stage({
9       container: 'container',
10      width: 140,
11      height: 350
12  });
```

Kuvio 32. Canvas-alueen käyttöönotto.

Bannerin muutkin elementit luodaan samaa syntaksia hyödyntäen, mutta ne kannattaa lisätä omille kerroksille (layer). Omilla kerroksilla oleviin elementteihin on helppo lisätä esimerkiksi animaatioita tai vaihtaa elementtien järjestystä. Kuvion 33 koodi havainnoi suorakulmion piirtämistä uudelle kerrokselle new Kinetic.Layer -käskyä käyttäen. Kerroksen määrittämisen jälkeen luodaan uusi suorakulmio new Kinetic.Rect -käskyllä. Kuvion 33 esimerkin ominaisuuksiin on lisätty suorakulmion leveys, korkeus ja täyttöväri sekä x- ja y-pisteet koordinaatistolla. Koordinaatisto toimii samalla logiikalla kuin esitetään luvuissa 3.2 ja 3.3.

```
15  var tausta4 = new Kinetic.Layer();
16  var rect5 = new Kinetic.Rect({
17      x: 0,
18      y: 0,
19      width: 140,
20      height: 350,
21      fill: '#ada99f'
22  });
```

Kuvio 33. Uuden kerroksen ja suorakulmion luominen.

KineticJS-kirjasto sisältää sisäänrakennettuja metodeja erilaisten kuvioiden, kuten tähtien, ympyröiden ja monikulmioiden, piirtämiseen (KineticJS 2013b). Bannerissa on käytetty tähtikuviota, joka on luotu Kinetic.Star-käskyllä. Kuvion 34 tähdelle on annettu ominaisuuksiksi sakarien pisteiden määrä, sisä- ja ulkoreunojen säteet, täyteväri, äärriviivan väri ja -leveys sekä x- ja y-piste koordinaatistolla. Ilman erillistä JavaScript-kirjastoa tähden voisi piirtää esimerkiksi viivoja yhdistelemällä.

```
54  // Luodaan tähti-kuvio
55  var tahti = new Kinetic.Layer();
56  var animatedStar = new Kinetic.Star({
57      x: 105,
58      y: 40,
59      numPoints: 9,
60      innerRadius: 20,
61      outerRadius: 25,
62      fill: '#4d4c48',
63      stroke: 'black',
64      strokeWidth: 1
65  });
```

Kuvio 34. KineticJS-kirjaston sisäänrakennettu tähtikuvio.

Tekstin luominen noudattaa samaa logiikkaa kuin kuvioiden luominen. Kuvion 35 esimerkin ominaisuudet sisältävät x- ja y-pisteen, tekstin, kirjasimen koon, rivinkorkeuden,

kirjasintyypin, tekstin kohdistamisen, täytevärin, varjon etäisyydet ja -läpinäkyvyyden. ShadowOffsetX- ja shadowOffsetY ominaisuudet ovat samoja kuin canvasin omassa 2d-kontekstissa (vertaa luku 3.3.2).

```
91  // Luodaan Varaa matkasi tästä -teksti
92  var tekstiVaraa = new Kinetic.Layer();
93  var textVaraa = new Kinetic.Text({
94          x: -130,
95          y: 255,
96          text: 'Varaa\n matkasi\n tästä',
97          fontSize: 20,
98          lineHeight: 1.3,
99          fontFamily: 'arial',
100         align: 'center',
101         fill: '#eee9e3',
102         shadowOffsetX: 0.2,
103         shadowOffsety: 0.2,
104         shadowOpacity: 0.4
105 });
```

Kuvio 35. Tekstin luominen.

Kun elementit on luotu, ne lisätään kerrokseen ja stage-olioon halutussa järjestyksessä. Alimmaiset kerrokset koodissa sijoittuvat canvasille päällimmäisiksi eli toisten kuvioiden eteen. Kuvio 36 ohjeistaa elementtien lisäämisen kerrokseen ja kerroksen lisäämisen stage-olioon. Elementti lisätään kerrokseen yhdistämällä kerroksen nimi add-käskyyn, joka saa parametriksi olion eli elementin nimen. Kerros lisätään stage-olioon samalla periaatteella eli sulkuihin tulee kerroksen nimi. Kuvion 36 oikealla puolella lisätään kuvion 35 elementti ensin kerrokseen ja sitten stage-olioon. Elementit ja kerrokset voidaan lisätä peräkkäin koodiin, kuten bannerin kohdalla on tehty (liite 2, sivu 7).

```
// Lisää elementin kerrokseen
kerroksenNimi.add(olionNimi);
```

```
390  tekstiVaraa.add(textVaraa);
```

```
// Lisää kerroksen stage-olioon
stage.add(kerroksenNimi);
```

```
408  stage.add(tekstiVaraa);
```

Kuvio 36. Elementin lisääminen kerrokseen ja kerroksen lisääminen stage-olioon.

Elementtien luominen oli melko yksinkertaista, kun oli opetellut canvasin oman 2d-kontekstin. Muun muassa osa ominaisuuksien nimistä on helposti pääteltävissä. Koodin kirjoitusmuoto KineticJS-kirjastoa käytettäessä on mielestäni yksinkertaisempi kuin canvasin omassa kontekstissa, koska ei tarvitse muistaa metodin sulkujen sisälle tule-

via parametreja erikseen (vertaa lukuun 3.3). Toisaalta koodi myös vie enemmän rivitilaa.

Kerrosten käyttämisen oppi nopeasti ja se helpotti canvasin kanssa työskentelyä. Kun elementit ovat omilla kerroksillaan, niiden ominaisuuksia ja paikkaa canvasilla on helppo muuttaa. Elementteihin on myös helpompi lisätä muun muassa animaatioita, mikäli halutaan animoida yksittäiset elementit eritavoin. Luvussa 4.3.3 käsitellään liikeanimaatioiden lisäämistä elementteihin.

Elementtien muodostukseen liittyvä keskeisin ongelma oli unohtaa lisätä elementit kerrokseen ja kerros stage-olioon. Tällöin elementit eivät piirry canvasille. Myös pienet kirjoitus- ja syntaksivirheet häiritsivät työskentelyä. Elementin syntaksista kannattaa tarkastaa pilkkujen paikat, koska ne saattavat estää elementin näkymisen canvasilla.

4.3.3 Liikeanimaation luominen

Liikeanimaatio eli tween mahdollistaa animaation nykyisen ja uuden tilan välillä. Edellisen luvun 4.3.2 elementit kuvaavat nykyistä tilaa ja tässä luvussa käsiteltävät animaatiot uutta tilaa. Luvussa esitetään esimerkkejä bannerissa käytettävistä liikeanimaatioista. Bannerin liikeanimaatiot löytyvät liitteestä 2 sivuilta 7–11.

```
421  // Luodaan tween päällimmäiselle taustalle - poistuu vasemmalle
422  var tween1 = new Kinetic.Tween({
423      node: tausta3,
424      duration: 1,
425      x: 140,
426      y: 0,
427      easing: Kinetic.Easings.EaseInOut
428  });
429
430  // Tween alkaa 4 sekuntin kuluttua
431  setTimeout(function() {
432      tween1.play();
433  }, 4000);
```

Kuvio 37. Esimerkki yksinkertaisesta liikeanimaatiosta.

Uusi liikeanimaatio luodaan samalla logiikalla kuin elementti-olio edellisessä luvussa. Kuviossa 37 on esimerkki yksinkertaisesta liikeanimaatiosta, jossa suorakulmionmallinen tausta poistuu canvasilta vasemmalle. New Kinetic.Tween luo liikeanimaation ja saa sulkujen sisään ominaisuuksia. Node-ominaisuus saa arvokseen elementin kerrok-

sen (layer) nimen, joka halutaan animoida ja duration-ominaisuus määrittää animaation keston. X- ja y pisteet määrittävät paikan, johon elementti liikkuu. Kuvion 37 esimerkissä suorakulmion vasen reuna liikkuu oikeaan reunaan eli pois canvasilta, koska canvasin leveys on 140 pikseliä. X-arvoksi siis voidaan antaa jokin luku, joka on suurempi kuin canvasin leveys.

KineticJS-kirjasto tarjoaa useita erilaisia metodeja animaation kulkunopeudelle (easing) (KineticJS 2013c). Banneriin on käytetty erilaisia kulkunopeuksia tuomaan vaihtelevuutta animaatioihin. Kuvion 37 esimerkissä kulkunopeus on määritetty Kinetic.Easings-käskyllä, johon on lisätty kulkunopeus EaseInOut. EaseInOut hidastaa vauhtia animaation alussa ja lopussa. Muita bannerissa käytettyjä kulkunopeuksia ovat lineaarinen (Linear), loppupuolella kimpoileva (BounceEaseOut), alkupuolella takaisinpäin liikkuva (BackEaseIn) ja vahvasti lopettava (StrongEaseOut) kulkunopeus.

Liikeanimaatioita voi ajastaa käyttämällä JavaScriptin setTimeout()-metodia. Metodiin määritetään aika millisekunteina. Kuviossa 37 on määritetty liikeanimaatio alkamaan neljän sekunnin kuluttua. Liikeanimaation käynnistämiseen käytetään KineticJS-kirjaston play()-metodia, johon liitetään liikeanimaatio-olion muuttujan nimi (kuvio 37).

```
625  // Luodaan tween Hinnalle - Hinta näkyviin ja häivytys
626  var tweenHinta3 = new Kinetic.Tween({
627      node: textHinta3,
628      duration: 1,
629      opacity: 1,
630      easing: Kinetic.Easings.Linear,
631      onFinish: function () {
632          var tween13 = new Kinetic.Tween({
633              node: textHinta3,
634              duration: 1,
635              opacity: 0,
636              easing: Kinetic.Easings.Linear
637          });
638          setTimeout(function() {
639              tween13.play();
640          }, 2600);
641      }
642  });
643
644  // Tween alkaa 9 sekuntin kuluttua
645  setTimeout(function() {
646      tweenHinta3.play();
647  }, 9000);
```

Kuvio 38. Esimerkki moniosaisesta liikeanimaatiosta.

Liikeanimaation sisälle voi lisätä useita liikeanimaatioita onFinish-ominaisuudella. OnFinish-ominaisuus saa arvokseen funktion, jonka sisään luodaan uusi liikeanimaatio

aiempien kappaleiden tavoin. Kuvion 38 olio on aiemmin määritetty näkymättömäksi (opacity: 0). Kuviossa oliolle luodaan liikeanimaatio, joka alkaa yhdeksän sekunnin kuluttua bannerin käynnistymisestä ja muuttaa elementin näkyväksi (opacity: 1). Liikeanimaation sisällä oleva liikeanimaatio käynnistyy 2,6 sekunnin päästä sitä ympäröivän animaation alkamisesta ja häivyttää elementin (opacity: 0) pois näkyvistä.

Liikeanimaatioiden luominen onnistui melko helposti syntaksin opittua ja kokeilemalla erilaisia animaatioita. Liikeanimaatiot, joissa elementit ilmestyvät canvasin ulkopuolelta tai siirtyvät sen ulkopuolelle, on helppo toteuttaa asemoimalla elementti-olioiden x- ja y-akselin arvot canvasin ulkopuolelle. Kun luodaan liikeanimaatio, määritetään x- ja y-akselille uusi sijainti, johon elementti liikkuu elementin alkuperäisestä pisteestä. Käytin samaa logiikkaa myös muun muassa elementtien häivyttämiseen.

Liikeanimaation sisään laitettavan toisen liikeanimaation luomiseen tarvitsin apua internetistä olevista esimerkeistä. Esimerkkien lukemisen ja kokeilemisen kautta animaatioiden luominen hahmottui kuitenkin melko nopeasti. Suurimpana ongelmana animaation alkuperäiseen epäonnistumiseen oli syntaksivirhe eli pilkku puuttui yhdestä kohdasta.

4.3.4 Kuvan luominen ja liikeanimaation lisääminen

Luvussa 3.3.5 käsiteltiin kuvan piirtämistä canvasin omalla 2d-kontekstilla. Luvun tavoin KineticJS-kirjastoa käytettäessä muodostetaan kuvaolio ja käytetään onload-funktiota varmistamaan kuvan latautuminen ennen kuvan käyttöä. Myös kuvan lähde merkitään samalla tavalla kuvaolion jälkeen. Kuviossa 39 on esimerkki kuvan luomisesta. Onload-funktion sisälle luodaan kuva käskyllä new Kinetic.Image. Sulkujen sisään asetetaan ominaisuuksia, kuten luvussa 4.3.2. Kuvion 39 esimerkissä kuvalle on määritetty ominaisuuksiksi kuvan x- ja y akselit koordinaatistolla, kuvaolio sekä kuvan leveys ja korkeus. Kuvion 39 esimerkissä kuvaa varten on myös luotu oma kerros.

Liikeanimaation lisääminen kuvaan, elementin lisääminen kerrokseen ja kerroksen lisääminen stage-olioon oli aluksi ongelmallista. Nämä oli tarkoitus luoda alun perin erikseen, jotta kerroksia olisi voinut hallita paremmin samasta paikasta ja koodi olisi ollut paremmassa järjestyksessä. Liikeanimaatio sekä lisäykset kerrokseen ja stage-olioon toimivat kuitenkin normaalisti, kun ne asetetaan kuvaolion sisään, kuten kuvan 39 esimerkissä on tehty. Lisäys kerrokseen ja stage-olioon luodaan kuvaolion sisään nor-

maalisti, kuten luvussa 4.3.2 on mainittu sekä liikeanimaatio luodaan luvun 4.3.3 mukaisesti.

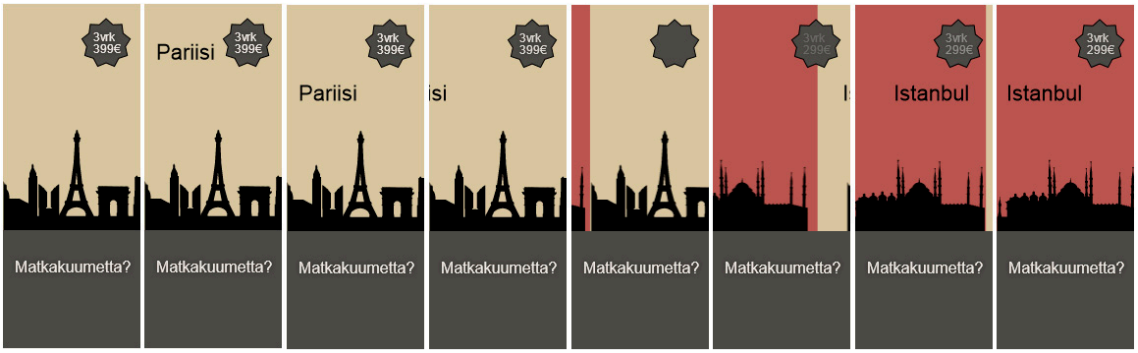
```
206  // Luodaan kuva 1 - Pariisi
207  var image1 = new Kinetic.Layer();
208      var imageObj1 = new Image();
209          imageObj1.onload = function() {
210              var kuva1 = new Kinetic.Image({
211              x: 0,
212              y: 122,
213              image: imageObj1,
214              width: 140,
215              height: 108
216          });
217
218          // Lisää kuvion kerrokseen (layer)
219          image1.add(kuva1);
220
221          // Lisää kerroksen stage-olioon
222          stage.add(image1);
223
224          // Tween-animaatio - kuva siirtyy oikealle
225          var tween3 = new Kinetic.Tween({
226              node: kuva1,
227              duration: 1,
228              x: 200,
229              easing: Kinetic.Easings.EaseInOut,
230              opacity: 1
231          });
232
233          // Tween alkaa 4 sekuntin kuluttua
234          setTimeout(function() {
235              tween3.play();
236          }, 4000);
237
238      };
239  // Kuvan lähde
240  imageObj1.src = 'images/paris.png';
```

Kuvio 39. Esimerkki kuvan luomisesta, missä on liikeanimaatio.

4.3.5 Bannerin lopputulos

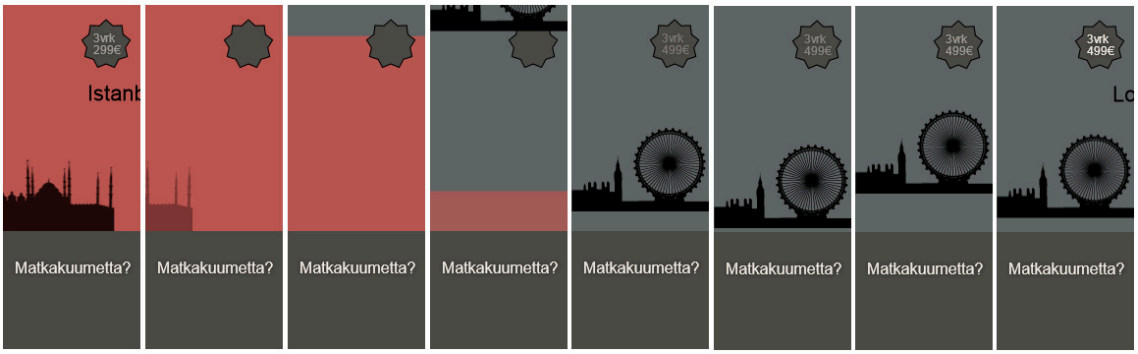
Käytännöntyön lopputuloksena on banneri, joka sisältää erilaisia ajastettuja liikeanimaatioita, kuvia, kuvioita (suorakulmioita ja tähden) ja tekstiä. Banneri on koolttaan 140 x 350 pikseliä, kestoltaan alle 15 sekuntia ja banneri pyörii läpi vain kerran. Bannerin tekemiseen käytetty JavaScript koodi (script.js) löytyy liitteestä 2 ja html-koodi luvusta 4.3.1. Kuvioissa 40, 41 ja 42 on havainnoitu bannerin animaation kulkua. Bannerin kaupunkikuvat on otettu Free Vector Graphic Download -verkkosivulta (http://7428.net/2013/07/city-landmarks-silhouette-2.html) ja logon kuva SeekLogo -sivustolta (http://seeklogo.com/below-cost-travel-agency-logo-18215.html). Kuvia on muokattu omiin tarpeisiin sopivaksi.

Bannerin alkunäkymä sisältää kuvan, alhaalla olevan suorakulmion, Matkakuumetta-tekstin, tähtikuvion, hinta-tekstin ja taustan (kuvio 40). Banneri etenee animoidulla Pariisi-tekstillä, joka ilmestyy ylhäältä, pysähtyy hetkeksi ja jatkaa vasemmalle. Tämän jälkeen tähtikuviosta vaihtuu hinta läpinäkyvyyttä säädellen. Samanaikaisesti seuraava tausta (suorakulmio) ja kuva ilmestyy banneriin vasemmalta. Istanbul-teksti ilmestyy canvasille oikealta ja jää hetkeksi paikalleen.



Kuvio 40. Bannerin animaation kulku.

Kuvion 41 näkymässä Istanbul-teksti siirtyy canvasilta oikealle ja Istanbul-kuva vasemmalle samalla muuttaen läpinäkyvyyttä. Hinta-teksti vaihtuu toiseen edellisen kappaleen tavoin. Bannerin yläpuolelta ilmestyy uusi tausta alaspäin, minkä jälkeen Lontoo-kuva tipahtaa alla olevan suorakulmion reunaan kimpoillen muutaman kerran ylös ja alas. Kuvion 41 loppupuolella kolmas hinta-teksti tulee näkyviin tähden kohdalle ja Lontoo-teksti ilmestyy vasemmalta.



Kuvio 41. Bannerin animaation kulku.

Kuviossa 42 Lontoo-näkymä jää hetkeksi paikalleen, minkä jälkeen häipyy pois. Matkakuumetta-teksti siirtyy oikealle ja Varaa matkasi tästä -teksti ilmestyy banneriin vasemmalta. Tausta vaihtuu toiseksi ja bannerin yläpuolella olevat tekstit ilmestyvät näkyviin. Bannerin loppupuolella logo-kuva ilmestyy ylhäältä keskelle banneria ja animaatio pysähtyy näkymään. Koko canvas-alue toimii linkkinä mahdollisen palveluntarjoajan sivulle.

Kuvio 42. Bannerin animaation kulku.

4.4 Canvasin soveltuminen banneriin

Tässä opinnäytetyössä tehty banneri toimii melko sulavasti useammilla selaimilla, mutta hyvin pientä pätkimistä esiintyy. Testasin banneria uusimmilla ja suosituilla selaimilla, kuten Firefoxilla, Chromella, Internet Explorerilla, Operalla ja Safarilla. Canvas ei toimi ilman erityistoimenpiteitä Internet Explorerin versioilla kahdeksan ja seitsemän, kuten luvussa 2.4 mainittiin. Käytin testaamiseen PC-tietokonetta.

Canvasin hyvänä puolena on se, että se toimii erilaisissa laitteissa, kuten pöytätietokoneissa, älypuhelimissa ja tableteissa, ja se ei tarvitse erillisiä liitännäisiä toimiakseen. Testasin bannerin toimivuutta PC-tietokoneen lisäksi iPad3-tabletilla, Sonyn Xperia Z- ja Samsungin Galaxy ACE 1 älypuhelimilla. Xperia Z-älypuhelin ja iPad-tabletti pyörittävät banneria hyvin. Animaatiot liikkuvat sujuvasti ja pätkimistä esiintyy hyvin vähän. Galaxy ACE 1 -älypuhelin pyörittää banneria tyydyttävästi. Animaatio pyörii, mutta pätkii enemmän kuin uudemmilla laitteilla. Luvussa 2.4 todettiinkin canvasin toimivan paremmin uudemmilla laitteilla.

Bannerin kuvanlaatu vaihtelee selaimesta ja laitteesta riippuen. Canvas on bittikarttagrafiikkaa, joten banneri näyttää osassa selaimista hieman epätarkalta. Parhaiten tämä

näkyy teksteistä. Xperia Z-älypuhelimessa ja iPad-tabletissa on enemmän pikseleitä kokoonsa nähden kuin 1920 x 1080 tarkkuuden omaavassa tietokonenäytössä, joten myös banneri näyttää paljon tarkemmalta.

Luvussa 4.2 mainittiin bannerin teknisiä vaatimuksia, joihin kuuluu muun muassa tiedostokoon rajoitus. Koko vaihtelee staattisilla kuvilla ja Flash-animaatioilla noin 20–30 kilotavun välillä, mutta paikasta riippuen se voi olla enemmänkin. Videomainoksen kokosuositus on kaksi megatavua. Bannerin tiedostojen, jotka käsiteltiin luvussa 4.3.1, kokonaiskoko on 122 kilotavua eli se ylittää reippaasti 30 kilotavun rajan. Suurin osa bannerin koosta syntyy KineticJS-kirjastosta, joka on kooltaan noin 99 kilotavua. Kirjastoa saa hieman pienemmäksi karsimalla tarpeettomia ominaisuuksia.

5 Yhteenveto

Opinnäytetyön lopputuloksesta tuli tiivis selvitys HTML5 Canvasista ja sen käytöstä teknisestä ja käytännönläheisestä näkökulmasta tarkasteltuna. Opinnäytetyössä myös esitettiin, miten banneri tehdään ja miten canvas soveltui opinnäytetyössä tehtyyn banneriin. Opinnäytetyöstä on hyötyä opiskelijoille ja muille canvasista kiinnostuneille, jotka haluavat oppia canvasin perusteet. Canvasilla toteutettu banneri tarjoaa vaihtoehtoisen tavan toteuttaa bannerin esimerkiksi alan ammattilaisille, jotka miettivät canvasin soveltuvuutta bannerin tekoon.

Opinnäytetyön teoreettinen osuus alkoi katsauksella HTML5 Canvasiin. Toisen luvun keskeisempinä asioina oli canvas-elementin ja HTML5 Canvasiin liittyvän 2d-kontekstin ero. Canvas-elementti toimii säiliönä grafiikalle ja 2d-kontekstin kautta luodaan varsinaista sisältöä canvasille yleensä JavaScriptillä. Luvussa käsiteltiin myös Canvasin käyttömahdollisuuksia. Canvas soveltuu monenlaisiin selaimessa toistettaviin töihin, kuten grafiikkaan, peleihin, animaatioihin sekä kuva- ja videomanipulaatioihin. Suurin osa canvasilla tehdyistä animaatioista on interaktiivisia eli ne reagoivat esimerkiksi käyttäjän kosketukseen tai hiirenklikkaukseen. Canvas soveltuu myös pienten applikaatioiden, kuten piirto-ohjelman ja kuvagallerian, tekemiseen.

Työssä kerrottiin canvasin toimivuudesta suosituimmissa selaimissa ja mobiililaitteissa. Canvas toimii yleisesti ottaen hyvin uusimmissa selaimissa. Canvas toimii useassa mobiililaitteessa, mutta monimutkaisempien animaatioiden kohdalla saattaa esiintyä

pätkimistä ja muita ongelmia. Testatessani canvasilla tehtyä grafiikkaa ja animaatioita mobiililaitteilla totesin monimutkaisten animaation välillä pätkivän, reagoivan kosketukseen jäljessä tai piirtyvän näytölle väärin. Canvasin toimivuus erilaisissa ympäristöissä on mielestäni tärkeää, koska sitä pidetään yleisesti canvasin vahvuutena esimerkiksi Flashiin verrattuna.

Kolmannessa luvussa käsiteltiin piirtämistä HTML5 Canvasille koodin kautta. Canvasin käyttöönotosta esitettiin yksinkertainen malli, jossa JavaScript oli yhdistetty HTML5-tiedostoon selkeyden vuoksi. Olennaista oli asettaa canvas-elementti body-tägin sisään, käyttää getElementById()-metodia canvas-elementin kutsumiseen tietyllä id-arvolla ja määrittää 2d-konteksti getContext()-metodilla.

Teoriaosuuden keskeisin asia canvasin oppimisen kannalta oli 2d-kontekstilla piirtäminen ja canvasin käyttämä koordinaatisto elementtien asemoimiseen. 2d-kontekstin ohjelmointirajapinta sisältää metodeja ja ominaisuuksia polkuihin, suorakulmioihin, tekstiin, siirtymiin, kuvan piirtämiseen, pikselimanipulaatioon ja elementtien tyylittelyyn. Näitä yhdistelemällä voidaan piirtää kuvioita ja luoda erilaisia efektejä. Tärkeintä oli tuoda esiin piirtämisen logiikka, eikä käydä kaikkia syntakseja läpi. Mielestäni opin käyttämään canvasia parhaiten käymällä läpi 2d-kontekstin esimerkkejä ja soveltamalla niitä omiin esimerkkeihin.

Opinnäytetyön teoriaosuuden loppupuolella käytiin läpi canvasille tarkoitettuja JavaScript-kirjastoja. JavaScript-kirjastot tuovat lisäominaisuuksia ja -metodeja canvasin käyttöön helpottaen elementtien piirtämistä ja animoimista. JavaScript-kirjastojen käyttämisen haasteena saattaa olla vaihtelevat syntaksit. Suosittuja JavaScript-kirjastoja ovat muun muassa KineticJS, EaselJS, Paper.js, Fabric.js ja Processing.js.

Opinnäytetyön käytännönosuudessa tehtiin yhden kerran pyörivä banneri KineticJS -JavaScript-kirjaston avulla. Banneri sisältää liikeanimaatioita, kuvioita ja kuvia sekä erilaisia siirtymäefektejä. Toteutuksessa otettiin huomioon joitain bannereiden yleisiä teknisiä vaatimuksia, kuten koko ja kesto. Yleisesti bannerit ovat melko erilaisia tyyliltään, joten sain suunnitteluun melko vapaat kädet. Käytännöntyössä tarkasteltiin banneria vain teknisestä näkökulmasta.

Ongelmia bannerin toteutuksessa KineticJS-kirjastolla oli vähän. Canvasin käyttämän 2d-kontekstin oppimisen jälkeen tietoa oli helppo soveltaa myös KIneticJS-kirjastoa

käytettäessä. Bannerin tekemisessä oli kuitenkin jotain ongelmia, kuten kaksinkertaiset liikeanimaatiot, syntaksivirheet ja liikeanimaatioiden yhdistäminen kuvaolioon. Ratkaisin ongelmat itse kokeilemalla tai tutkimalla esimerkkikoodia internetistä.

Toimivuuden osalta canvas sopii hyvin banneriin. Canvasin käytön etuna on, että banneri toimii useammalla alustalla, kuten pöytäkoneella, älypuhelimella ja tabletilla. Opinnäytetyössä tehty banneri pyöri sulavasti uusimmilla selaimilla ja muutamalla mobiililaitteella, joita kokeilin. Olin yllättynyt bannerin hyvästä suorituskyvystä, koska aiemmin testaamani animaatiot pätkivät enemmän. Voi kuitenkin olla mahdollista, että interaktiivinen banneri toimisi huonommin, mutta tällaiseen liikeanimaatioita sisältävään banneriin canvas on hyvä vaihtoehto.

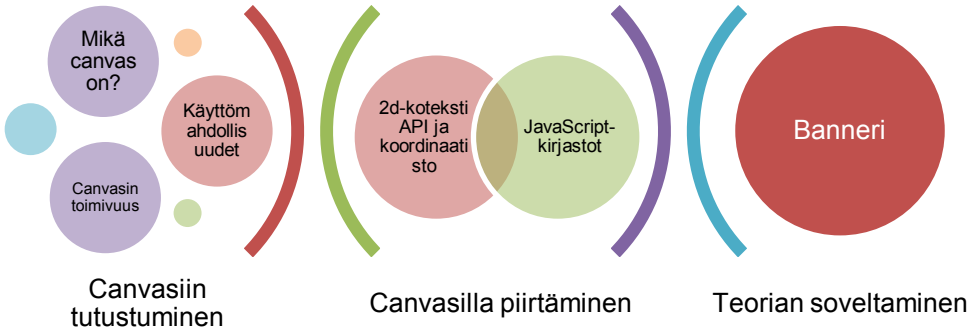
Bannerin kuvanlaatu vaihteli selain- ja laitekohtaisesti. Suhteellisen uusilla laitteilla, kuten älypuhelimella ja tabletilla, bannerin grafiikka näytti hyvältä. PC-tietokoneelta tarkasteltuna varsinkin teksti näytti hieman epätarkalta joissain selaimissa. Tämä johtuu osittain siitä, että canvas on bittikarttagrafiikkaa. Esimerkiksi perinteisiin bannereihin käytetty Flash on vektoripohjainen. Banneria tehdessä olisin voinut ottaa selville, onko vektorigrafiikkaa mahdollista käyttää canvasilla.

Koska käytin banneriin KineticJS-kirjastoa, niin bannerista tuli tiedostokooltaan melko suuri. Bannereihin yleisemmin käytetty Flash-animaatio on tiedostokooltaan pienempi, joten se on lähempänä yleistä tiedostokoon suositusta. Toisaalta canvas ei tarvitse erillistä liitännäistä toimiakseen ja latautuu melko nopeasti. Jälkeenpäin ajatellen olisin voinut kokeilla tehdä bannerin ilman KineticJS-kirjastoa, jolloin tiedostokoosta olisi voinut tulla pienempi. Toisaalta KineticJS teki bannerin tekemisestä huomattavasti helpompaa. Olisin voinut myös käyttää järkevämmin olioita hyväksi, jolloin koodista olisi tullut lyhyempää. Nyt loin jokaiselle elementille oman olion. Tämä johtuu osittain siitä, etten ole vielä tarpeeksi kehittynyt JavaScriptissä ja tämä oli ensimmäinen työni, jossa käytin olioita.

Canvasin käyttö banneriin saattaa tuoda esiin muitakin haasteita. Canvasilla piirretään suurimmaksi osaksi koodin kautta, mikä vaikeuttaa monen suunnittelijan työtä ja rajoittaa täten canvasin käyttöönottoa bannereiden työkaluksi. Joihinkin piirto-ohjelmiin löytyy kuitenkin liitännäisiä grafiikan muuttamiseen canvasille, mutta niitä on vielä aika vähän. Itse pidin koodin kautta piirtämistä positiivisena, koska tarvitsin työskentelyyn suurimmaksi osaksi vain muistiota ja koodin tulos näkyi heti selaimessa.

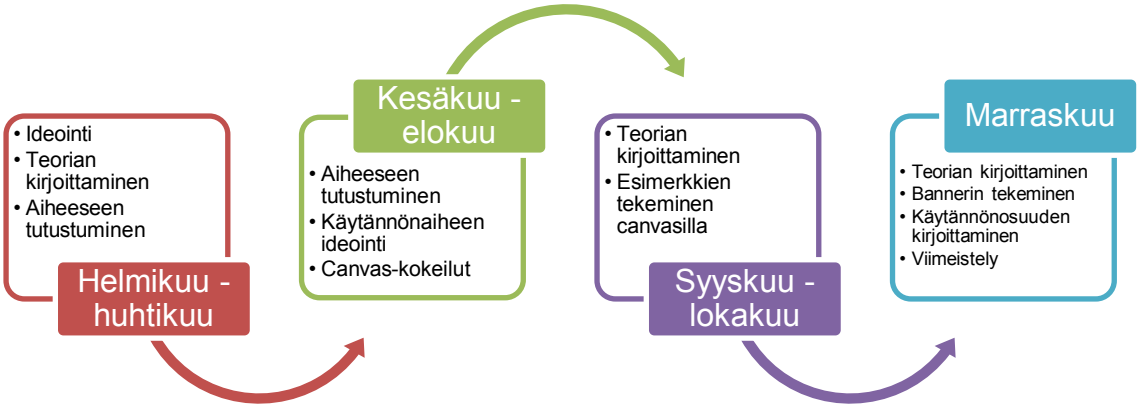
Käytännöntyön lopputulokseksi voin todeta canvasin sopivan opinnäytetyössä olevan tyyppiseen banneriin hyvin varsinkin, jos bannerin halutaan toimivan eri laitteilla ja jos tiedostokoko saa olla hieman suurempi. Canvasilla tehty banneri on varsinkin varteenotettava vaihtoehto mobiililaitteille, koska ne eivät välttämättä toista Flash-animaatioita. Hyvänä lisänä canvas on myös tekstipohjainen eli se on hakukoneiden luettavissa.

Opinnäytetyön tavoitteet toteutuivat hyvin. Opinnäytetyön yhtenä tavoitteena oli toimia oppaana canvasin perusteisiin. Teoriaosuus käsitteli keskeisimmät asiat canvasista, esimerkiksi mikä canvas on ja mitä se sisältää, mitä canvasilla voi tehdä, missä canvas toimii ja miten canvasilla piirretään käytännössä. Toisena tavoitteena oli toteuttaa banneri canvasilla ja tutkia canvasin soveltuvuutta banneriin. Lopputuloksena oli toimiva banneri sekä pohdintaa ja testaamista canvasin soveltumisesta banneriin. Lähdin tutustumaan canvasiin aloittelijan tasolta ja opin opinnäytetyön teoriaosuuden pohjalta soveltaa opittua isompaan kokonaisuuteen eli banneriin. Kuvio 43 kuvaa oppimisprosessiani.



Kuvio 43. Oppimisprosessi canvasin perusteista teorian soveltamiseen.

Opinnäytetyötyöskentelyni ajoittuu vuodelle 2013. Kävin suurimmaksi osaksi opinnäytetyöskentelyni ohessa koulussa tai työharjoittelussa, joten opinnäytetyö eteni melko hitaasti. Arvioin myös aikataulun aika kiireelliseksi, joten en pysynyt alkuperäisessä tavoitteessani ja opinnäytetyön loppupuolella tuli kiire. Aiheeltaan opinnäytetyö vaati arvioitua enemmän perehtymistä ja koin kirjoittamisen välillä hankalaksi. Olen kuitenkin tyytyväinen suorituksestani, koska opintojeni mukainen opinnäytetyön palautus olisi vasta keväällä 2014. Kuvio 44 havainnoi opinnäytetyön tekemiseen käytettyä aikaväliä.



Kuvio 44. Opinnäytetyöhön käytetty aika.

HTML5 Canvas on laaja ja monipuolinen aihe, joten sitä voi tutkia monesta näkökulmasta. Yleisesti canvasiin liittyviä jatkokehitysaiheita voisi olla esimerkiksi jokin uudenlainen interaktiivinen animaatio, responsiivinen canvas, canvasin hyödyntäminen verkkosivuilla, uudenlainen peli, uudentyyppinen canvasilla tehty applikaatio tai canvasin vertaaminen johonkin vaihtoehtoiseen menetelmään. Banneriin ja canvasiin liittyviä aiheita voisi olla muun muassa interaktiivinen banneri tai canvasin hyödyntäminen videobanneriin.

Lähteet

Albers, Brian & Lubbers, Peter & Salim, Frank 2010. Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. New York: Springer Science+Business Media LLC.

Banneri.info 2013a. Bannerit. [verkkosivu] <http://www.banneri.info/bannerit.html> (Luettu 6.11.2013)

Banneri.info 2013b. Suurtaulu. [verkkosivu] <http://www.banneri.info/suurtaulu.html> (Luet-tu 6.11.2013)

Becker, James 2011. Releasing the power of the HTML5 canvas. [verkkosivu] <http://www.bhivecanvas.com/> (Luettu 6.11.2013)

Bradford, Anselm & Haine, Paul 2011. HTML5 Mastery - Sematics, Standards, and Styling. New York: Springer Sci-ence+Business Media LLC.

Bright, Peter 2012. W3C announces plan to deliver HTML 5 by 2014, HTML 5.1 in 2016. Information Technology. [verkkosivu] <http://arstechnica.com/information-technology/2012/09/w3c-announces-plan-to-deliver-html-5-by-2014-html-5-1-in-2016/> (Luettu 30.3.2013)

Canvas Query 2013. Canvas Query. [verkkosivu] <http://canvasquery.com/#canvas-query> (Luettu 6.11.2013)

CreateJS 2013a. CreateJS. [verkkosivu] <http://www.createjs.com/#!/CreateJS> (Luettu 6.11.2013)

CreateJS 2013b. EaselJS. [verkkosivu] <http://www.createjs.com/#!/EaselJS> (Luettu 6.11.2013)

CreateJS 2013c. TweenJS. [verkkosivu] <http://www.createjs.com/#!/TweenJS> (Luettu 6.11.2013)

CreateJS 2013d. SoundJS. [verkkosivu] <http://www.createjs.com/#!/SoundJS> (Luettu 6.11.2013)

CreateJS 2013e. EaselJS Module. [verkkosivu] <http://www.createjs.com/Docs/EaselJS/modules/EaselJS.html> (Luettu 6.11.2013)

Design your way 2013. HTML5 Canvas Wrapper Libraries Based On Javascript. [verkkosivu] <http://www.designyourway.net/drb/html5-canvas-wrapper-libraries-based-on-javascript/> (Luettu 5.11.2013)

Evans, Caleb 2013. jCanvas. [verkkosivu] <http://calebevans.me/projects/jcanvas/> (Luettu 5.11.2013)

Fabric.js 2013. Introduction to Fabric.js. Part 1. [verkkosivu] <http://fabricjs.com/fabric-intro-part-1/#why_fabric> (Luettu 6.11.2013)

Geary, David 2012. Core HTML5 Canvas – Graphics, Animation, and Game Development. New Jersey: Prentice Hall. Kindle-sovellus iPadissa.

Hawkes, Rob 2011. Foundation HTML5 Canvas. New York: Springer Science+Business Media LLC. Kindle-sovellus iPadissa.

Heyes, Richard 2008. RGraph – HTML5 and JavaScript Charts. [verkkosivu] <http://www.rgraph.net/> (Luettu 6.11.2013)

IAB Finland 2006. Mainosten animoinnin pituus. [verkkosivu] <http://www.iab.fi/media/pdf-tiedostot/standardit-ja-oppaat/mainosten-animoinnin-pituus.pdf> (Luettu 6.11.2013)

IAB Finland 2009. IAB:n suositus In Banner -verkkovideomainoksesta. [verkkosivu] <http://www.iab.fi/media/pdf-tiedostot/standardit-ja-oppaat/iab-suositus-in-banner-verkkovideomainokseksi.pdf> (Luettu 6.11.2013)

jCanvaScript 2013. jCanvaScript. [verkkosivu] <http://jcscript.com/> (Luettu 6.11.2013)

Jansson, Julius 2011. Display-mainonta kasvaa – näin teet bannereistasi tehokkaita. [verkkosivu] <http://www.searchbox.fi/Artikkelit/display-mainonta-kasvaa-%E2%80%93-nain-teet-bannereistasi-tehokkaita/> (Luettu 6.11.2013)

Jenkov, Jakob 2013. HTML5 Canvas: Shadows. [verkkosivu] <http://tutorials.jenkov.com/html5-canvas/shadows.html> (Luettu 4.11.2013)

Jyväskylän Yliopisto 2013. Vektorialgebraa. Matemaattisia apuneuvoja. [verkkosivu] <http://users.jyu.fi/~pheikkin/fysa220/Chapter_0.pdf> (Luettu 30.3.2013)

KineticJS 2013a. Enterprise class interactive web graphics. [verkkosivu] <http://kineticjs.com/> (Luettu 5.11.2013)

KineticJS 2013b. Documentation. [verkkosivu] <http://kineticjs.com/docs/> (Luettu 19.11.2013)

KineticJS 2013c. Kinetic Easings. [verkkosivu] <http://kineticjs.com/docs/Kinetic.Easings.html> (Luettu 19.11.2013)

KWD Digital 2013. Banneri. [verkkosivu] <http://www.kwd.fi/sanasto/banneri> (Luettu 6.11.2013)

Lawson, Diederick 2011. HTML Canvas Javascript Library. [verkkosivu] <https://github.com/dkln/canvas_library/> (Luettu 5.11.2013)

Mozilla Developer Network 2013. Canvas tutorial. [verkkosivu] <https://developer.mozilla.org/en-US/docs/HTML/Canvas/Tutorial?redirectlocale=en-US&redirectslug=Canvas_tutorial> (Luettu 18.3.2013)

Overscan 2013. EaselJS: Getting Started. [verkkosivu] <http://www.overscan.co.uk/tutorials/Getting%20Started/> (Luettu 6.11.2013)

Paananen, Petteri 2011. Flash CS4 & CS5 julkaisijan opas. Jyväskylä: WSOYpro Oy.

Paper.js 2013. About. [verkkosivu] <http://paperjs.org/about/> (Luettu 6.11.2013)

Processing.js 2013. A port of the Processing Visualization Language. [verkkosivu] <http://processingjs.org/> (Luettu 6.11.2013)

Pilgrim, Mark 2013. Let's Call It A Draw(ing Surface). Dive Into HTML5. [verkkosivu] <http://diveintohtml5.info/canvas.html> (Luettu 23.3.2013)

Rowell, Eric 2011. HTML5 Canvas Cookbook. Birmingham: Packt Publishing Ltd. Kindle-sovellus iPadissa.

Rowell, Eric 2012a. HTML5 Canvas Element Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/tutorials/html5-canvas-element/> (Luettu 18.3.2013)

Rowell, Eric 2012b. HTML5 Canvas Custom Shape Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/tutorials/html5-canvas-custom-shapes/> (Luettu 1.4.2013)

Rowell, Eric 2013b. HTML5 Canvas Line Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/tutorials/html5-canvas-lines/> (Luettu 1.10.2013)

Rowell, Eric 2013c. HTML5 Canvas Linear Gradient Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/tutorials/html5-canvas-linear-gradients/> (Luettu 30.10.2013)

Rowell, Eric 2013d. HTML5 Canvas Global Composite Operations Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/advanced/html5-canvas-global-composite-operations-tutorial/> (Luettu 23.10.2013)

Rowell, Eric 2013e. HTML5 Canvas Global Alpha Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/advanced/html5-canvas-global-alpha-tutorial/> (Luettu 23.10.2013)

Rowell, Eric 2013f. HTML5 Canvas Text Font, Size, and Style Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/tutorials/html5-canvas-text-font-size/> (Luettu 23.10.2013)

Rowell, Eric 2013g. HTML5 Canvas Image Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/tutorials/html5-canvas-images/> (Luettu 26.10.2013)

Rowell, Eric 2013h. HTML5 Canvas Grayscale Image Colors Tutorial. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/advanced/html5-canvas-grayscale-image-colors-tutorial/> (Luettu 31.10.2013)

Rowell, Eric 2013i. Web Graphics Trends in 2013. Html5 Canvas Tutorials. [verkkosivu] <http://www.html5canvastutorials.com/articles/web-graphics-trends-in-2013/> (Luettu 5.11.2013)

Rowell, Eric 2013j. KineticJS. [verkkosivu] <https://github.com/ericdrowell/KineticJS/wiki> (Luettu 5.11.2013)

Rowell, Eric 2013k. KineticJS. [verkkosivu] <https://github.com/ericdrowell/KineticJS/blob/master/src/Canvas.js> (Luettu 5.11.2013)

Vepsäläinen, Juho 2013. Brief Overview of HTML5 Canvas Libraries.
JSter is a catalog of 1436 JavaScript libraries and tools for web development. [verkkosivu] <http://jster.net/blog/brief-overview-of-html5-canvas-libraries#.Unkj5xAy0f5> (Luettu 5.11.2013)

WHATWG 2013. The canvas element. HTML Living Standard. [verkkosivu] <http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-element> (Luettu 23.3.2013)

W3C 2013. The Canvas Element. HTML 5.1 Nightly – A vocabularity and associated APIs for HTML and XHTML. [verkkosivu] <http://www.w3.org/html/wg/drafts/html/master/embedded-content-0.html#the-canvas-element> (Luettu 17.3.2013)

W3C 2012. HTML Canvas 2D Context. [verkkosivu] <http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas_CR/ > (Luettu 17.3.2013)

W3 Schools 2013a. HTML Canvas Reference. [verkkosivu] <http://www.w3schools.com/tags/ref_canvas.asp> (Luettu 6.10.2013)

W3 Schools 2013b. HTML canvas arc() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_arc.asp> (Luettu 7.10.2013)

W3 Schools 2013c. HTML canvas arc() Method. [verkkosivu] <http://www.w3schools.com/js/js_obj_math.asp> (Luettu 7.10.2013)

W3 Schools 2013d. HTML canvas strokeStyle Property. [verkkosivu] <http://www.w3schools.com/tags/canvas_strokestyle.asp> (Luettu 8.10.2013)

W3 Schools 2013e. HTML canvas fillStyle Property. [verkkosivu] <http://www.w3schools.com/tags/canvas_fillstyle.asp> (Luettu 8.10.2013)

W3 Schools 2013f. HTML canvas globalCompositeOperation Property. [verkkosivu] <http://www.w3schools.com/tags/canvas_globalcompositeoperation.asp> (Luettu 23.10.2013)

W3 Schools 2013g. HTML canvas fillText() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_filltext.asp> (Luettu 26.10.2013)

W3 Schools 2013h. HTML canvas transform() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_transform.asp> (Luettu 27.10.2013)

W3 Schools 2013i. HTML canvas setTransform() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_settransform.asp> (Luettu 27.10.2013)

W3 Schools 2013j. HTML canvas rotate() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_rotate.asp> (Luettu 28.10.2013)

W3 Schools 2013k. HTML canvas scale() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_scale.asp> (Luettu 28.10.2013)

W3 Schools 2013l. HTML canvas translate() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_translate.asp> (Luettu 28.10.2013)

W3 Schools 2013m. HTML canvas getImageData() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_getimagedata.asp> (Luettu 31.10.2013)

W3 Schools 2013n. HTML canvas putImageData() Method. [verkkosivu] <http://www.w3schools.com/tags/canvas_putimagedata.asp> (Luettu 31.10.2013)

Kuviolähteet

Kuvio 2. Interaktiivinen Canvasilla tehty animaatio, jota pystyy liikuttamaan ja josta saa irrotettua terälehtiä.
Rowell, Eric 2013a. <http://www.html5canvastutorials.com/labs/html5-canvas-interactive-flower/>

Kuvio 3 vasenpuoli. Canvasilla tehty piirtoalusta ja kuvagalleria.
Bonomo 2013. <http://bomomo.com/>

Kuvio 3 oikeapuoli. Canvasilla tehty piirtoalusta ja kuvagalleria.
Osmani, Addy 2013. <http://www.addyosmani.com/resources/canvasphoto/>

Kuvio 4 vasenpuoli. Vasemmalla puolella on ampumapeli ja oikealla matemaattinen pulmapeli.
Wagner, Jonas 2009. <http://29a.ch/jswars/>

Kuvio 4 oikeapuoli. Vasemmalla puolella on ampumapeli ja oikealla matemaattinen pulmapeli.
Hyperandroid 2010.<http://labs.hyperandroid.com/mathmayhem#comment-181145353>

Kuvio 5. Video, jonka voi hajottaa palasiksi hiiren klikkauksella.
Christmann, Sean 2010. <http://craftymind.com/factory/html5video/CanvasVideo.html>

Kuvio 6. 2d-konteksilla piirretty kuva.
Rajput, Angad Singh 2012. <http://angadrajput.wordpress.com/2012/07/02/html5-canvas-drawing>

Kuvio 12. Kuutiollinen ja neliöllinen Bézier-käyrä.
Tondeur, Paul 2008. <http://www.paultondeur.com/2008/03/09/drawing-a-cubic-bezier-curve-using-actionscript-3/>

Kuvio 17. GlobalCompositeOperation-ominaisuuden 12 eri vaihtoehtoa.
Rowell, Eric 2013f. <http://www.html5canvastutorials.com/advanced/html5-canvas-global-composite-operations-tutorial/>

Kuvio 23. Käyttäjien kiinnostus JavaScript-kirjastoja kohtaan.
Rowell, Eric 2013i. <http://www.html5canvastutorials.com/articles/web-graphics-trends-in-2013/>

Kuvio 24. KineticJS-kirjaston käyttämä rakenne.
Rowell, Eric 2013j. <https://github.com/ericdrowell/KineticJS/wiki >

Kuvio 25. Täytetty ääriviivallinen suorakulmio, joka on toteutettu KineticJS-kirjastolla. Rowell, Eric 2013l. <http://www.html5canvastutorials.com/kineticjs/html5-canvas-kineticjs-rect-tutorial/>

Kuvio 26. Täytetty ympyrä, joka on toteutettu EaselJS-kirjastolla.
Overscan 2013. <http://www.overscan.co.uk/tutorials/Getting%20Started/>

Kuvio 27. Erilaisia bannerityyppejä.
Banneri.info 2013. <http://www.banneri.info/suurtaulu.html>

Liitteen 1 lähteet

CoffeeScript 2013. CoffeeScript is a little language that compiles into JavaScript. [verkkosivu] <http://coffeescript.org/> (Luet-tu 19.11.2013)

Hawkes, Rob 2011. Foundation HTML5 Canvas. New York: Springer Science+Business Media LLC. Kindle-sovellus iPadissa.

The jQuery Foundation 2013. What is jQuery?. [verkkosivu] <http://jquery.com/> (Luettu 19.11.2013)

Kashyap, Varun 2013. What Is An API & What Are They Good For?. [verkkosivu] <http://www.makeuseof.com/tag/api-good-technology-explained/> (Luettu 3.4.2013)

Koulutuskeskus Salpaus 2013. Oliot. [verkkosivu] <http://edu.phkk.fi/opiskelu/javascript04/oliot.htm> (Luettu 3.4.2013)

Nixon, Robin 2009. Learning PHP, MySQL & JavaScript. O'reilly Media: California.

Paananen, Petteri 2011. Flash CS4 & CS5 julkaisijan opas. Jyväskylä: WSOYpro Oy.

PCmag 2013a. Definition of: Plug-in. [verkkosivu] <http://www.pcmag.com/encyclopedia/term/49395/plug-in> (Luettu 3.4.2013)

PCmag 2013b. Definition of: Widget. [verkkosivu] <http://www.pcmag.com/encyclopedia/term/49395/plug-in> (Luettu 3.4.2013)

2kmediat 2013a. Dynaaminen HTML ja DOM. Artikkelipankki. [verkkosivu] <http://www.2kmediat.com/dhtml/dokumenttimalli.asp> (Luettu 3.4.2013)

2kmediat 2013b. CSS-opas. Artikkelipankki. [verkkosivu] <http://www.2kmediat.com/css/johdanto.asp> (Luettu 3.4.2013)

2kmediat 2013c. JavaScript-opas. Artikkelipankki. [verkkosivu] <http://www.2kmediat.com/jscript/johdanto.asp> (Luettu 3.4.2013)

WHATWG 2013. The WHATWG. FAQ. [verkkosivu] <http://wiki.whatwg.org/wiki/FAQ#What_is_the_WHATWG.3F> (Luettu 3.4.2013)

W3 Schools 2013o. HTML Introduction. Learn to Create Websites. [verkkosivu] <http://www.w3schools.com/html/html_intro.asp> (Luettu 3.4.2013)

W3 Schools 2013p. SVG Tutorial. [verkkosivu] <http://www.w3schools.com/svg/> (Luettu 19.11.2013)

W3C 2013. W3C. [verkkosivu] <http://www.w3.org/> (Luettu 3.4.2013)

Työssä käytettyjä termejä

CoffeeScript

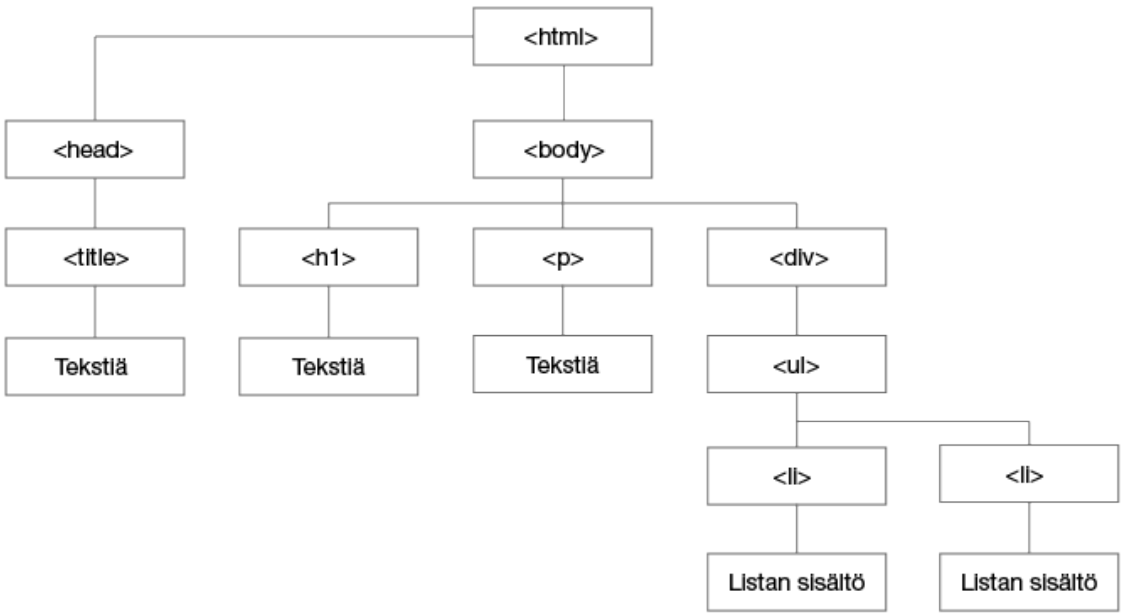
CoffeeScript on pieni kieli, joka kääntyy JavaScriptiksi. CoffeeScriptin tarkoituksena on helpottaa JavaScriptin kirjoittamista yksinkertaisemmalla kirjoitusmuodolla. (CoffeeScript 2013.)

CSS

CSS eli Cascading Style Sheets on sivunkuvauskieli, jota käytetään muun muassa html:n ulkoasun ja esitystavan määrittämiseen (2kmediat 2013b). CSS-tyylillä voidaan määrittää esimerkiksi verkkosivun taustaväri tai -kuva, tekstin ja otsikoiden väri, fontti ja koko sekä html-elementtien esittämistapa. CSS-tyylitiedosto voi olla erillinen ulkoinen tyylitiedosto tai sen voi upottaa esimerkiksi html-sivuun (2kmediat 2013b).

Dokumenttioliomalli eli DOM

DOM tulee sanoista Document Object Model. Dokumenttioliomalli määrittelee, miten elementit välittävät tietoa toisilleen ja miten niihin voidaan viitata dokumentin sisällä. Dokumenttioliomalli muodostaa arvojärjestystä noudattavan porrasteisen dokumenttipuun. Jokainen html-elementti eli solmu (node) on yksittäinen olio (object), jolla on omat metodinsa ja ominaisuutensa. Alla olevassa kuviossa on esitetty yksinkertainen html versio dokumenttioliomalli-puumallista. (Nixon 2009, 314; 2kmediat 2013a.)



Flash

Flash on suosittu multimediaformaatti, jota käytetään muun muassa bannerien, Flash-videoiden, käyttöliittymien, näköisjulkaisujen ja animaatioiden tekemiseen. Flashilla tehty teos voi olla interaktiivinen eli reagoida esimerkiksi käyttäjän hiiren klikkaukseen tai esimerkiksi lineaarisesti etenevä animaatio. Flash tarvitsee toimiakseen liitännäisen. (Paananen, Petteri 2011, 6–7.)

HTML ja html-tägi

HTML tulee sanoista Hyper Text Markup Language. HTML on merkkauskieli, jolla voidaan rakentaa html-dokumentteja eli verkkosivuja. Html-dokumentit sisältävät html-tägejä (<tägi>) ja tekstiä. Html-tägit kuvaavat dokumentin sisällön ja niitä on useimmiten kaksi, aloitus-tägi ja loppu-tägi. Esimerkiksi . (W3 Schools 2013o.)

HTML5

HTML5 on uusin versio html:stä, jota ei ole vielä standardoitu virallisesti. HTML5 tuo mukanaan monta uutta elementtiä muun muassa rakenteeseen ja mediasisältöön sekä lomakkeiden validoimiseen. Uusia rakenteellisia elementtejä ovat muun muassa nav, section, footer, header ja article. Mediaelementtejä ovat canvas, audio ja video. (Hawkes 2011.)

JavaScript

JavaScript on olioperusteinen asiakaspuolen (clientside) skriptikieli verkkodokumenteille. JavaScript tarjoaa oman sisäisen oliomallin, jolla on erilaisia metodeja ja ominaisuuksia. JavaScriptin toiminnallisuutta voidaan lisätä omilla olioilla. JavaScript voidaan upottaa tai kirjoittaa suoraan html-dokumenttiin tai laittaa erilliseen tiedostoon. JavaScriptin on kehittänyt alun perin Netscape 1990-luvulla ja kieli perustuu EcmaScriptiin. (2kmediat 2013c.)

Oliot, metodit ja ominaisuudet

Olio (object) on kokoelma erilaisia ominaisuuksia (properties, attributes) ja metodeja (method). Olion ominaisuudet kuvaavat itse oliota. Ominaisuus voi olla esimerkiksi merkkijono, luku tai toinen omia ominaisuuksia omaava olio. Olion ominaisuuksia käsitteleviä oliokohtaisia funktioita kutsutaan metodeiksi. Metodeja voidaan kutsua esimerkiksi käyttämällä olion nimeä tai metodin nimeä. (Koulutuskeskus Salpaus 2013.)

Esimerkkinä kissa-olio. Kissa-olion ominaisuuksia voi olla kullanväriset silmät, pörröinen turkki ja terävät hampaat. Metodit ovat tehtäviä, joita kissa-olio osaa suorittaa. Esimerkiksi hypätä ja kehrätä.

jQuery

jQuery on yksi suosituimmista JavaScript-kirjastoista. Se yksinkertaistaa muun muassa tapahtumien käsittelyä ja animaatioiden tekemistä. jQuery toimii monissa selaimissa ja on suhteellisen nopea kirjasto. (The jQuery Foundation 2013.)

Liitännäinen eli plugini

Liitännäinen (plugin, plug-in) tarkoittaa ohjelmaa, joka voidaan asentaa olemassa olevaan sovellukseen vahvistamaan sovelluksen kyvykkyyttä. Eli toisin sanoen liitännäisellä lisätään uutta toiminallisuutta sovellukseen. Esimerkiksi liitännäisiä lisätään selaimiin tukemaan erilaista sisältöä, kuten videoita ja ääntä. (PCmag 2013a.) Esimerkkejä liitännäisistä ovat Abobe Flash Player ja Applen Quicktime.

Ohjelmointirajapinta eli API

API tulee sanoista Application Programming Interface. Ohjelmointirajapinta tarkoittaa käyttöliittymää, jossa eri ohjelmat voivat kommunikoida keskenään keskustelemalla toisilleen. Ohjelmointirajapintaa voidaan kuvitella pieniksi koodi-pätkiksi, joiden avulla sovellusten on mahdollista olla vuorovaikutuksessa muiden sovellusten kanssa. Esimerkiksi DOM on ohjelmointirajapinta. Muita esimerkkejä ovat Google Maps API ja Twitter API. (Kashyap 2013; 2kmediat 2013.)

SVG

SVG tulee sanoista Scalable Vector Graphics. SVG tarkoittaa skaalautuvaa vektorigrafiikkaa, joka määritetään xml-kielen kautta. (W3 Schools 2013p.)

WHATWG

WHATWG muodostuu sanoista Web Hyper Text Application Technology Working Group. WHATWG on kasvava ryhmä ihmisiä, joiden mielenkiintona on kehittää internetiä. Se keskittyy pääasiassa HTML:n ja websovellusten ohjelmointirajapintojen kehittämiseen. WHATWG:n perustivat Applessa, Mzillan Foundatiossa ja Operassa työskentelevät yksittäiset henkilöt vuonna 2004. (WHATWG 2013.)

Widget

Widget (välillä suomennettu sanalla vimpain) on kuvakepohjainen käyttöliittymä, joita on esimerkiksi tableteissa ja älypuhelimissa. Widget voi olla myös elementti graafisessa käyttöliittymässä tai pieni sovellus, joka näkyy näytöllä ja on vuorovaikutuksessa käyttäjän kanssa. Widget voi olla esimerkiksi kuvakemainen kello, sääennuste tai laskin. Windows-ympäristössä widget on nimellä gadget. (PCmag 2013b.)

W3C

W3C tulee sanoista World Wide Web Consortium. W3C on kansainvälinen yhteisö, joka kehittää avoimia webstandardeja varmistaakseen webin pitkäaikaisen tulevaisuuden. (W3C 2013.)

Liite 2 Bannerin JavaScript-lähdekoodi

```
// Suorittaa JavaScriptin, kun sivusto on ladattu
window.onload = function() {


// ********************************OLIOT*****************************************

// Luodaan stage-olio ja määritetään canvasin koko
var stage = new Kinetic.Stage({
    container: 'container',
    width: 140,
    height: 350
});

// Luodaan viimeisin tausta
var tausta4 = new Kinetic.Layer();
var rect5 = new Kinetic.Rect({
    x: 0,
    y: 0,
    width: 140,
    height: 350,
    fill: '#ada99f'
});

// Luodaan toiseksi viimeinen tausta
var tausta1 = new Kinetic.Layer();
var rect = new Kinetic.Rect({
    x: 0,
    y: 0,
    width: 140,
    height: 350,
    fill: '#616566'
});

// Luodaan toinen tausta
var tausta2 = new Kinetic.Layer();
var rect2 = new Kinetic.Rect({
    x: 0,
    y: 0,
    width: 140,
    height: 350,
    fill: '#a65752'
});

// Luodaan päällimmäinen tausta
var tausta3 = new Kinetic.Layer();
var rect3 = new Kinetic.Rect({
    x: 0,
    y: 0,
    width: 140,
    height: 350,
    fill: '#d4c6a1'
});

// Luodaan tähti-kuvio
var tahti = new Kinetic.Layer();
var animatedStar = new Kinetic.Star({
    x: 105,
    y: 40,
    numPoints: 9,
    innerRadius: 20,
    outerRadius: 25,
```

```
    fill: '#4d4c48',
    stroke: 'black',
    strokeWidth: 1
});

// Luodaan alareunassa oleva suorakulmio
var suorakulmio = new Kinetic.Layer();
var rect4 = new Kinetic.Rect({
    x: 0,
    y: 230,
    width: 140,
    height: 120,
    fill: '#4d4c48'
});

// Luodaan Matkakuumetta?-teksti
var teksti1 = new Kinetic.Layer();
var text1 = new Kinetic.Text({
    x: 12,
    y: 260,
    text: 'Matkakuumetta?',
    fontSize: 16,
    fontFamily: 'arial',
    fill: '#eee9e3',
    shadowOffsetX: 0.2,
    shadowOffsety: 0.2,
    shadowOpacity: 0.4
});

// Luodaan Varaa matkasi tästä -teksti
var tekstiVaraa = new Kinetic.Layer();
var textVaraa = new Kinetic.Text({
     x: -130,
     y: 255,
     text: 'Varaa\n matkasi\n tästä',
     fontSize: 20,
     lineHeight: 1.3,
     fontFamily: 'arial',
     align: 'center',
     fill: '#eee9e3',
     shadowOffsetX: 0.2,
     shadowOffsety: 0.2,
     shadowOpacity: 0.4
});


// Luodaan Tekstit vaihtuville hinnoille
// Hinta 1
var tekstiHinta1 = new Kinetic.Layer();
var textHinta1 = new Kinetic.Text({
     x: 92,
     y: 29,
     text: '3vrk\n399€',
     fontSize: 12,
     fontFamily: 'arial',
     fill: '#eee9e3',
     shadowOffsetX: 0.2,
     shadowOffsety: 0.2,
     shadowOpacity: 0.4
});

// Hinta 2
var tekstiHinta2 = new Kinetic.Layer();
var textHinta2 = new Kinetic.Text({
```

```
    x: 92,
    y: 29,
    text: '3vrk\n299€',
    fontSize: 12,
    fontFamily: 'arial',
    fill: '#eee9e3',
    shadowOffsetX: 0.2,
    shadowOffsety: 0.2,
    shadowOpacity: 0.4,
    opacity: 0
});

// Hinta 3
var tekstiHinta3 = new Kinetic.Layer();
var textHinta3 = new Kinetic.Text({
    x: 92,
    y: 29,
    text: '3vrk\n499€',
    fontSize: 12,
    fontFamily: 'arial',
    fill: '#eee9e3',
    shadowOffsetX: 0.2,
    shadowOffsety: 0.2,
    shadowOpacity: 0.4,
    opacity: 0
});


// Luodaan tekstit loppunäkymään (hinnat)
var tekstiHinnat = new Kinetic.Layer();
var textHinnat = new Kinetic.Text({
    x: 12,
    y: 29,
    text: 'Pariisi 3vrk 399€\nIstanbul 3vrk 299€\nLontoo 3vrk 499€',
    fontSize: 14,
    lineHeight: 1.4,
    fontFamily: 'arial',
    fill: '#eee9e3',
    shadowOffsetX: 0.2,
    shadowOffsety: 0.2,
    shadowOpacity: 0.2,
    opacity: 0
});

// Luodaan Pariisi-teksti
var teksti2 = new Kinetic.Layer();
var text2 = new Kinetic.Text({
    x: 12,
    y: -20,
    text: 'Pariisi',
    fontSize: 22,
    fontFamily: 'arial',
    fill: '#000'
});

// Luodaan Istanbul-teksti
var teksti3 = new Kinetic.Layer();
var text3 = new Kinetic.Text({
   x: 160,
   y: 80,
   text: 'Istanbul',
   fontSize: 22,
   fontFamily: 'arial',
   fill: '#000'
```

```
});

// Luodaan Lontoo-teksti
var teksti4 = new Kinetic.Layer();
var text4 = new Kinetic.Text({
    x: 160,
    y: 80,
    text: 'Lontoo',
    fontSize: 22,
    fontFamily: 'arial',
    fill: '#000'
});


//************** LUODAAN KUVA-OLIOT (sisältää tweenit) ******************

// Luodaan kuva 1 - Pariisi
var image1 = new Kinetic.Layer();
var imageObj1 = new Image();
      imageObj1.onload = function() {
          var kuva1 = new Kinetic.Image({
                        x: 0,
                        y: 122,
                        image: imageObj1,
                        width: 140,
                        height: 108
            });

       // Lisää kuvion kerrokseen (layer)
       image1.add(kuva1);

      // Lisää kerroksen stage-olioon
      stage.add(image1);

     // Tween-animaatio - kuva siirtyy oikealle
     var tween3 = new Kinetic.Tween({
          node: kuva1,
          duration: 1,
          x: 200,
          easing: Kinetic.Easings.EaseInOut,
          opacity: 1
      });

    // Tween alkaa 4 sekuntin kuluttua
   setTimeout(function() {
        tween3.play();
   }, 4000);

                        };
// Kuvan lähde
imageObj1.src = 'images/paris.png';


// Luodaan kuva 2 - Istanbul
var image2 = new Kinetic.Layer();
var imageObj2 = new Image();
      imageObj2.onload = function() {
          var kuva2 = new Kinetic.Image({
                        x: -142,
                        y: 149,
                        image: imageObj2,
                        width: 142,
                        height: 81
          });
```

```
    // Lisää kuvion kerrokseen (layer)
    image2.add(kuva2);

    // Lisää kerroksen stage-olioon
    stage.add(image2);

   // Tween-animaatio - kuva siirtyy vasemmalta ja häivyttyy takaisin vasemmalle
   var tween5 = new Kinetic.Tween({
        node: kuva2,
        duration: 1,
        x: 0,
        y: 149,
        easing: Kinetic.Easings.EaseInOut,
        opacity: 1,
        onFinish: function () {
                    var tween7 = new Kinetic.Tween({
                        node: kuva2,
                        duration: 1,
                        opacity: 0,
                        x: -140
                    });

                    setTimeout(function() {
                    tween7.play();
                    }, 2000);
                                                        }
        });

    // Tween alkaa 4 sekuntin kuluttua
    setTimeout(function() {
          tween5.play();
    }, 4000);

};
// Kuvan lähde
imageObj2.src = 'images/istanbul.png';


// Luodaan kuva 3 - Lontoo
var image3 = new Kinetic.Layer();
var imageObj3 = new Image();
     imageObj3.onload = function() {
          var kuva3 = new Kinetic.Image({
                    x: 0,
                    y: -110,
                    image: imageObj3,
                    width: 140,
                    height: 108
           });

          // Lisää kuvion kerrokseen (layer)
          image3.add(kuva3);

          // Lisää kerroksen stage-olioon
          stage.add(image3);

         // Tween-animaatio - kuva tippuu ylhäältä ja häivytys
         var image3Tween = new Kinetic.Tween({
                    node: kuva3,
                    duration: 2,
                    x: 0,
                    y: 122,
                    easing: Kinetic.Easings.BounceEaseOut,
```

```
                    opacity: 1,
                    onFinish: function () {
                        var tween9 = new Kinetic.Tween({
                                    node: kuva3,
                                    duration: 1,
                                    opacity: 0,
                                    easing: Kinetic.Easings.Linear
                        });

                        setTimeout(function() {
                                    tween9.play();
                        }, 2500);
                        }
                    });

        // Tween alkaa 8,5 sekuntin kuluttua
                                    setTimeout(function() {
                                                image3Tween.play();
                                    }, 8500);

    };
// Kuvan lähde
imageObj3.src = 'images/london.png';


// Luodaan kuva 4 - Logo
var image4 = new Kinetic.Layer();
    var imageObj4 = new Image();
        imageObj4.onload = function() {
                    var kuva4 = new Kinetic.Image({
                    x: 0,
                    y: -110,
                    image: imageObj4,
                    width: 140,
                    height: 92
                    });

       // Lisää kuvion kerrokseen (layer)
       image4.add(kuva4);

      // Lisää kerroksen stage-olioon
      stage.add(image4);

     // Tween-animaatio - ilmestyy ylhäältä
     var image4Tween = new Kinetic.Tween({
                    node: kuva4,
                    duration: 2,
                    x: 0,
                    y: 122,
                    easing: Kinetic.Easings.BackEaseIn,
                    opacity: 1
     });

     // Tween alkaa 14,5 sekuntin kuluttua
     setTimeout(function() {
         image4Tween.play();
     }, 14500);

};
// Kuvan lähde
imageObj4.src = 'images/logo.png';


// OLIOIDEN LISÄYS KERROKSIIN JA STAGE-OLIOON
```

```
// Lisätään kuviot kerrokseen (layer)
tausta4.add(rect5);
tausta1.add(rect);
tausta2.add(rect2);
tausta3.add(rect3);
suorakulmio.add(rect4);
teksti1.add(text1);
tekstiVaraa.add(textVaraa);
teksti2.add(text2);
teksti3.add(text3);
teksti4.add(text4);
tahti.add(animatedStar);
tekstiHinta1.add(textHinta1);
tekstiHinta2.add(textHinta2);
tekstiHinta3.add(textHinta3);
tekstiHinnat.add(textHinnat);


// Lisätään kerrokset (layer) stage-olioon
stage.add(tausta4);
stage.add(tausta1);
stage.add(tausta2);
stage.add(tausta3);
stage.add(suorakulmio);
stage.add(teksti1);
stage.add(tekstiVaraa);
stage.add(teksti2);
stage.add(teksti3);
stage.add(teksti4);
stage.add(tahti);
stage.add(tekstiHinta1);
stage.add(tekstiHinta2);
stage.add(tekstiHinta3);
stage.add(tekstiHinnat);


// **********TWEENIT**************************************

// Luodaan tween päällimmäiselle taustalle - poistuu vasemmalle
var tween1 = new Kinetic.Tween({
    node: tausta3,
    duration: 1,
    x: 140,
    y: 0,
    easing: Kinetic.Easings.EaseInOut
});

// Tween alkaa 4 sekuntin kuluttua
setTimeout(function() {
    tween1.play();
}, 4000);


// Luodaan Pariisi-tekstin tween - ylhäältä alas ja vasemmalle
var tweenTeksti = new Kinetic.Tween({
    node: teksti2,
    duration: 0.5,
    y: 100,
    easing: Kinetic.Easings.EaseInOut,
    opacity: 1,
    onFinish: function () {
        var tween4 = new Kinetic.Tween({
            node: teksti2,
```

```
            duration: 1,
            x: -100,
            easing: Kinetic.Easings.StrongEaseOut
        });
        setTimeout(function() {
            tween4.play();
        }, 1500);
    }
});

// Tween alkaa sekuntin kuluttua
setTimeout(function() {
    tweenTeksti.play();
}, 1000);


// Luodaan tween toiselle taustalle - liikkuu alaspäin
var tween2 = new Kinetic.Tween({
    node: tausta2,
    duration: 1,
    x: 0,
    y: 350,
    easing: Kinetic.Easings.EaseInOut,
    opacity: 0.5
});

// Tween alkaa 8,3 sekuntin kuluttua
setTimeout(function() {
    tween2.play();
}, 8300);


// Luodaan Istanbul-tekstin tween - oikealta vasemmalle ja takaisin oikealle
var tweenTeksti2 = new Kinetic.Tween({
    node: teksti3,
    duration: 0.5,
    x: -150,
    easing: Kinetic.Easings.EaseInOut,
    opacity: 1,
    onFinish: function () {
        var tween6 = new Kinetic.Tween({
            node: teksti3,
            duration: 1,
            x: 150
        });
        setTimeout(function() {
            tween6.play();
        }, 2000);
    }
});

// Tween alkaa 4,5 sekuntin kuluttua
setTimeout(function() {
    tweenTeksti2.play();
}, 4500);


// Luodaan Lontoo-tekstin tween - oikealta vasemmalle ja häivytys
var tweenTeksti4 = new Kinetic.Tween({
    node: teksti4,
    duration: 0.5,
    x: -150,
    easing: Kinetic.Easings.EaseInOut,
    opacity: 1,
```

```
        onFinish: function () {
            var tween8 = new Kinetic.Tween({
                node: teksti4,
                duration: 1,
                opacity: 0,
                easing: Kinetic.Easings.Linear
        });
        setTimeout(function() {
            tween8.play();
        }, 2500);
    }
});

// Tween alkaa 10 sekuntin kuluttua
setTimeout(function() {
    tweenTeksti4.play();
}, 10000);


// Luodaan tween tähdelle - häivytys
var tweenTahti = new Kinetic.Tween({
    node: animatedStar,
    duration: 1,
    opacity: 0,
    easing: Kinetic.Easings.Linear
});

// Tween alkaa 13 sekuntin kuluttua
setTimeout(function() {
    tweenTahti.play();
}, 13000);


// Luodaan tween toiseksi viimeisimmälle taustalle - häivytys
var tweenTausta1 = new Kinetic.Tween({
    node: tausta1,
    duration: 2,
    opacity: 0,
    easing: Kinetic.Easings.Linear
});

// Tween alkaa 14 sekuntin kuluttua
setTimeout(function() {
    tweenTausta1.play();
}, 14000);


// Luodaan tween Matkakuumetta-tekstille - poistuu kuvasta oikealle
var tweenMatkakuume = new Kinetic.Tween({
    node: text1,
    duration: 1,
    x: 170,
    easing: Kinetic.Easings.EaseInOut,
    opacity: 0.5
});

// Tween alkaa 14 sekuntin kuluttua
setTimeout(function() {
    tweenMatkakuume.play();
}, 14000);


// Luodaan tween Varaa matka tästä -tekstille - teksti ilmestyy näkyviin vasemmalta
var tweenVaraa = new Kinetic.Tween({
```

```
    node: textVaraa,
    duration: 1,
    x: 30,
    easing: Kinetic.Easings.EaseInOut
});

// Tween alkaa 14,5 sekuntin kuluttua
setTimeout(function() {
    tweenVaraa.play();
}, 14500);


// Luodaan tween hinnalle 1 - häivytys
var tweenHinta1 = new Kinetic.Tween({
    node: textHinta1,
    duration: 1,
    opacity: 0,
    easing: Kinetic.Easings.Linear
});

// Tween alkaa 3 sekuntin kuluttua
setTimeout(function() {
    tweenHinta1.play();
}, 3000);


// Luodaan tween hinnalle 2 - hinta näkyviin ja häivytys
var tweenHinta2 = new Kinetic.Tween({
     node: textHinta2,
     duration: 1,
     opacity: 1,
     easing: Kinetic.Easings.Linear,
     onFinish: function () {
        var tween12 = new Kinetic.Tween({
            node: textHinta2,
            duration: 1,
            opacity: 0,
            easing: Kinetic.Easings.Linear
     });
     setTimeout(function() {
         tween12.play();
     }, 1200);
    }
});

// Tween alkaa 4,5 sekuntin kuluttua
setTimeout(function() {
    tweenHinta2.play();
}, 4500);


// Luodaan tween Hinnalle - Hinta näkyviin ja häivytys
var tweenHinta3 = new Kinetic.Tween({
    node: textHinta3,
    duration: 1,
    opacity: 1,
    easing: Kinetic.Easings.Linear,
    onFinish: function () {
        var tween13 = new Kinetic.Tween({
            node: textHinta3,
            duration: 1,
            opacity: 0,
            easing: Kinetic.Easings.Linear
        });
```

```
      setTimeout(function() {
          tween13.play();
      }, 2600);
    }
});

// Tween alkaa 9 sekuntin kuluttua
setTimeout(function() {
    tweenHinta3.play();
}, 9000);


// Luodaan tween kaupungit + hinnat - teksti ilmestyy näkyviin viimeiseen näkymään
var tweenHinnat = new Kinetic.Tween({
    node: textHinnat,
    duration: 2,
    opacity: 1,
    easing: Kinetic.Easings.Linear
});

// Tween alkaa 14,5 sekuntin kuluttua
setTimeout(function() {
    tweenHinnat.play();
}, 14500);



// Suljetaan window.onload-funktio
};
```