HAAGA-HELIA
ammattikorkeakoulu

HTML5:n multimediatoiminnallisuus

Juha Rötkin

Opinnäytetyösuunnitelma

Tietotekniikan koulutusohjelma

2014

HAAGA-HELIA
ammattikorkeakoulu

Tietotekniikan koulutusohjelma

Tiivistelmä

20.11.2014

Tekijä tai tekijät Juha Rötkin	**Ryhmä tai aloitusvuosi** T3Ti
Opinnäytetyön nimi HTML5:n multimediatoiminnallisuus	**Sivu- ja liitesivumäärä** 52+6
Ohjaaja tai ohjaajat Heikki Hietala	

Opinnäytetyön tarkoituksena oli testata HTML5:n multimediaelementtien toimintaa neljällä työpöytäselaimella. Tarkoitusta varten tehtiin yksinkertaiset testisivut, jolle eri kolme eri elementtiä sijoitettiin. Elementit olivat video, audio ja canvas. Testisivun avulla testattiin elementtien perustoiminnallisuutta, sekä laajempaa toiminnallisuutta itse tehtyjen kontrollien avulla. Sivut toteutettiin HTML5:llä ja lisätoiminnallisuus tehtiin JavaScriptillä.

Tietoperusta käsitteli Internetin ja HTML:n historiaa, erilaisia HTML:n avuksi tarkoitettuja laajennuksia, HTML5:ttä, Flashia sekä Flashin ja HTML5:n välisiä eroja. Erityistä huomiota kiinnitettiin video-, audio- ja canvas-elementtien ominaisuuksiin ja Flashin osalta pääpaino oli samoin video- ja audio-ominaisuuksilla.

Testauksessa testattiin kaikkien selaimien osalta samat asiat, että saatiin vertailukelpoinen lopputulos. Testauksessa käytettiin selaimina Internet Explorer 11:tä, Chromea, Firefoxia ja Operaa. Pääpiirteissään elementit toimivat kaikissa neljässä selaimessa samalla tavalla, joskin pieniä eroavaisuuksia löytyi, eritoten ulkoasun suhteen.

Yhteenvetona voitiin todeta, että HTML5:n multimediaelementit toimivat kaikissa testatuissa selaimissa riittävän hyvin niiden kaupallista käyttöä varten. Kunhan suunnittelussa muistetaan ottaa huomioon erot elementtien ulkoasun kohdalla.

Asiasanat
HTML5, yhteensopivuus, selaimet, multimedia, vertailu, Flash

HAAGA-HELIA
University of Applied Sciences

Abstract

20.11.2014

Degree Programme in Information Technology

Author or authors Juha Rötkin	**Group or year of entry** T3Ti
The title of thesis HTML5 multimedia functionality	**Number of pages and appendices** 52+6
Supervisor or supervisors Heikki Hietala	

This thesis was meant to test the functionality of HTML5 multimedia elements on four desktop browsers. For testing purposes simple testing pages were created. On those pages three different HTML5 elements were placed. The elements were video, audio, and canvas. The test pages were used to test basic functionality of the elements and their wider functionality by creating controls for them. The pages were put together with HTML5 and the added functionality was implemented with JavaScript.

The theoretical part of the thesis deals with the history of the Internet and HTML, extensions for HTML, HTML5, Flash, and the differences between HTML5 and Flash. Careful consideration is given to the features of multimedia elements of HTML5, such as video, audio, and canvas. When examining Flash the main point of interest is likewise its video and audio features.

In testing, the same features were tested in all four browsers to make sure to have consistent data to base conclusions on. The testing was performed with Internet Explorer 11, Chrome, Firefox, and Opera. The elements performed mainly alike in all four browsers, but there were minor differences, especially regarding layout.

In conclusion it could be said that the multimedia elements of HTML5 work in all tested browser well enough for commercial use. However, differences in appearance of the elements between different browsers need to be taken into consideration when designing web pages.

Key words
HTML5, compatibility, browsers, multimedia, comparison, Flash

Sisällys

1 Johdanto

Tämän opinnäytetyön tavoitteena on vertailla keskenään HTML5:n multimediaominaisuuksia Flashin vastaaviin. Opinnäytetyössä käydään läpi HTML:n kehitystä, Flashin tulo osaksi verkkosivuja, Flashin ongelmat sekä HTML5:n mahdollistama multimediasisältö, jolla voidaan ainakin osaksi korvata Flash. Oletuksena minulla on, että HTML5:llä ei pystytä Flashia kokonaan korvaamaan, mutta pelejä lukuun ottamatta sillä voidaan käytännössä välttää Flashin käyttö.

HTLM5 on World Wide Web Consortiumin (W3C) yllä pitämä standardi tai suositus, kuten W3C sitä itse kutsuu. Flash taas on koko nimeltään Adobe Flash ja sitä voidaan käyttää moninaisten kuvagallerioiden, videoiden ja äänitiedostojen toistamiseen. Flashillä on mahdollista myös tehdä esimerkiksi kokonaisia verkkosivuille upotettavia pelejä.

Kysymykset, joihin pyritään vastaukset löytämään, on kolme. Mitä multimediakomponentteja HTML5:ssä on? Miten hyvin ja yhdenmukaisesti ne toimivat eri työpöytäselaimilla? Onko Flashissä sellaisia ominaisuuksia, joita HTML5:llä ei voida korvata?

Tämän opinnäytetyön osana tehdään verkkosivut, joille sijoitetaan HTML5:n multimediakomponentteja. Tämän sivun avulla voidaan eri selaimien välillä vertailla komponenttien toimivuutta, ulkoasua ja asettelua. Oletuksena on, että eri selaimilla HTML5:n komponentit ovat ulkoasultaan eroavaisia ja, että asettelun suhteen saattaa ilmetä pieniä eroja, jotka kuitenkin verkkosivujen yleisen ulkonäön osalta saattavat olla hyvinkin merkittäviä. Oletuksena on myös se, että tähän mennessä on kaikissa tutkittavana olevissa selaimissa saatu HTML5:n multimediakomponentit toimimaan.

Tarkasteltavat selaimet ovat Internet Explorer, Chrome, Firefox ja Opera. Suurimmat ongelmat, jos niitä esiintyy, ovat ennakkotutkimuksen perusteella Internet Explorerissa, joskin senkin HTML5-tuki on nykyisellään varsin riittävä.

2 Internet, HTML:n kehitys ja Flash

2.1 Internetin historia

J.C.R. Licklider oli ensimmäinen ihminen joka kuvaili Internettiä. Licklider kuvaili vuoden 1962 elokuussa Galactic Network nimistä konseptia, jossa tietokoneet ympäri maailman olisivat yhteydessä toisiinsa. Lokakuussa 1962 Lickliderista tuli DARPAn (The Defense Advanced Research Projects Agency) ensimmäinen tietokonetutkimuksen johtaja. DARPAssa toimiessaan hän vakuutti seuraajansa konseptinsa tärkeydestä. (Leiner ym.)

MIT:n (Massaschusetts Institute of Technology) Leonard Kleinrock kirjoitti ensimmäisen tutkielman, 1961, ja kirjan, 1964, datan siirtämisestä paketteina ja sai vakuutettua Leonard Robertsin tekniikan toteuttamiskelpoisuudesta. 1965 Roberts yhdessä Thomas Merrillin kanssa yhdistivät ensimmäistä kertaa puhelinverkon kautta kaksi eri puolilla Yhdysvaltoja olevaa konetta toisiinsa. Testi vahvisti uskon, että maantieteellisesti toisistaan kaukana olevat koneet oli mahdollista yhdistää jakamaan voimavaransa, mutta samalla osoitti, että puhelinverkko oli tarkoitukseen aivan liian tehoton väline ja vahvisti Kleinrockin näkemykset datan siirtämisestä paketteina. (Leiner ym.)

Vuoden 1966 loppupuolella Roberts siirtyi DARPAn palvelukseen kehittämään ajatusta tietoverkoista pidemmälle. Hän laati ja julkaisi suunnitelmansa ARPANETistä (The Advanced Research Projects Agency Network) nopeasti vuoden 1967 aikana. Samassa konferenssissa, jossa hän julkaisi tutkimuksensa, julkaistiin myös toinen tutkimus, jonka aiheena oli pakettidataverkko. Kävi ilmi, että samaa konseptia oli kehitetty useassa eri organisaatiossa ilman, että tutkijat olisivat tienneet toistensa työstä. Toiset kehittäjät olivat RAND Corporation (**R**esearch **AN**d **D**evelopment) ja NPL (National Physical Laboratory). (Leiner ym.)

UCLA (University of California, Los Angeles) valittiin ARPANETin ensimmäiseksi solmuksi pitkälti Kleinrockin ansiosta ja toiseksi solmuksi valittiin SRI (Stanford Research Institute). Lokakuussa 1969 SRI:n solmu yhdistettiin UCLA:n solmuun ja ARPANETin ensimmäinen yhteys oli avattu. Vuoden 1969 lopussa ARPANETissä oli

yhteensä neljä solmua ja Internetin ensimmäiset todelliset askeleet oli otettu. (Leiner ym.)

Tulevina vuosina ARPANET:iin lisättiin solmuja kiihtyvällä vauhdilla ja joulukuussa 1970 saatiin valmiiksi ensimmäinen host-to-host protokolla nimeltään NCP eli Network Control Protocol ja vuosina 1971 ja 1972 voitiin siirtyä kehittämään sovelluksia. Vuonna 1972 saatiin valmiiksi ensimmäinen sähköpostisovellus jolla pystyttiin lukemaan ja välittämään viestejä ja vastaamaan viesteihin. (Leiner ym.)

Pikkuhiljaa ARPANET kasvoi Internetiksi siinä muodossa kuin me sen nykyään tunnemme. ARPANET:in rinnalle kasvoi muita tapoja välittää datapaketteja, kuten esimerkiksi satelliittien ja radioverkkojen kautta. Tapoja liittyä verkkoon ei rajoitettu, vaan kaikki tavat olivat sallittuja. Internetin osaverkot saattoivat olla teknologialtaan erilaisia ja niillä saattoi olla erilaisia käyttöliittymiä ja rajapintoja ja sijaita missä vain. (Leiner ym.)

Avoimen arkkitehtuurin käsite oli lähtöisin Bob Kahnilta. Kahnin lähdettyä DARPAlta hän siirtyi kehittämään avointa arkkitehtuuria, aluksi radiopakettidatan välityksen kehitysohjelman osana ja myöhemmin omana kehitysohjelmanaan. Kahn aikoi ensin jatkaa NCP:n käyttöä myös radioverkoissa, mutta NCP:n rajoituksien vuoksi se ei ollut mahdollista. Yksi suurimmista rajoitteista oli se, että NCP:tä ei ollut suunniteltu lainkaan huomioimaan tiedonsiirtovirheitä. Yksikin puuttuvat paketti riitti keskeyttämään tiedonsiirron. Koska Kahn työskenteli verkkomallin parissa, jossa virheet tiedonsiirrossa olivat hyvinkin todennäköisiä, piti hänen kehittää NCP:stä edistyneempi versio, joka otti huomioon avoimen arkkitehtuurin tarpeet. Tämän uuden protokollan nimeksi tuli aikanaan TCP/IP (Transmission Control Protocol/Internet Protocol). Kahnin ja Vint Cerfin ensimmäinen tutkielma aiheesta käsitteli protokollaa nimeltä TCP, joka sisälsi siirto- ja välityspalvelut Internetin tarpeisiin. TCP:n ensimmäinen versio ei pystynyt kunnolla käsittelemään tilanteita, joissa paketteja katosi siirron aikana. Tämä johtii TCP:n jakamiseen kahteen eri osaan, joista IP hoiti pakettien välityksen ja TCP:n vastuulla oli virhetilanteiden korjaaminen. (Leiner ym.)

Alun perin kehitys jaettiin kolmeen osaan. Alkuperäiseen ARPANETiin, pakettien radiosiirtoon ja pakettien siirtoon satelliittien välityksellä. Ensimmäinen virstanpylväs oli näiden kolmen eri osa-alueen yhteistoimivuus TCP:n avulla. Alun perin TCP:tä pidettiin liian raskaana tavallisten pöytäkoneiden liittämiseen Internetiin, mutta MIT:n lisätutkimus ja kehitys tekivät mahdolliseksi TCP:n käytön myös pöytäkoneilla. PC-koneiden ja lähiverkkojen kehitys 80-luvulla muutti Internetin painopistettä suurkoneilta pienien työasemien suuntaan. Tämä kehitys, sekä lähiverkkojen yleistyminen, saivat aikaan sen, että pelkkä numeraalinen nimiavaruus kasvoi liian suureksi helpon käytön kannalta ja johti DNS:n (Domain Name System) kehitykseen. DNS teki mahdolliseksi hierarkkisen nimiavaruuden kehittämisen, joka on edelleen käytössä. Toinen merkittävä askel oli TCP/IP:n sisällyttäminen UNIX-käyttöjärjestelmään, jonka avulla voitiin ratkaista palvelinkoneiden yhteysprotokollien päivittämisen ongelmat. Vuoteen 1985 mennessä Internet oli jaettu kahteen osaan, MILNETiin asevoimien tarpeisiin ja ARPENETiin tutkijoiden ja muiden käyttäjien tarpeita varten. (Leiner ym.)

Pikkuhiljaa Internetin kehityksen rahoitus alkoi siirtyä pois valtioiden käsistä ja erilaiset yritysten rahoittamat tutkimusryhmät siirtyivät Internetin kehityksen etualalle. Koska Internetin kehitykseen ottivat osaa nyt useammat eri ryhmät, sai standardointi yhä tärkeämmän roolin. Internet Society perustettiin 1991 ja vuonna 1994 oli W3C:n (World Wide Web Consortium) vuoro. Vuonna 1995 FNC (Federated Networking Council) julkaisi termin Internet virallisen määritelmän. Määritelmä oli pääpiirteissään, että termi Internet tarkoitti maailmaanlaajuista tietoverkkoa, jonka osat on yhdistetty toisiinsa loogisesti yksilöllisten IP:hen perustuvien osoitteiden tai IP:n tulevien kehitysversioiden avulla, kommunikointi tapahtuu TCP/IP:n tai sen seuraajien avulla ja että se käyttää tai tekee mahdolliseksi käyttää ylemmän tason palveluita ja niihin liittyvää infrastruktuuria. (Leiner ym.)

2.2 HTML:n kehitys

HTML eli HyperText Markup Language on kieli, jolla kuvataan verkkosivujen sisältöä sekä linkkejä toisiin osiin sivustoa tai toisille sivustoille eli hyperlinkkejä. Kuvaukseen käytetään erilaisia elementtejä, jotka kertovat selaimelle minkä muotoista sisältöä on odotettavissa ja millaisena se on tarkoitus näyttää. Elementeillä on erikseen alku- ja loppumerkinnät. (Beal.)

Hyperteksti ajatuksena on lähtöisin jo 1940-luvulta, mutta se ajatuksen tasolta todellisuudeksi se muuttui vasta 1980-luvulla, kun henkilökohtaiseen käyttöön tarkoitetut tietokoneet alkoivat yleistyä. Applen Bill Atkinson teki ensimmäisen hypertekstiä hyödyntävän sovelluksen 1980-luvun lopulla. (Longman 1998)

Kun World Wide Webin (WWW) keksijäksi luettu Berners-Lee teki omaa työtään, hän totesi hypertekstin olevan WWW:iin toimiva ratkaisu, mutta kaikki sillä hetkellä olemassa olevat valmiit ratkaisut olivat laitteistosidonnaisia, pystyivät linkittämään tietoja vain yhdeltä koneelta ja lisäksi ne olivat monimutkaisia käyttää. Berners-Lee päätyikin kehittämään oman protokollansa tietojen siirtoon joka nykyään tunnetaan nimellä HyperText Transfer Protocol eli HTTP. (Longman 1998)

HTTP:n lisäksi Berners-Lee kehitti yksinkertaisen tekstiformaatin hypertekstille, jonka nimeksi tuli HTML. Berners-Lee ei keksinyt pyörää uudestaan, vaan otti kielen rakenteet toisesta jo olemassa olevasta kielestä, joka oli Standard Generalized Markup Language (SGML). SGML:ssä ei ollut lainkaan hypertekstiominaisuuksia, mutta sen Berners-Lee korjasi helposti lisäämällä kieleen siitä puuttuvat pari pientä ominaisuutta ja tuli näin luoneeksi ensimmäisen version HTML:stä. (Longman 1998)

Berners-Lee ei suinkaan jäänyt hautomaan keksintöään yksin vaan jakoi sen Internetin kautta kiinnostuneille tahoille. Erinäiset ryhmät ja henkilöt alkoivatkin kehittää HTML:ää eteenpäin, sekä rakentamaan ensimmäisiä selaimia. Yksi ensimmäisistä selaimista oli Lynx, joka oli täysin tekstipohjainen, Mosaic puolestaan toi kuvat WWW:hen ja Mosaicin kehittäjien ansiota onkin IMG-tagi, jonka avulla kuvat edelleen usein näytetään. Tässä vaiheessa eli 90-luvun alkupuolella isot yritykset eivät vielä HTML:llä lämmenneet, vaan se oli edelleen enimmäkseen akateeminen projekti. (Longman 1998)

Vuonna 1994 kehitys jatkui kiihtyvällä vauhdilla. HTML 2 näki päivänvalon, Netscape perustettiin, samoin kuin W3C. Netscape jatkoi HTML:n kehittämistä omista lähtökohdistaan ja W3C perustettiin ainakin osaksi vastavoimaksi Netscapelle, vaalimaan avoimen standardin periaatteita. 1995 HTML 3 tuotiin julkisuuteen. HTML 3:n ongelma oli alusta asti sen laajuus. HTML 3 oli niin laaja, että selaimien kehittäjät valitsivat

vain osan sen ominaisuuksista käyttöönsä. Eri selaimien kehittäjät ilmoittivat tukevansa HTML 3-standardia, mutta käytännössä eri selaimet tukivat eri osia standardista ja tekivät siihen omia lisäyksiään. Tämän vuoksi HTML 3 ei lopulta ollut sanan varsinaisessa merkityksessä standardi laisinkaan. (Longman 1998)

2.3 HTML 4

HTML Working Group alkoi kehittää uutta standardia vuonna 1997. Aluksi uusi standardi tunnettiin nimellä Cougar. HTML 4 sisälsi monia uusia ominaisuuksia aiempiin HTML-versioihin nähden ja oli askel uuteen monimutkaisempaan, mutta rikkaampaan verkkokehitykseen. HTML 4 toi tullessaan muun muassa laajemmat lomakkeet, skriptauksen, tuen muillekin kielille kuin englannille ja tyylisivut. (Longman 1998)

Ensimmäinen versio HTML 4:stä tuli ulos jo vuoden 1997 lopulla. Päivitettyä versiotakaan ei tarvinnut odottaa kuin huhtikuuhun 1998. W3C:n 4.01 suositusta saatiin sentään odottaa vuoteen 1999. HTML 4.01 versiossa ei ollut suuria eroja ensimmäisiin versioihin, vaan lähinnä kyse oli pienistä parannuksista esimerkiksi taaksepäin yhteensopivuuteen. HTML 4 oli edellisiin versioihin nähden suuri harppaus eteenpäin. (W3C 1999.)

2.4 HTML 4:n lisäosat

Koska HTML 4 ehti olla käytössä vuodesta 1999 ilman merkittäviä muutoksia, se sai avukseen vuosien varrella useita kolmansien osapuolien lisäosia paikkaamaan puutteita ja lisäämään uusia ominaisuuksia. Lisäosista tässä kappaleessa käsitellään Ajax, Silverlight ja Flash. (W3C 1999.)

Oma kokemukseni viiden verkkosivujen kehityksen parissa vietetyn vuoden ajalta on, että ilman HTML 4:n lisäosia nykyaikaisten verkkosivujen tekeminen ennen HTML5:n markkinoille tuloa olisi ollut vähintäänkin vaikeaa, ellei jopa mahdotonta. Lisäosien tuoma etu toiminnallisuuden ja elementtien määrän osalta on ollut hyvin merkittävä. Voidaankin siis sanoa, että lisäosat vastasivat olemassa olevan tarpeen tyydyttämiseen.

2.4.1 Ajax

Ajax on lyhenne sanoista Asynchronous JavaScript + XML ja se ei ole mikään yksittäinen työkalu, vaan kokoelma useita erillisiä työkaluja. Ajax mahdollisti tiedon käsittelyn käyttäjän koneella ilman, että sitä välillä lähetettiin palvelimelle. (Garrett 2005)

Ajax sisälsi ulkoasun yhdenmukaisen muokkauksen XHTML:n ja CSS:n avulla, tiedon dynaamisen esittämisen DOM:in (Dynamic Object Model) avulla, tiedon vaihdon ja manipuloimisen XML:n ja XSLT:n kautta, XMLHttpRequestin, jolla voi hakea uutta tietoa palvelimelta lataamatta sivua uudelleen ja tietenkin JavaScriptin, jolla kaikkea tätä pystyttiin ohjaamaan. (Garrett 2005)

Perinteisesti verkkosivut toimivat niin, että kun verkkosivu pyytää uutta tietoa, niin tieto haetaan palvelimelta, mukaan lukien HTML- ja CSS-tiedot. Kun väliin lisätään Ajax, niin verkkosivu pyytää tiedot Ajaxilta, joka puolestaan pyytää tiedot palvelimelta. Näennäisesti ero ei vaikuta suurelta, mutta erona aiempaan malliin on se, että koska Ajax käsittelee tiedon verkkosivun puolesta, niin verkkosivua ei tarvitse ladata kokonaan uudestaan, vaan Ajax muokkaa vain tarvittavat osat sivusta. Tämä pienentää kaistantarvetta ja latausaikoja huomattavasti, koska Ajax käsittelee tietoja käyttäjän omalla koneella (Garrett 2005)

Ajaxin XMLHttpRequest oli jo vuonna 2005 osa Microsoftin ActiveX:ää ja toimi natiivisti Mozillan ja Applen selaimilla ja pian se muuttuikin osaksi verkkokehittäjien vakiotyökalupakkia, jota ilman oli vaikea tulla toimeen. Muutama esimerkki Ajax-frameworkeistä on esimerkiksi JQuery, JQueryUI sekä ASP.NET Ajax. (McLellan 2005)

2.4.2 Silverlight

Microsoftin Silverlight oli erikseen asennettava laajennus, joka oli tarkoitettu paikkaamaan HTML 4:n puutteita multimediasisällön toistamisessa. Näin ollen se oli suora kilpailija Flashille. Ensimmäinen versio Silverlightista tuli ulos vuonna 2007 uusin päivitysversio Silverlight 5:stä on niinkin tuore kuin heinäkuulta 2014. (Microsoft b.)

Ensimmäinen versio Silverlightista oli hyvinkin puutteellinen ominaisuuksiltaan. Se oli tarkoitettu lähinnä alustaksi myöhemmille päivityksille ja sillä pystyi toistamaan yksinkertaisia animaatioita ja valmiita kontrolleja ei juuri tekstinsyöttöä lukuun ottamatta ollut. 2008 Microsoft sai ulos ensimmäisen kunnollisen version, Silverlight 2:en, joka mahdollisti .NET pohjaisten komponenttien käyttämisen muillakin alustoilla, kuin Windowsilla. Muut alustat olivat toki vain ja ainoastaan Mac OS ja virallista versiota Linuxille ei koskaan tehty. Silverlightiin lisättiin uusia ominaisuuksia jatkuvasti ja Microsoft panostikin siihen vakavasti omana versionaan Flashistä (Bugnion 2010.)

Vuonna 2011 Microsoft ehti jo ilmoittamaan, että Silverlight ei tule olemaan osa Windows 8:an selainta ja näin ollen käytännössä lopetti Silverlightin kehityksen kokonaan. Perusteluna oli se, että selainlisäosat ovat historiaa ja täysin tarpeettomia HTML 5:n tullessa laajaan käyttöön. Lisäksi oli tietenkin sellainen seikka, että Windows 8:n Internet Explorerin ei ollut silloisen suunnitelman mukaan mahdollista käyttää lisäosia lainkaan. Samalla tietenkin suljettiin ulos Windows 8:n selaimesta myös kaikki muut lisäosat kuten Flash ja PDF Reader. (Allen 2011)

Myöhemmin Microsoft pyörsi päätöksensä ja Windows 8:n selain tukee nykyään lisäosia ja Silverlight on yksi tuettavista lisäosista. Silverlight kuitenkin vaatii aina lisäosan asentamisen päinvastoin kuin esimerkiksi HTML5. (Microsoft a.)

2.4.3 Flash

Flash aloitti elämänsä SmartSketch nimisenä piirto-ohjelmana PenPaintille. PenPaintin kohdattua loppunsa SmartSketchistä tehtiin vektorianimaatiotyökalu ja se sai uudeksi nimekseen FutureSplash Animator ja sen toiminta laajennettiin usealle eri alustalle. Vuonna 1996 Macromedia osti FutureSplashin ja nimesi sen uudelleen Flashiksi. Flashin todellinen voittokulku alkoi vuonna 2005 Adoben ostettua sen. Kehittäjät alkoivat herätä siihen, että Flashilla pystyi tekemään paljon asioita suhteellisen helposti ja samalla saattoi säästää latausajoissa. Kun Youtube saavutti suuren suosion, useat kehittäjät lähtivät valloittamaan omaan nurkkaansa uudelta markkinalta ja Flash osoittautui hyväksi alustaksi käyttäjien omille videolatauksille ja Flash levisikin laajaan käyttöön. (Aune 2014)

Flashin alamäki alkoi 2007, kun Apple julkaisi ensimmäisen iPhonensa. Ennen iPhonen julkaisua Adobe kävi neuvotteluja Applen kanssa Flashin sisällyttämisestä iPhonen työkalupakkiin. Neuvottelut kuitenkin kariutuivat, koska Flashistä ei saatu riittävän luotettavasti toimivaa. Samaan aikaan Youtube päätti tarjota videonsa mobiiliystävällisessä formaatissa ja samalla teki Flashin tarpeettomaksi videotoistossa mobiililaitteiden kannalta. Seuraavien vuosien aikana videoiden toisto siirtyi yhä laajemmin tukemaan formaattia, jota valtaosa käyttöjärjestelmistä ja mobiililaitteista pystyivät toistamaan ilman ylimääräisiä ohjelmia. Flashin markkinaosuus jatkoi laskuaan kunnes vuonna 2010 iPadin julkaisun aikaan Flashistä ei ollut puhettakaan iPadin yhteydessä muuten kuin pahassa mielessä ja Steve Jobs arvostelikin rankasti Flashia.
(Warren 2012.)

Flash on kuitenkin edelleen käytössä laajalti esimerkiksi pelisivustoilla, kuten Kongregate. Adobe ilmoittikin lopettavansa Flashin kehityksen marraskuussa 2011. Toistaiseksi kuitenkaan Flash ei toimi esimerkiksi Windows Phonessa tai iPhonessa. (Ulanoff 2011.)

2.5 HTML5

HTML5:n kehitys aloitettiin vuonna 2004. HTML 4.01:en viimeisestä päivityksestä oli ehtinyt vierähtää vuonna 2004 jo useampi vuosi. Vuonna 2008 W3C sai valmiiksi ensimmäisen luonnoksensa HTML5:stä. Vuoteen 2012 mennessä W3C sai ulos Candidate Recommendation-version, jota voidaan pitää ensimmäisenä valmiina versiona. (W3C 2012) W3C:n on tarkoitus saada valmiiksi ensimmäinen virallinen suositus HTML5:stä tänä vuonna ja HTML 5.1:en on aikataulun mukaan samassa vaiheessa vuonna 2016. (W3C)

HTML5 sisältää lukuisia uusia ominaisuuksia verrattuna HTML 4:ään. Uusia elementtejä on tullut dokumentin rakenteen parempaan määrittelyyn, audio- ja videotiedostojen näyttämiseen, bittikarttagrafiikoiden dynaamiseen muodostamiseen, combo boxien tekoon sekä esimerkiksi oma elementtinsä vaikkapa tiedostojen latauksen edistymisen näyttämistä varten. (W3C 2011.)

2.5.1 Rakenteen määrittely

Rakenteen suhteen iso asia ovat header ja footer elementit. HTML5:ssä on mahdollista yhdelle sivulle rakenteellisesti määritellä useampi alasivu. Header ja footer elementit pysyvät näillä kaikilla alasivuilla samoina. Täten headeriin voidaan sijoittaa esimerkiksi otsikkotiedot tai valikkorakenteet. Valikkoa ei siis enää tarvitse rakentaa erilliseksi master-sivuksi tai muulla vastaavalla tavalla. Footeriin voidaan vastaavasti sijoittaa vaikkapa yhteys- ja tekijänoikeustiedot. (W3C 2011.)

Samaten rakenteen uusia elementtejä on luotu kuvaamaan paremmin sisältöä. Article-elementin on tarkoitus pitää sisällään yksi kokonaisuus, joka voi olla esimerkiksi kirjaimellisesti lehtiartikkeli tai muu vastaava. Mielenkiintoinen ajatusmalli on aside-elementti, jonka sisään on suosituksen mukaan tarkoitus sijoittaa materiaali, joka liittyy muuhun sivuun vain löyhästi. Mieleen tulevat helposti esimerkiksi mainokset, jotka liittyvät aiheeseen, jota sivulla käsitellään tai lista aiheeseen liittyvistä muista artikkeleista. Tämän kaltaiset elementit ovat osa HTML5:n semanttisuutta ja ne auttavat jäsentämään sivun rakenteen niin, että sivun lähdekoodi on helpommin tulkittavissa. Tämä auttaa selaimia tunnistamaan erilaista sisältöä ja ei pidä tietenkään väheksyä asian tärkeyttä verkkosivujen kehittämisen selkeyden kannalta. (W3C 2011.)

2.5.2 Multimedia

Tämän opinnäytetyön kannalta oleellisimmat uudet elementit ovat video, audio ja canvas. Kyseiset elementit on HTML5:ssä nimetty kuvaavasti ja niiden sisällöstä ei täten jää epäselvyyttä. Video-elementti antaa mahdollisuuden näyttää videoita, kuten sen nimestä voi päätellä. Audio luonnollisesti tekee saman audio-tiedostoille ja canvas-elementtiä käytetään grafiikan luomiseen lennossa eli dynaamisesti ja sitä voidaan käyttää esimerkiksi selainpelien näyttämiseen. (HTML Dog.)

2.5.3 Muut HTML5:n uudistukset

Muita uusia elementtejä HTML5:ssä on melkoinen lista, joista tässä on lajitelma mielenkiintoisimpia:

- progress, tiedostojen tai raskaampien tehtävien etenemiseen tarkoitettu elementti.
- meter, elementti, jolla voidaan ilmaista esimerkiksi jäljellä olevaa levytilaa.

- data, ajan ja päivämäärän näyttämiseen tarkoitettu oma elementtinsä.
- bdi, tarkoitettu tekstin leveyden säätelyyn esimerkiksi kolumneissa.
- datalist, voidaan käyttää yhdessä input elementin list atribuutin kanssa muodostamaan combo bokseja.
- keygen, elementti, joka mahdollistaa suoran tuen avainparien muodostamiseen.
- output, tarkoitettu sekalaisten tulosteiden näyttämiseen.
- input elementin type attribuutin uudet mahdollisuudet, joihin sisältyy esimerkiksi puhelinnumero, hakuteksti, url, päivämäärä ja aika ja paikallinen päivämäärä ja aika.

Kaiken kaikkiaan HTML5:een on lisätty iso tukku pieniä uudistuksia, jotka ovat toki olleet olemassa aiemminkin, mutta ovat vaatineet HTML 4:n lisäksi erilaisten laajennuksien käyttöä. Laajennukset ovat toki ajaneet oman asiansa, mutta mitä vähemmän erillisten laajennuksien kanssa joutuu tekemisiin, sitä varmemmin lopputuloksena oleva sivu toimii kaikissa selaimissa sen sijaan, että osa sivusta lakkaa toimimasta yllättäen selaimen uudessa versiossa. (W3C 2011.)

HTML5 ei edelleenkään sisällä aivan kaikkea mahdollista, vaan laajennuksien kanssa joutuu varmasti tekemisiin, jos tahtoo tehdä kaikki asiat täsmälleen haluamallaan tavalla. Se kuitenkin mahdollistaa entistä suuremman vapauden tehdä sellaiset sivut kuin haluaa ja samalla antaa suuremman varmuuden siitä, että sivut toimivat kaikilla suhteellisen moderneilla laitteilla ja selaimilla. Suurimman ongelman muodostavat luonnollisesti edelleen vanhemmat Internet Explorerin versiot, jotka tukevat HTML5:ttä joko vajavaisesti tai eivät lainkaan, mutta uusimmat versiot Internet Explorerista tekevät HTML5:n kanssa jo aivan siedettävästi yhteistyötä. (HTML5 Test.)

3 HTML5:n multimediakomponentit ja Flash

HTML5:ssä on kolme pääasiallista elementtiä, joita voidaan käyttää multimediasisällön näyttämiseen. Video, audio ja canvas. Näistä video on tarkoitettu video-tiedostojen näyttämiseen, äänen kanssa tai ilman (Korpela 2014, 368), audio äänen toistamiseen (Korpela 2014, 374) ja canvas on tarkoitettu grafiikan näyttämiseen (Korpela 2014, 395). Grafiikka voi olla valmista tai se voidaan canvas-elementtiin tuottaa dynaamisesti. Näin ollen canvas-elementin pitäisi olla sopiva myös pelien ja vastaavien näyttämiseen.

Flash taas on kokonaan oma toimintaympäristönsä, josta löytyy useita eri versioita. Eri versioissa on käytettävissä erilainen valikoima toimintoja. Osa toiminnoista voidaan kevyemmissä versioissa ottaa käyttöön ActionScriptin avulla, mutta ei kaikkia. (Manninen & Marttila 2006, 10.)

3.1 HTML5:n multimediaelementtien tarkempi tarkastelu

Video-elementti on tarkoitettu videotiedostojen toistamiseen. (Korpela 2014, 369). Audio-elementti on vastaavasti tarkoitettu äänitiedostojen toistamiseen (Korpela 2014, 374-375). Canvas-elementtiä ei ole tarkoitettu lainkaan staattisen kuvan esittämiselle vaan elementille sijoitetaan joko JavaScriptillä piirrettyä grafiikkaa ja/tai valmiita elementtejä. Valmiit elementit voivat toki olla myös staattisia kuvia. Canvas-elementti mahdollistaa myös interaktiivisen käytön. (Korpela 2014, 395.)

Video- ja audioelementeillä on yhteinen HTMLMediaElement-liittymä, jossa on multimediatiedostojen käsittelyyn tarkoitettuja ominaisuuksia (Korpela 2014, 379). Canvas-elementillä ei samoja ominaisuuksia ole, mutta canvas-elementille määritellään erikseen piirtokonteksti, joka määrittelee käytettävissä olevat mahdollisuudet. (Korpela 2014, 396.)

3.1.1 Video

Video-elementillä on tavallisten HTML5-elementtien ominaisuuksien lisäksi myös HTMLMediaElementin ominaisuudet. Video-elementti on periaatteessa API-rajapinta, jonka kautta videoita voidaan näyttää. Toistettavien video-formaattien yhteensopivuus riippuu selaimesta ja video-elementti ei sinällään aseta niille rajoituksia. H.264 on laa-

jimmin tuettu formaatti, joka toimii kaikissa muissa tämän opinnäytetyön piirissä olevissa selaimissa paitsi Operassa. H.264 on kaikeksi onneksi yleisin käytettävä formaatti nykyään. Internet Explorerissa tuki on ollut olemassa versiosta 9 lähtien. (Korpela 2014, 369-371.)

Yksinkertaisimmillaan video-elementti tarvitsee vain src-attribuutin eli tiedon siitä, että mistä toistettava videotiedosto löytyy. Oletusarvoisesti elementti ei sisällä kontrolleja videolle eli videota ei voida laittaa käyntiin. Kontrollit voidaan määritellä controls-attribuutin avulla jolloin käytetään selaimen omia kontrolleja. Kontrollit voidaan tehdä myös JavaScriptin avulla. Oletuksena ennen videon käynnistymistä kuvana käytetään videon ensimmäistä kuvaa, mutta poster-attribuutilla voidaan määrittää haluttu kuva. Video-elementille voidaan määritellä src-atribuutin sijaan useampi source-elementti, jolloin se käyttää ensimmäistä videota, jonka formaattia selain tukee. Näin voidaan varmistua siitä, että oli selain mikä tahansa, niin video voidaan näyttää. (Korpela 2014, 368-371.)

Video-elementin height- ja width-attribuuteilla voidaan ilmoittaa elementin koko selaimelle, jotta selain osaa varata sille sopivan tilan. Videon koko voi kuitenkin poiketa käytännössä ilmoitetusta jolloin selain hoitaa skaalauksen automaattisesti, pyrkien kuitenkin pitämään videon kuvasuhteen samana. Videon koon kuitenkin kannattaa olla sopivassa suhteessa sille varattuun tilaan. Jos varattu tila on pienempi kuin videon oikea koko, niin koko video ladataan silti ja siirretään turhaa dataa. Jos varattu tila taas on paljon videon oikeaa tilantarvetta suurempi, niin kuvanlaatu kärsii, kun video skaalataan isommaksi. Jos height- ja width-attribuutteja ei määritellä, video näytetään alkuperäisessä koossaan. (Korpela 2014, 370-371.)

Vaikka kaikissa tarkasteltavissa selaimissa on olemassa tuki video-elementille, niin tuki ei ole kaikissa selaimissa täsmälleen samanlainen. Chromessa esimerkiksi kontrollit piirretään videon alareunan päälle, jolloin osa videosta jää kontrollien alle. Chromessa pitää muistaa erikseen JavaScriptillä piilottaa kontrollit silloin kuin niitä ei tarvita. (Korpela 2014, 372.)

Videon käynnistymistä voidaan ohjata joko preload-attribuutilla tai sitten ottamalla käyttöön autoplay-attribuutti, jolloin video latautuu ja käynnistyy automaattisesti. Preloadilla taas on kolme eri vaihtoehtoa, none, metadata tai auto. Nonen ollessa käytössä selain ei lataa videota tai siihen liittyvää dataa lainkaan. Metadata hakee vain videon metadatan, eli esimerkiksi videon korkeyden ja leveyden, ja auto lataa sekä metadatan, että itse videon ellei selain sitä estä. Preload määritteen oletusarvo on selainkohtainen, mutta suositus on metadata. Eri vaihtoehtojen välinen ratkaisu on tehtävä sivun luonteen mukaan. Itsestään käynnistyvät ääntä sisältävät tiedostot eivät yleensä ole suositeltavia, ellei sitten ole tarkoitus erikseen hakea huomiota videolle. (Korpela 2014, 370.)

3.1.2 Audio

Myös Audio-elementti toimii HTML5:ssä API-rajapintana ja se voidaan luoda JavaScriptillä, jossa on elementin luomiseen oma konstruktori. Audio-elementin sisältö on yksinkertaisimmallaan vain src-attribuutti, joka kertoo audio-datan sijainnin. Audio-elementti ei ole lainkaan näkyvissä ellei sille määritellä kontrolleja. (Korpela 2014, 374-375.)

Aivan kuten video-elementin ollessa kyseessä voidaan audio-elementille määritellä, mitä se lataa valmiiksi preload-attribuutin avulla. Myös autoplayn käyttö on mahdollista, mutta kontrollittoman, eli näkymättömän audio-elementin automaattinen lataaminen ja käynnistäminen, ei ole hyvä idea, koska se on hyvin harvoin käyttäjälle mieleen. Kontrollit voidaan luoda myös erikseen JavaScriptillä, jos audio-elementin omat kontrollit eivät ole halutunlaiset. Audio-elementissäkin voidaan käyttää useampaa erillistä source-elementtiä, jotta voidaan varmistua selaimen tukeman audio-formaatin saatavuudesta. (Korpela 2014, 374-375.)

3.1.3 Canvas

Canvas-elementti on HTML5:n piirtoalusta, joka toimii myös API-rajanpintana monimutkaisempien grafiikoiden ollessa kyseessä. Yksinkertaisimmillaan Canvas-elementtille piirretään valmiita kuvioita. JavaScriptillä määritellään mistä piirtäminen aloitetaan ja minne viiva vedetään. Canvaksen avulla voidaan tehdä myös vuorovaikutteisia esityksiä. (Korpela 2014, 395-396.)

Canvas-elementti vaatii aina toimiakseen piirtokontekstin ja sillä on getContext-metodi JavaScriptin puolella, jolla konteksti määritellään. Käytännössä konteksteja on kaksi, 2d ja WebGL. Canvaksen sisältö voidaan myös muuttaa staattiseksi kuvaksi käyttämällä toDataUrl-metodia. Oletuksena toDataUrl-metodi tuottaa png-kuvan. ToDataUrl-metodi tuottaa png-kuvan myös siinä tapauksessa, että haluttu kovaformaatti ei ole selaimen tukema. ToBlob-metodilla voidaan tuottaa blob-olio. Myös blob-olion sisältämän kuvan formaattina on oletuksena png. Blob-olio on mahdollista pilkkoa useampaan osaan slice-metodilla. Blobille ei ole tukea kaikissa selaimissa. Canvaksen APIt sallivat värien korjaamisen vain kun piirros liitetään tai kun piirros näytetään. (Korpela 2014, 396-397.)

Canvaksen API-rajapinnan avulla on mahdollista tehdä yksinkertaisia pelejä suhteellisen helposti. Avuksi pelin tekoon on olemassa useampia valmiita JavaScript kirjastoja, joiden avulla pelien tekeminen on helpompaa. Näistä mainitsemisen arvoinen on EaselJS, joka muistuttaa toiminnallisuudeltaan ja syntaksiltaan Flashia ja on näin ollen helppo lähtökohta ihmisille, jotka ovat kiinnostuneita pelien teosta HTML5:lle canvaksen avulla. (Rousset)

3.1.4 Video- ja audio-elementin käyttö

Kun halutaan tehdä video- tai audio-elementille kontrollit itse, on turvallisinta määritellä kontrollit controls-attribuutin kautta ja sitten poistaa ne käytöstä JavaScriptillä. Tällöin elementille jää käytettävissä olevat kontrollit vaikkei JavaScript jostain syystä toimisikaan halutulla tavalla. JavaScriptillä tehdyt kontrollit määritellään button-elementeiksi joiden onclick tapahtumille määritellään halutut toiminnot. (Korpela 2014, 379.)

Video- ja audio-elementeillä on olemassa lukuisia attribuutteja. Niille voidaan määritellä tekstiraita tekstitystä varten, ääniraitoja voi olla useampi eri kieliä varten, ääni voidaan laittaa kokonaan pois päältä tai sen tasoa voidaan säätää. Video- tai audio-tiedostossa voidaan liikkua eteen tai taaksepäin tietty määrä, toiston nopeutta voidaan säädellä ja aloituskohta voidaan määritellä erikseen, jos ei haluta aloittaa alusta, ja toisto voidaan tietenkin keskeyttää väliaikaisesti. . (Korpela 2014, 380-381.)

Näiden kahden elementin virhekoodit on määritelty ja niiden avulla voidaan käyttäjälle antaa tarkempaa tietoa kohdatusta ongelmasta. Virhekoodeja on neljä. MEDIA_ERR_ABORTED, joka tarkoittaa, että käyttäjä keskeytti latauksen itse. MEDIA_ERR_NETWORK, joka tarkoittaa, että lataus on keskeytynyt verkkovirheen takia. MEDIA_ERR_DECODE eli mediatiedoston purussa tapahtunut virhe ja MEDIA_ERR_SRC_NOT_SUPPORTED, joka kertoo, että tiedosto on sellaista tyyppiä, jota selain ei voi toistaa. Nämä virheilmoitukset voidaan ottaa kiinni JavaScriptillä ja kertoa käyttäjälle selkokielellä mikä on mennyt pieleen. (Korpela 2014, 381.)

Source-elementissä voidaan käyttää type-attribuuttia, jolla voidaan määritellä mediatiedoston MIME-tyyppi. Tätä voidaan käyttää testaamaan, se että tukeeko selain kyseistä mediatyyppiä canPlayType-metodilla. CanPlayType palauttaa joko tyhjän merkkijonon tai jommankumman arvoista maybe tai probably. Tyhjä merkkijono tarkoittaa, että kyseistä tiedostomuoto ei tueta. Maybe ja probably taas luonnollisesti tarkoittavat, että tiedostomuotoa joko ehkä tuetaan tai sitä todennäköisesti tuetaan. WC3:n suositus on, että arvo probably palautetaan vain jos tiedostomuotoa tuetaan varmasti. Maybe arvon palautus tarkoittaa käytännössä sitä, että tiedostomuotoa tuetaan, mutta sillä voi olla koodekki, jota ei tueta. (Korpela 2014, 382.)

Video- tai audio-tiedoston kestosta voidaan saada tietoa durationin kautta. Duration sisältää tiedoston keston sekunteina. Sekuntimääräisen keston lisäksi duration voi palauttaa arvon NaN, jos kestoa ei ole lainkaan tai Infinity, jos kyseessä on suoratoistotiedosto, jonka kestoa ei ole rajoitettu etukäteen. CurrentTime taas on ominaisuus, jonka avulla saadaan tietää, että missä kohdassa tiedostoa ollaan menossa. Senkin palauttama arvo on aika sekunteina. CurrentTime arvoa muuttamalla voidaan siirtyä tiedossa tietty aikamäärä eteen- tai taaksepäin. Tiedosto voidaan myös määritellä toistumaan rajaton määrä kertoja loop-attribuutilla. (Korpela 2014, 384.)

Video- ja audio-elementeissä on käytössä myös vain luettavissa oleva attribuutti readyState. Tämän attribuutin arvona on mediaelementin tämänhetkinen tila. Arvona on kokonaisluku välillä 0-4 joille on määritelty tunnukset. 0 on HAVE_NOTHING eli mitään tietoa ei ole. 1 on HAVE_METADATA, joka kertoo, että tieto kestosta on saatavissa, videon tapauksessa myös leveys ja korkeus. 2 on HAVE_CURRENT_DATA eli

data on olemassa, mutta dataa ei ole esityksen jatkamiseen. 3 on HAVE_FUTURE_DATA, joka tarkoittaa, että dataa on olemassa myös nykyisestä kohdasta eteenpäin ja viimeisenä on 4 eli HAVE_ENOUGH_DATA, joka ilmaisee, että dataa on tarpeeksi ja dataa siirtyy tarpeellisella nopeudella tiedoston toistamiseen loppuun asti. (Korpela 2014, 384.)

Mediaelementeillä on lisäksi olemassa AudioTrackList- ja VideoTracklist-oliot. VideoTrackList-olio käytännössä sisältää listan eri videoista, joista sitten voidaan valita video, joka video-elementissä halutaan näyttää. VideoTracklistiä käyttämällä voidaan siis säilöä samassa elementissä näytettävien videoiden tieto. AudioTrackListiä voidaan käyttää samaan tarkoitukseen, mutta lisäksi sillä voidaan tarjota samalla videolle useampi ääniraita. Raidoille voidaan määrittää suoraan tieto niiden kielestä, jolloin tuki useammalle kielelle voidaan toteuttaa helposti. (Korpela 2014, 385-386.)

VideoTrackListin ja AudioTracklistin lisäksi medialla voi olla vielä TextTrackList. TextTracklistin avulla voidaan määrittää medialle tekstitykset, aivat kuten AudioTracklistillä voidaan määrittää ääniraita. Yksittäinen TextTrack koostuu kohdistuspisteistä. Jokaisella kohdistuspisteellä on tieto siitä, että milloin sen näyttäminen aloitetaan ja koska se lopetetaan. Käytännössä kyseessä on siis hyvinkin perinteinen tekstitystapa. (Korpela 2014, 390-391.)

3.1.5 MediaController

Mediaelementeille voidaan tehdä yhteinen kontrolleri. Kontrolleri hoitaa usean esityksen koordinoinnin. Kontrollerilla voidaan huolehtia siitä, että eri medioiden toistonopeus ovat samoja sekä, että ne pysyvät synkronoituina. (Korpela 2014, 387-388.)

Mediaelementtien kontrolleri pitää luoda MediaController-konstruktorilla. Mediaelementeissä taas on controller attribuutti, jolla kyseisen mediaelementin kontrolleri voidaan ilmoittaa. Kontrollerilla on monia samoja attribuutteja kuin itse mediaelementeillä eli videolla ja audiolla. Kontrollerin tapauksessa attribuuteilla ohjataan kaikkien niiden mediaelementtien toimintaa joihin kontrolleri on liitetty. Kun yhden mediaelementin toisto pysäytetään tai katkeaa, niin kontrolleri pitää huolen siitä, että kaikki muutkin

toimivat samalla tavalla. Samaten kontrolleri ohjaa kerralla kaikkien mediaelementtien äänenvoimakkuutta. Korpela 2014, 388.)

3.2 Flashin ominaisuudet

Flash koostuu kolmesta eri pääosasta. Osat ovat Flash-ohjelma, jolla esitykset tehdään ja toteutetaan. Toinen osa on ActionScript-ohjelmointikieli, jolla ohjelmoidaan lisätoiminnallisuudet, jotka Flash-ohjelmasta puuttuvat. Kolmantena osana on Flash Player, joka on selainten lisäosa, jolla esitykset näytetään selaimessa. (Manninen & Marttila 2006, 10.)

Itse Flash-ohjelma koostuu erilaisista elementeistä. Ohjelmassa on Stage, aikajana, kymmeniä erilaisia paneeleita, sekä paljon erilaisia valikoita, joista halutut toiminnot löytyvät. Flash-ohjelma onkin oma kehitysympäristönsä, joka sisältää kaikki tarvittavat työkalut esityksien tekemiseen. Flash-ohjelma toimintaympäristönä onkin kuvattu tarkkaan monissa kirjoissa. (Manninen & Marttila 2006, 10.)

3.2.1 Videotiedostot

Flashillä voidaan videota sisällyttää suoraan osaksi Flash-esitystä tai sitten suoraan Flash Videon avulla (FLV) ilman, että tarvitsee käyttää erillistä Flashin ulkopuolista videon toistoon tarkoitettua ohjelmaa. FLV-tiedostot voivat sisältää sekä kuvan, että äänen. Videotiedostolla on kuvan ja äänen lisäksi metadatana mukana videon korkeus ja leveys, toiston vaatima koodekki, videon kesto ja mahdolliset kohdistuspisteet. Flashissä on mukana työkalut eri videoformaattien muuntamiseksi FLV-muotoon, jolloin voidaan käyttää lähdemateriaalina varsin laajasti erilaisia video-formaatteja. Muunnoksen jälkeen ne ovat formaatissa, joka vastaa käyttötarkoitusta ja on yhteensopiva Flash Playerin kanssa. Flash Player tukee useampaa erilaista koodekkia ja käyttötarkoitukseen paras koodekki valitaankin jo videon muunnoksen kohdalla. Ääniraidan kohdalla voidaan valita formaatin lisäksi myös haluttu äänen laatu. Lisäksi samalla voidaan määritellä videon kontrollien ulkoasu, jos ulkoasu jätetään määrittelemättä, ei videolla ole oletuksena kontrolleja. (Manninen & Marttila 2006, 238-241.)

Tämän lisäksi on olemassa erillinen FLVPlayback-komponentti, jolla voidaan näyttää erillinen Flash-ohjelman ulkopuolinen video. Komponentille voidaan määritellä halutut

ominaisuudet. Ominaisuuksiksi voidaan määritellä esimerkiksi contentPath, joka määrittelee videon lähteen. AutoPlay, joka kertoo sen, että aloitetaanko videon toisto automaattisesti. AutoRewind eli se, että kelataanko video alkuun, kun toisto loppuu. AutoSize, joka määrittelee komponentin automaattisen skaalauksen videota vastaavaksi. CuePoints, joka kertoo videon kohdistuspisteet. MaintainAspectRatio, joka määrittelee sen, että skaalataanko video komponentin kokoon. TotalTime, joka kertoo videon kokonaiskeston. Ilman totalTimen määrittelyä progressbar-toiminnot eivät ole käytössä. Lisäksi voidaan määritellä volume, joka määrittelee videon äänen tason. (Manninen & Marttila 2006, 242-243.)

FLVPlayback-komponentin ulkoasua voidaan muokata ActionScriptillä sekä käyttämällä CustomUI-komponentteja. FLVPlaybackille on lisäksi oma luokkansa, jolla on laaja valikoima metodeita, tapahtumia ja ominaisuuksia, joita muokkaamalla voidaan ulkoasu säätää halutunlaiseksi. Tällöin voidaan videon kontrolleista ottaa käyttöön vain ne joille on todellisuudessa tarvetta ja muokata käyttöliittymä yksinkertaisemmaksi. FLVPlayback-komponentti mahdollistaa myös sen, että videomateriaali valitaan käytössä olevan siirtonopeuden mukaan. Siirtonopeuden mittauksen yhteydessä lähetetään palvelimelta dataa käyttäjän koneelle ja mitataan siirtoon kulunut aika. Tämän perusteella siirtonopeus voidaan laskea. Tämän ominaisuuden hyödyntäminen vaatii eri laatuisia videotiedostoja, joista sopiva valitaan siirtonopeuden perusteella. (Manninen & Marttila 2006, 244-245, 249.)

Koska kontrollien käyttäminen videon näytön yhteydessä lisää tiedoston kokoa, kaikissa tilanteissa ei ole toivottavaa käyttää erillistä komponenttia. Komponenttia toiminnot voidaan tehdä myös ActionScriptillä. ActionScriptistä löytyy erilliset luokat nimiltään Video, NetConnection ja NetStream, joilla videoita voidaan käsitellä sovelluksessa. (Manninen & Marttila 2006, 245-247.)

NetConnection-luokka mahdollistaa yhteyden muodostamisen palvelimella olevaan FLV-muotoiseen videoon. NetConnection-luokassa on connect-metodi, jolla voidaan ottaa yhteys joko paikalliseen videotiedostoon tai HTTP-osoitteen avulla valmiiseen videotiedostoon, joka sijaitsee toisessa sijainnissa. RTMP-osoitteella (Real Time Messaging Protocol) voidaan ottaa yhteys suoratoistettavaan tiedostoon. Merkittävänä erona

RTMP-yhteyden käytössä on se, että videotiedostoa ei tällöin tallenneta paikallisesti kovalevylle lainkaan, vaan se ladataan ainoastaan muistiin. Tiedostoa ei voida tällöin tallentaa omaan käyttöön. NetSream-luokalla siirretään NetConnection-luokan avulla avatun videon data Video-luokan instanssiin. NetConnection-luokalla on onStatus-tapahtuma, jonka avulla saadaan tieto yhteyden muodostamisen tilasta. (Manninen & Marttila 2006, 246-247.)

3.2.2 Äänitiedostot

Flashissä on mahdollista luonnollisesti siirtää ja toistaa pelkkää ääntä. Äänitiedostot voidaan sijoittaa suoraan sovelluksen aikajanalle tai käyttää ActionScriptin Sound-luokan avulla. Äänitiedostot on jaoteltu event- ja stream-tyyppisiin tiedostoihin. Event-tyyppiset tiedostot pitää ladata kokonaan ennen kuin niiden toisto voidaan aloittaa ja stream-tyyppisten tiedostojen toisto voidaan aloittaa jo ennen kuin ne on kokonaan ladattu. (Manninen & Marttila 2006, 232.)

Kuten video-tiedostojen kohdalla, niin myös äänitiedostot voidaan tuoda Flash-sovellukseen. Tiedostoja tuotaessa voidaan määritellä niiden ominaisuudet, joita voidaan muokata myöhemmin. Äänitiedoston pakkaukseen käytettävä formaatti valitaan tässä vaiheessa, kuten myös pakkauksen laatu. Parempi laatu johtaa suurempaan tiedostokokoon ja huonolaatuista ääntä ei tietenkään saa muutettua oikeasti parempilaatuiseksi. Tällä tavoin Flash-sovellukseen tuodut äänitiedostot voidaan ottaa käyttöön vetämällä se sovellukseen haluttuun kohtaan stagelle. Kun näin tehdään, niin äänitiedosto tulee samalla näkyviin sovelluksen aikajanalle. Äänille voidaan määritellä myös erilaisia efektejä, kuten se mistä kaiuttimesta ääni kuuluu tai äänen häivyttäminen. Jos Flash-sovelluksen grafiikanpiirto ei pysy äänen perässä, tiputetaan videosta kehyksiä (frames) pois tarpeellinen määrä. Flash-sovellukseen tuotuja äänitiedostoja voidaan käyttää myös ActionScript-ohjelmoinnin kautta jolloin ääni tiedostolle pitää syöttää Linkage Properties Indentifier, jolloin se voidaan napata kiinni ActionScriptin Sound-luokan avulla. (Manninen & Marttila 2006, 232-234.)

Flash-sovellukseen voidaan ladata ActionScriptillä myös sovelluksen ulkopuolisia äänitiedostoja. Tämä tapahtuu Sound-luokan loadSound-metodilla. Metodille määritellään tiedoston URL-osoite ja se, että käytetäänkö äänitiedostoa ladatessa suoratoistotekniik-

kaa vai ei. Jos suoratoistotekniikkaa ei käytetä, niin äänitiedosto tallentuu paikalliselle kovalevylle kokonaisuudessaan ennen toiston aloittamista. Äänitiedoston aloituskohta voidaan määritellä start-metodilla, jos äänitiedostoa ei haluta soittaa alusta asti. Ulkopuolisia äänitiedostoja voidaan ladata suoraan ActionScriptin kautta myös käyttämällä MediaPlayback-komponenttia, jolloin komponentti pitää sijoittaa valmiiksi sovelluksen stagelle ja sen jälkeen määritellä komponentille haluttu ulkopuolinen äänitiedosto. (Manninen & Marttila 2006, 233-234.)

Sound-luokassa voidaan käsitellä sekä sovellukseen valmiiksi tuotuja äänitiedostoja sekä ulkoisia äänitiedostoja. Ääniä käsitellään luomalla sound-luokasta instanssi konstruktorilla. Konstruktorille määritetään MovieClip-instanssi, johon ääniobjekti kohdistetaan. AttachSound- tai loadSound-metodilla otetaan äänitiedosto käyttöön. AttachSound-metodi ottaa käyttöön Flash-sovellukseen ennalta tuodun äänitiedoston ja loadSound lataa ulkoisen tiedoston. Jos loadSound-metodille määritetään mahdollisuus suoratoistotekniikan käyttöön, voidaan äänitiedoston toisto aloittaa automaattisesti. Muussa tapauksessa, tai attachSound-metodin tapauksessa, äänitiedoston toisto pitää aloittaa erikseen start-metodilla. Sound-luokalla on myös getBytesLoaded- ja getBytesTotal-metodit, joiden avulla voidaan laskea ladatun datan sekä puuttuvan datan määrät. Samaten on mahdollista setPan- ja SetTransform-metodeilla säätää äänen kuulumista eri kaiuttimista. Luonnollisesti myös äänen volyymiä on mahdollista säädellä setVolume-metodilla. Näille kolmelle metodille on olemassa myös vastaavat get-metodit. (Manninen & Marttila 2006, 234-236.)

3.2.3 Kuvatiedostot

Flashin perinteisesti hyvä ominaisuus on hyvin laaja kirjo eri kuvaformaatteja, joita sen avulla voidaan käyttää. Valmiit kuvat voidaan tuoda suoraan Flash-sovellukseen. Kuvat voidaan tuoda suoraan sovelluksen aikajanalle tai tuoda ne sovelluksen kirjastoon, josta ne voidaan ottaa käyttöön tarvittaessa. Flash tukee sekä vektorigrafiikkaa, että bittikarttamuotoisia kuvatiedostoja. Erilaisia Flashin tukemia formaatteja onkin niin paljon, että on vaikea kuvitella tilannetta, jossa olisi välttämätöntä käyttää sellaista kuvaformaattia, jota Flash ei tue. (Manninen & Marttila 2006, 259.)

Kun kuva tuodaan Flash-sovelluksen stagelle tai kirjastoon se tallentuu kirjastoon automaattisesti. Flashissä on mahdollista säätää kuvaan liittyviä pehmennys- ja pakkausasetuksia. Kuvan pehmennyksen yhteydessä Flash pehmentää kuvaa anti-aliasing-tekniikalla, joka poistaa tekstin tai kuvien reunojen sahalaitaisuutta. Pakkauksen osalta voidaan kuva pakata JPEG-formaatin mukaisesti tai jättää se pakkamatta kokonaan jolloin lopputuloksena on joko GIF- tai PNG-tiedosto. JPEG:n kohdalla valitaan haluttu pakkauksen taso. Suurempi pakkauksen taso pienentää tiedostokokoa, mutta heikentää kuvan laatua. Import-toiminnolla voidaan korvata vanha kuvatiedosto uudella ilman, että sovellukseen tarvitsee tehdä muita muutoksia. Kuvia voidaan Flashissä muokata esimerkiksi muuttamalla niiden kokoa, kallistamalla niitä ja kiertämällä niitä Free Transform työkalulla. Kuvista voidaan myös irrottaa osia Break Apart-toiminnolla ja bittikarttapohjaiset kuvat voidaan muuntaa vektorigrafiikaksi määritellen sille tietyt parametrit. (Manninen & Marttila 2006, 259.)

ActionScriptillä voidaan tuoda Flash-sovellukseen sovelluksen ulkopuolisia kuvia käyttämällä loadMovie- tai loadMovieNum-metodeja. Näiden metodien lisäksi kuvat voidaan ladata MovieClip-luokan metodeja tai MovieClipLoader-luokan metodeja. Valittava luokka ja metodi kannattaa valita kuvan käyttötarkoituksen perusteella. (Manninen & Marttila 2006, 260.)

ActionScriptillä on mahdollista jälkikäsitellä bittikarttamuotoisia kuvia monella eri tavalla. Kuvasta muun muassa voidaan zoomata haluttu alue (Manninen & Marttila 2006, 265.), kuvaan voidaan tuottaa kohinaa halutulle alueelle (Manninen & Marttila 2006, 267-268.) tai kuva voidaan vaihtaa toiseen kuvaan pikseli kerrallaan (Manninen & Marttila 2006, 270.). Näiden jälkikäsittelytapojen hyödyntäminen on lähinnä mielikuvituksesta kiinni. Zoomausta voidaan käyttää esimerkiksi verkkokauppojen tuotekuvien kohdalla, kohinalla voidaan tehdä kuvasta sumuisempi ja kuvan vaihtaminen pikseli kerrallaan toiseen on yksi tapa tehdä siirtymä kuvien välillä.

3.2.4 CreateJS:n käyttö HTML5 ja JavaScript-koodin tuottamiseen

CreateJS on apuväline Flash-ammattilaisille, jonka avulla Flash-sisältö voidaan muuntaa HTML5:n ymmärtämään muotoon. CreateJS muuntaa Flash-sisältöä JavaScriptiksi ja näin ollen se ei voi muuntaa sellaista sisältöä, jota ei JavaScriptillä pysty tuottamaan.

CreateJS ei olekaan tarkoitettu esimerkiksi pelien muuntamiseen HTML5:llä toimiviksi. CreateJS tukee Flashin ydinominaisuuksia kuten bittikartta- ja vektorimuotoisen grafiikan tuottamisen, äänien soittamisen ja aikajanan skriptauksen. CreateJS hoitaa kätevästi JavaScript-kirjastojen, kuvien ja äänien kopioinnin HTML5-sivulle. CreateJS koostuu neljästä eri kirjastosta. TweenJS hoitaa aikajanalla tapahtuvat animaatiot, EaselJS hiiren käytön ja HTML5:n canvas-elementin filterit, AudioJS äänimaailman ja PreloadJS sivun sisällön lataamisen. Osalle kirjastoista löytyy myös vaihtoehtoisia versioita, joita voidaan käyttää pääkirjastojen sijaan. (Skinner)

CreateJS muuntaa Flash-kirjaston JavaScript-tiedostoksi, joka toimii uudelleen käytettävänä luokkana. Tiedoston nimiavaruus voidaan määritellä erikseen. CreateJS yrittää muuntaa MovieClip, Graphic ja Button sisällön. Vain sellaiset elementit, jotka ovat käytössä tai jotka on erikseen asetettu muunnettaviksi, sisällytetään JavaScript-tiedostoon. Bittikartta-muotoinen grafiikka ja äänet kopioidaan oikeisiin kansioihin automaattisesti. Aikajanalla olevien animaatioiden muuntoon sisältyy rajoituksia ja kaikki uusimmat animaatio-ominaisuudet eivät ole tuettuina. Uuden tyyppiset tweenit (animaatiot, joissa sama kuva liikkuu tai muuntuu ja Flash hoitaa alun ja lopun välissä olevien kuvien tuottamisen) eivät muunnu samassa muodossa kuin Flashissä vaan jokaisesta kuvan muuntuman välivaiheesta muodostuu oma kuvansa, mikä tekee animaatiosta huomattavasti raskaamman. Vanhemman malliset tween-animaatiot joissa kuva ei muutu, vaan ainoastaan siirtyy, sen sijaan muuntuvat sellaisenaan. (Skinner)

CreateJS:n avulla tapahtuva muunnos tukee kaikissa tapauksissa kuvan sijaintiin, kiertoon, kokoon, näkyvyyteen ja vääntelyyn liittyviä ominaisuuksia. Lisäksi se tukee Text:in ja MovieClipin kohdalla varjoihin ja kuvan hehkuun liittyviä ominaisuuksia, joskin hehkun (glow) kohdalla Canvas-elementin suorituskykyongelmien takia sen käyttöä ei suositella. Grafiikan kohdalla muunnokset toimivat pääosin. Ongelmia aiheuttavat esimerkiksi ovaalit ja bittikarttojen väritäytön muodon muuttaminen. Vektorigrafiikan kohdalla suositellaan tween-animaatioita jokaisen erillisen kuvan piirtämisen sijaan. Äänien tapauksessa äänet tuotetaan MP3-muotoisina. Koska kaikki selaimet eivät tue MP3-ääniä niistä pitää erikseen tehdä versiot, jotka ovat eri muodossa ja antaa HTML5:n audio-elementin valita käytettävä versio. CreateJS on siis kätevä työkalu

HTML5 sisällön kehittämisessä Flashillä, mutta sillä on omat rajoituksensa, kuten tämän kaltaisilla muunnostyökaluilla yleensä. (Skinner)

3.3 HTML5:n ja Flashin edut ja haitat

HTML5:n avulla voi kehittää videoita ja ääntä sisältäviä verkkosivuja, jotka toimivat ilman erillisiä selain lisäosia. Flash ja Silverlight ovat tarjonneet samankaltaisen toiminnallisuuden jo vuosi, mutta molemmat vaativat erillisen lisäosan asentamisen. Flashin kohdalla lisäosa ei enää ole saatavilla kaikille laitteille, joista mainitsemisen arvoisia ovat iOS-alustan päällä toimivat mobiililaitteet. HTML5 ei myöskään ole riippuvainen yhdestä laite- tai ohjelmistovalmistajasta. Canvas-elementin avulla on myös mahdollista hyödyntää vektorigrafiikkaa omilla verkkosivuilla ilman selainlisäosien asentelua. Näiden etujen lisäksi HTML5:llä toteutetut sivut, joilla on videoita, ääniä tai canvas-elementin avulla toteutettuja graafisia esityksiä säästävät kannettavien laitteiden akkutehoa, koska ne eivät vaadi erillisen ohjelman käynnistämistä esityksen näyttämistä varten. (DPCI.)

Flashin suurin etu on se, että se ei ole selainriippuvainen. Koska se on erikseen asennettava lisäosa, se toimii kaikilla selaimilla samalla tavalla ja lopputulos on yhdenmukainen. Flashin avulla on helppo lisätä verkkosivulle videoita ja ääntä. Se, että Flash vaatii erikseen asennettavan lisäosan, on myös sen suurin heikkous. Jos sivulla käyvällä ihmisellä ei ole lisäosaa asennettuna, ei sivu toimi. Lisäksi Flash-lisäosaa ei ole saatavissa kaikille mobiililaitteille. Applen laitteiden luomat ongelmat on mahdollista kiertää tekemällä niille omat versiot sivusta, mutta siinä on oma pieni vaivansa. (Ryland Designs)

Iso etu Flashissä on myös se, että Flash tukee hyvin monia kuva-, audio- ja videoformaatteja. Tämä mahdollistaa hyvinkin kirjavan lähdemateriaalin käyttämisen ilman, että tarvitsee erillisillä työkaluilla muuntaa tiedostoja eri muotoon. Flashillä voidaan hoitaa tiedostojen formaattimuunnokset siinä vaiheessa, kun ne tuodaan Flash-sovelluksen kirjastoon. (Manninen & Marttila 2006, 232, 239, 259.)

HTML5:ssä taas esimerkiksi eri videotiedostoformaattien tuki riippuu käytettävästä selaimesta ja yksi tietty formaatti ei välttämättä toimi kaikissa selaimissa. HTML5:ssä ongelman saa kierrettyä määrittelemällä videolle useamman eri lähteen, joista selain

käyttää ensimmäistä joka toimii. Tuki eri formaateille mobiililaitteissa on vieläkin monimutkaisempi tapaus ja video-materiaalin lähteen määrittely tehdään niissä hieman vaihtelevasti. Jos haluaa varmistua sivujen toimivuudesta kaikissa tapauksissa, onkin välttämätöntä määritellä videoille jonkinlainen varasisältö, jota käytetään, jos videon näyttäminen ei onnistu lainkaan esimerkiksi siksi, että käyttäjä käyttää vanhaa versiota Internet Explorerista. (Korpela 2014, 370-372.)

4 HTML5:n multimediavertailu eri selaimissa

Lähtökohtana on tehdä HTML5:llä yksinkertainen verkkosivu video-, audio- ja canvas-elementtien testausta varten. Testauksessa keskitytään eri elementtien toimintaan PC-koneiden verkkoselaimilla eli Microsoftin Internet Explorerilla, Mozilla Firefoxilla, Google Chromella ja Opera Softwaren Operalla. Elementtien toiminnan testauksen lisäksi keskitytään ulkoasun yhdenmukaisuuteen eli tarkastellaan elementtien toimintaa myös verkkosivujen ulkoasun ja sommittelun kannalta.

Testauksessa tehdään video-elementin osalta versio, jossa käytetään oletuskontrolleja sekä versio, jossa kontrollit tehdään itse. Oletuskontrollien käyttö on oleellista siksi, että jos JavaScript ei jostain syystä toimi ja oletuskontrolleja ei ole lainkaan, niin elementille ei muodostu kontrolleja lainkaan. Videon tapauksessa videota ei tällöin voida käynnistää lainkaan ellei videota ole asetettu käynnistymään itsestään. Audio-elementin kohdalla kontrollien puuttuminen tarkoittaa sitä, että elementtiä ei näy laisikaan. Canvas-elementin kohdalla tehdään yksinkertainen piirtoalusta ja testataan sen toiminta eri selaimilla.

Video- ja audio-elementtien kohdalla pitää ottaa huomioon myös lähdetiedoston formaatti. Videotiedostojen kohdalla kaikki selaimet eivät tue samoja formaatteja eli videolle pitää tarjota useampi eri lähdetiedosto. Myös siihen pitää varautua, että videota ei lähdetiedostoista pystytä syystä tai toisesta näyttämään lainkaan. Tämän toteutus voi olla videotiedostojen osalta hankalahko, käytännössä voidaan yrittää tehdä dia-show, jossa videon sisältö esitetään kuvina ja tekstinä, mutta tämänkään ei ole täysin tyydyttävä ratkaisu. Vähintään pitää kuitenkin kertoa käyttäjälle se, että videotiedostoa ei voitu näyttää. Audion osalta yritetään käyttää mp3-formaattia, jonka periaatteessa pitäisi toimia kaikissa neljässä selaimessa.

4.1 Tavoite

Tarkoituksena on kokeilla HTML5:n multimedia-elementtien toimintaa eri PC-selaimissa. Tiukan määrittelyn pohjalta canvas ei ole media-elementti, koska se ei käytä HTMLMediaElementin ominaisuuksia, mutta canvas on kuitenkin toiminnallisuudeltaan selvästi multimediaelementti siinä missä videokin, koska canvas on tarkoitettu

graafisten esitysten näyttämiseen ja voi sisältää animaatioita, kuvia tai tekstiä. Lisäksi tietoperustassa tapahtuva vertailu HTML5:n multimediaelementtien ja Flashin välillä jäisi varsin puutteelliseksi, jos Canvasta ei vertailuun otettaisi mukaan.

Tavoitteeseen kuuluu myös selvittää se, että toimivatko elementit kaikissa neljässä selaimessa samalla tavalla ja se, että mitä seikkoja pitää ottaa huomioon eri selaimissa. Produktina syntyy testisivu ja raportti sen toimivuudesta. Pääpaino on testisivun hyödyntäminen elementtien toiminnan ja ulkoasun testaamisessa ja testisivu itsessään ei tule olemaan kovinkaan monimutkainen.

Flash-sisällön tekeminen ja sen muuntaminen HTML5:n tukemaan muotoon EaselJS:llä olisi mielenkiintoinen lisä, mutta se jää rajauksen ulkopuolelle, koska Flashin käytön opettelu pitäisi tehdä erikseen ja se ei mahdu aikatauluun. Sisältö joka produktin verkkosivu-osuudessa on, olisi kuitenkin helposti toteutettavissa Flashilla ja tietääkseni se ei sisällä mitään sellaista, joka ei EaselJS:llä kääntyisi. EaselJS:n rajoitteina ovat lähinnä selaimien ja HTML5:n asettamat seikat, eivät itsessään EaselJS:n ominaisuudet.

4.2 Ongelmat ja kehittämistehtävä

Ongelmat liittyvät sekä rajaukseen, toiminnallisuuteen, sekä ulkoasuun. Koska video- ja audio-elementit ovat käytännössä vain rajapintoja, on niiden toiminnallisuuden toteutus selainten kehittäjien käsissä. Sama pätee myös niiden ulkoasuun. Ulkoasun eroavaisuudet esiintyvät lähinnä elementtien valmiiden kontrollien kohdalla, jotka eroavat selaimesta toiseen varsin reilusti. Video- ja audio-elementtien kontrollien eroavaisuudet voidaan kiertää laittamalla valmiit kontrollit piiloon ja tekemällä kontrollit itse JavaScriptillä. Valmiit kontrollit on myös luonnollisesti rajattu yleisimpiin ja jos on tarvetta tarkemmille kontrolleille, ne pitää tehdä itse. JavaScriptillä itse tehtyjen kontrollien ongelmana on luonnollisesti se, että selaimen asetuksissa saattaa olla JavaScriptin käyttö kielletty. Tällöin ei hirveästi ole muuta mahdollisuutta, kuin käyttää selaimen elementille luomia kontrolleja, riippumatta siitä, että miten pahasti ne ulkoasua sotkevat.

Rajauksen osalta ongelmana on se, että kohdeselaimet ovat kaikki PC:llä käytettäviä työpöytäselaimia. Tämä rajaa aihealueen ulkopuolelle osan mobiiliselaimista, jotka toki perustuvat työpöytäselaimiin. Applen tuotteiden osalta rajauksen ulkopuolelle jää sekä

työpöytä-, että mobiilipohjaiset selaimet, joka on valitettava, mutta oli välttämätöntä opinnäytetyön tekemiseksi sen nykyisessä aikataulussa ja laajuudessa.

Canvaksen kohdalla ongelmat ovat lähinnä kyseisen elementin laajuudessa. Canvas kokonaisuutena voisi olla oman opinnäytetyönsä aiheena ja tässä opinnäytetyössä tullaankin käsittelemään vain suhteellisen yksinkertaisia ominaisuuksia. Teoriassa canvaksella voidaan toteuttaa ihan mukiinmeneviä pelejäkin, mutta toistaiseksi niiden valikoima on sangen rajallinen verrattuna esimerkiksi Flashillä tehtyihin selainpeleihin.

4.3 Suunnitelmakuvaus

Visual Studiolla rakennetaan testisivut. Jokainen testattava elementti sijoitetaan omalle sivulleen selkeyden vuoksi. Testattavia elementtejä on kolme. Video, audio ja canvas. Eri elementeistä kustakin valitaan testattavaksi joukko ominaisuuksia, joiden toimivuus testataan eri selaimilla. Lisäksi kiinnitetään erityistä huomiota eri selaimien välillä ilmeneviin ulkoasuseikkoihin. Koko toteutus tehdään suoritettavaksi paikallisesti lukuun ottamatta mediatiedostoja, jotka luonnollisesti ladataan palvelimelta.

Videon osalta testataan elementin omien kontrollien toiminta ja vertaillaan niiden ulkoasua keskenään. Videolle tehdään myös kontrollit itse ja testataan niiden toiminta. Testattavat ominaisuudet ovat lataus, käynnistys, toiston tauotus, äänen tason muuttaminen, toiston nopeuden muuttaminen sekä siirtyminen videossa tietty sekuntimäärä eteenpäin ja taaksepäin ja siirtyminen videossa suoraan valmiiksi määriteltyyn kohtaan. Kontrollien toiminnan lisäksi testataan kolmen eri formaatin toiminta valituilla selaimilla. Formaatit ovat H-264 eli MPEG-4, OGG-formaatti ja WEBM-formaatti. H.264 on formaattina jo vakiinnuttanut asemansa selainpohjaisessa videotoistossa. OGG ja WEBM ovat formaatteina hieman tuntemattomimpia, ilmaiseen käyttöön tarkoitettuja formaatteja. Testivideo muunnetaan kaikkiin kolmeen eri muotoon ja eri versiot tarjotaan video-elementille source-alielementissä, joita voi video-elementin sisällä olla useita, joista selain käyttää lähdetiedostoa, joka toimii kyseisessä selaimessa.

Audion osalta tarkastellaan elementin omien kontrollien toiminta, sekä vertaillaan niiden ulkoasua keskenään. Myös audio-elementille tehdään kontrollit myös itse. Itse tehdään kontrollit lataukselle, käynnistykselle, toiston tauotukselle, äänen tason muuttami-

selle, toiston nopeuden muuttamiselle, siirtymiselle audio-tiedostossa eteen- ja taaksepäin sekä hyppäämiselle suoraan tiettyyn kohtaan. Formaattina on tarkoitus käyttää MP3:a, joka toivottavasti toimii kaikissa kohdeselaimissa. Jos MP3 ei toimi kaikissa kohdeselaimissa, etsitään ääniformaatti jolla ongelma voidaan sivuuttaa.

Canvas-elementtiä testataan tekemällä sille JavaScriptillä yksinkertaiset piirtotyökalut ja kokeilemalla niiden toiminta. Canvas-elementinkin kohdalla tarkastellaan mahdollisia selainten välisiä eroja ulkoasun suhteen. Canvakselle on aina välttämätöntä valita piirtokonteksti ja tässä tapauksessa kontekstiksi valitaan 2d sen yksinkertaisemman luonteen vuoksi verrattuna WebGL:ään verrattuna. 2d-kontekstin puitteissa tehdään mahdolliseksi piirtää alustalle vapaasti hiirelle, poistaa jo piirrettyä sisältöä, vaihtaa piirtoväriä, kopioida piirretty sisältö toiselle canvas-elementille ja tallentaa piirretty kuva oman koneen kovalevylle.

4.4 Toteutus

Produktin rakentaminen tapahtui Visual Studiolla. Lähtökohdaksi otettiin tyhjä webbisivuprojekti. Ensin luotiin kullekin testattavalle elementille oma testisivu eli luotiin kolme tyhjää sivua, video.html, audio.html ja canvas.html. Kun sivut oli luotu, niille luotiin käsin kirjoittamalla tarvittavat elementit.

Video.html sivulle sijoitettiin videoelementti, jolla oli valmiiksi määritelty koko ja oletuksena elementin omat kontrollit käytössä. Tämän jälkeen lähdetiedosto, joka oli H.264-tiedosto, muunnettiin ogg- ja webm-formaatteihin, että saatiin video-elementille useampi mahdollinen lähdetiedosto, koska jo tässä vaiheessa oli tiedossa, että kaikki selaimet eivät kaikkia kolmea formaattia tue. Konvertoinnin jälkeen tiedostot sijoitettiin verkkosivun alikansioon ja lisättiin Visual Studion projektiin. Kun tiedostot oli lisätty projektiin, niin video-elementille lisättiin kolme source-alielementtiä, joissa määritettiin lähdetiedostojen nimet ja sijainnit sekä niiden tiedostomuodot. Sivulle lisättiin tämän jälkeen itse tehdyt kontrollit video-elementille, jotka jäävät oletuksena piiloon. JavaScriptillä laitettiin tämän jälkeen valmiit itse tehdyt kontrollit näkyviin. JavaScriptin alkuun lisättiin rivi, jolla elementin omat kontrollit voidaan tarvittaessa laittaa pois päältä. Tällä tavalla varmistetaan, että video-elementin omat kontrollit jäävät näkyviin, jos JavaScriptin käyttö on selaimen asetuksissa estetty. Alla on produktissa käytetty yksinkertainen

video-elementti. Koska elementille on määritetty valmiiksi sekä korkeus, että leveys, videotiedosto skaalautuu elementin mittojen mukaiseksi.

```
<video id="video1" controls="" preload="auto" height="480" width="854">
      <source id="StarCitizenMP4" src="Videos/SC.mp4" type="video/mp4">
      <source id="StarCitizenOGG" src="Videos/SC.ogv" type="video/ogg">
      <source id="StarCitizenWEBM" src="Videos/SC.webmhd.webm"
type="video/webm">
      <p>Selaimesi ei tue the HTML5:n Video-elementtiä.</p>
</video>
```

Audio.html sivulle sijoitettiin luonnollisesti audio-elementti, jonka kontrollit määritettiin näkyviksi. Verkkosivun alikansioon sijoitettiin mp3-muotoinen audiotiedosto ja tiedosto lisättiin projektiin. Audio-elementin lähteeksi määritettiin yksittäinen audiotiedosto. Tässä poikettiin video-elementin useamman lähteen mallista, koska oletuksena oli, että mp3 toimii kaikissa selaimissa. Sivulle lisättiin itse tehdyt kontrollit audio-elementille. JavaScriptissä laitettiin audio-elementin omat kontrollit piiloon ja laitettiin itse tehdyt kontrollit näkyviin. Kuten video-elementissä, tällä varmistetaan, että audio-elementin omat kontrollit ovat näkyvissä, jos JavaScriptin toiminta on selaimessa estetty. Alla on yksinkertainen audio-elementti, joka löytyy produktista.

```
<audio id="dangerZone" controls="">
    <source id="audioFile" src="Audios/DZ.mp3" type="audio/mp3" />
</audio>
```

Canvas.html sivulle lisättiin canvas-elementti. Elementille määriteltiin valmiiksi koko. Tämän jälkeen yritettiin JavaScriptillä luoda yksinkertainen piirtotyökalu hiirtä varten, mutta jostain syystä sitä ei saatu toimimaan lainkaan tavalla. Tämän ongelman syyksi paljastui aikanaan se, että esimerkeissä, joita yritettiin hyödyntää piirtotyökalun luomiseen, oletettiin käytössä olevan myös JQuery. Koska tässä produktissa ei ollut tarkoitus käyttää JQueryä lainkaan, ei piirtotyökaluja ollut tarkoituksenmukaista tehdä sen avulla. Pienen etsimisen jälkeen löytyi toinen tapa tehdä toimivat piirtotyökalut. Tämän menetelmän ongelmana oli se, että se pohjautui absoluuttisiin sijainteihin ja ei täten ole sellaisenaan kovinkaan hyödyllinen oikean verkkosivun tarpeisiin, mutta koska samalla sivulla ei ollut muita näkyviä elementtejä lainkaan, se ei ollut ongelma. Piirtotyökalu toimii ainoastaan hiiren ollessa canvas-elementin päällä, kuten on tarkoituksenmukaista. Piir-

toalustan lisäksi lisättiin mahdollisuus valita muutamasta väristä se, jota halutaan käyttää piirtämiseen sekä piirtojäljen kumittaminen, käytännössä piirtämällä alustan päälle valkoisella värillä. Piirto-ominaisuuksien lisäksi lisättiin JavaScriptillä mahdollisuus tyhjentää piirtoalusta kokonaan, kopioida sen sisältö toiseen piirtoalustaan sekä mahdollisuus tallentaa kuva. Alla on produktissa käytetty yksinkertainen canvas-elementti.

```
<canvas id="piirtoAlusta1" width="400" height="400"
style="position:absolute;top:10%;left:10%;border:2px solid;">
</canvas>
```

4.5 Toiminnallisuuden ja ulkoasun testaus

Produktissa on kolme sivua, yksi jokaisella testattavalle elementille. Elementit on sijoitettu eri sivuille, että elementtien erot eri selaimissa saadaan selvemmin esille. Elementtien sijoitus omille sivuilleen tekee tässä tapauksessa myös JavaScriptistä selkeämmin luettavaa, koska erilaisiin elementteihin liittyvä JavaScript on samaten eritelty eri sivuille.

Video-elementistä testattiin ensin perustoiminnallisuus selaimen omien oletuskontrollien avulla sekä tarkistettiin mitä kolmesta testattavasta formaatista selain tukee. Formaatit olivat H.264, Ogg ja WebM. Tämän jälkeen testattiin itse tehtyjen kontrollien toiminta. Itse tehdyt kontrollit tarjoavat toiminnallisuutta, jota ei selaimen omissa kontrolleissa ole.

Audio-elementin kohdalla testattiin samoin ensin selaimen omilla oletuskontrolleilla. Audio-formaattina käytetään vain mp3:a, joka toimii kaikissa selaimissa. Kun selaimen omat kontrollit oli testattu, niin testattiin itse tehtyjen, laajempien, kontrollien toiminta.

Canvas-elementin toiminta testattiin hiirellä piirtämisen osalta, jonka lisäksi testattiin värinvaihto, piirretyn kuvan pyyhkiminen, piirretyn kuvan kopiointi sekä piirretyn kuvan tallennus. Oletuksena oli, että canvaksen toiminnassa ei näiltä osin ole selaimissa eroja.

4.5.1 Video-elementin perustoiminnallisuus

Video-elementin sivu on nimeltään video.html ja se sisältää video-elementin sekä sen kontrollit ja itse tehdyt kontrollit. Alla on vertailu eri selainten omien kontrollien ulkoasusta ja toiminnasta.

Kuva1. Video-elementti IE:ssä oletuskontrolleilla

- Selaimen versionumero, IE11 11.0.9600.17358
- Play, toimii
- Pause, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Full screen, toimii
- Siirtyminen suoraan tiettyyn kohtaan, toimii
- Play pausen jälkeen, toimii
- H.264 formaatti, toimii
- Ogg formaatti, ei toimi
- WebM formaatti, ei toimi

Kaikki selaimen omat kontrollit video-elementille siis toimivat. Internet Explorer näyttää tukevan video-formaateista ainoastaan H.264:ää. Kaiken kaikkiaan IE:n tuki video-

elementin perustoiminnallisuudelle on riittävä, joskin ainoastaan H.264:än tukeminen voi olla pieni ongelma siinä tapauksessa, että selaimesta tulee uusi versio saataville, jossa H.264 tuki on jostain syystä toimimaton. Todennäköisyys on toki pieni, mutta toteutuessaan se estää video-elementin toiminnan käytännössä kokonaan. IE:n omat videokontrollit ovat vertailussa olevista selaimista selvästi suurimmat. Pieni miinus kontrolleissa on se, että sekä äänen vaimennus, että äänen voimakkuuden säätö ovat saman kontrollit alla, mikä voi aiheuttaa pienimuotoista vaivaa käyttäjälle, joka on tottunut siihen, että video-elementissä nämä kaksi toimintoa on erotettu toisistaan.

Kuva2. Video-elementti Firefoxissa oletuskontrolleilla

- Selaimen versionumero, 33.0.3
- Play, toimii
- Pause, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Full screen, toimii
- Siirtyminen suoraan tiettyyn kohtaan, toimii
- Play Pausen jälkeen, toimii
- H.264 formaatti, toimii
- Ogg formaatti, toimii
- WebM formaatti, ei toimi

Firefoxin omat oletuskontrollit toimivat kuten on tarkoituskin. Video-formaateista on tuettuna H.264 ja Ogg. Tämän ansiosta Firefox on toimintavarmempi video-elementin suhteen, koska yksittäinen ongelma lähdetiedoston formaattituen osalta ei estä video-elementtiä toimimasta kokonaan. Selaimen omat video-elementin kontrollit ovat huomattavasti IE:tä sirommat.

Kuva 3. Video-elementti Chromessa oletuskontrolleilla

- Selaimen versionumero, 38.0.2125.111 m
- Play, toimii
- Pause, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Full screen, toimii
- Siirtyminen suoraan tiettyyn kohtaan, ei toimi
- Play pausen jälkeen, toimii
- H.264 formaatti, toimii
- Ogg formaatti, ei toimi
- WebM formaatti, toimii

Chromessakin toimivat selaimen omat oletuskontrollit pääosin hyvin, ainoastaan suoraan tiettyyn kohtaan hyppääminen muodostaa ongelman. Selain tukee H.264-formaattia ja WebM-formaattia. Kuten Firefoxin kohdalla, myös Chromen tapauksessa tämä tarkoittaa sitä, että Chrome saavuttaa paremman toimintavarmuuden kuin IE. Selaimen omien kontrollit ulkoasu poikkeaa IE:stä ja Firefoxista. Kontrollit ovat kuitenkin Chromessakin hyvin selkeät.

Kuva 4. Video-elementti Operassa oletuskontrolleilla

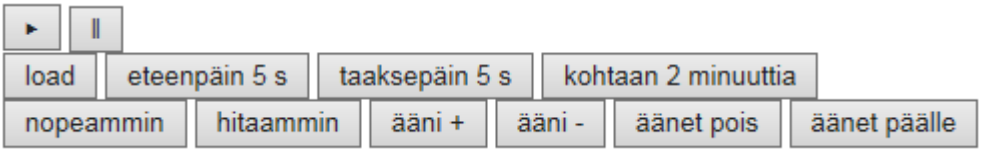
- Selaimen versionumero, 25.0.1614.68
- Play, toimii
- Pause, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Full screen, toimii
- Siirtyminen suoraan tiettyyn kohtaan, ei toimi
- Play pausen jälkeen, toimii
- H.264 formaatti, toimii
- Ogg formaatti, ei toimi
- WebM formaatti, toimii

Kuten jo tässä vaiheessa olettaa saattaa, myös Operan omat video-kontrollit toimivat pääsosin moitteetta. Jostain syystä myös Operassa siirtyminen suoraan tiettyyn kohtaan ei toimi kunnolla. Formaateista on tuettuna H.264 ja WebM, joka on etu IE:hen verrattuna. Oletuskontrollit ovat täsmälleen samannäköiset kuin Chromessa, mikä on etu sellaisten käyttäjien kohdalla, jotka ovat Chromeen ehtineet jo tottua.

Näiden neljän selaimen kohdalla ei siis synny suuria eroja selainten omien kontrollien toiminnan osalta. Videotiedostojen formaattituki vaihtelee selaimien kesken ja IE on selaimista ainoa, joka ei tue Ogg tai WebM formaattia lainkaan. H.264-formaatti on kaikkien selaimien tukema, mikä on positiivinen yllätys, koska tietokehyksessä käytetty tuore lähde kertoi, että tukea Operassa sille ei ollut.

4.5.2 Video-elementin laajempi toiminnallisuus

Laajempaa testausta varten video-elementtiin lisättiin itse tehdyt kontrollit, jotka sisältävät saman toiminnallisuuden kuin selaimen omat kontrollit sekä lisäksi videon toistonopeuden säädön, siirtymisen- eteen ja taaksepäin tietyn sekuntimäärän verran ja hyppäämisen suoraan tiettyyn kohtaan. Tässä testissä selainten omat kontrollit on piilotettu. Kontrollien lisäksi testataan tekstityksen toiminta.

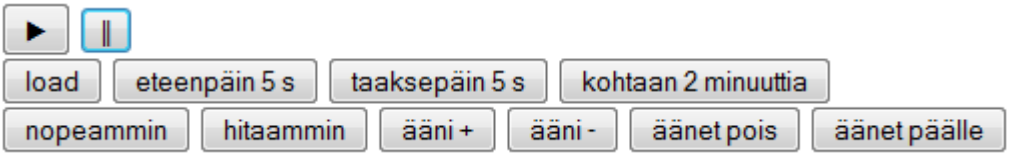


Kuva 5. Video-elementin itse tehdyt kontrollit IE:ssä

- Selaimen versionumero, IE11 11.0.9600.17358
- Play, toimii
- Pause, toimii
- Load, toimii
- Eteenpäin 5 s, toimii
- Taaksepäin 5 s, toimii
- Kohtaan 2 minuuttia, toimii
- Nopeampi kuvan toisto, toimii
- Hitaampi kuvan toisto, toimii

- Äänen taso ylös, toimii
- Äänen taso alas, toimii
- Äänet pois, toimii
- Äänet päälle, toimii

IE:ssä kaikki itse tehdyt kontrollit toimivat moitteetta ja ilman viiveitä. Kuvan toiston hidastus alle alkuperäisen tason toimii pausen tavoin. Kuvaa ei siis voi katsoa hidastettuna. Kontrollien väliin ei jää pystysuunnassa oletuksena tilaa ja play- ja pause-napit ovat jostain syystä hieman eri tasolla.

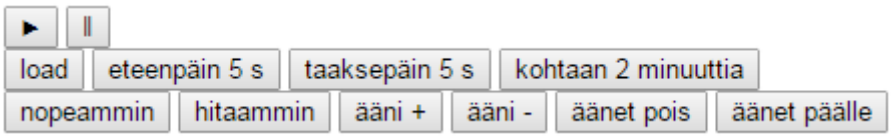


Kuva 6. Video-elementin itse tehdyt kontrollit Firefoxissa

- Selaimen versionumero, 33.0.3
- Play, toimii
- Pause, toimii
- Load, toimii
- Eteenpäin 5 s, toimii
- Taaksepäin 5 s, toimii
- Kohtaan 2 minuuttia, toimii
- Nopeampi kuvan toisto, toimii
- Hitaampi kuvan toisto, toimii
- Äänen taso ylös, toimii
- Äänen taso alas, toimii
- Äänet pois, toimii
- Äänet päälle, toimii

Firefoxissa kaikki kontrollit toimivat. Kuten IE:ssä, myös Firefoxissa kuvan hidastus alle alkuperäisen tason toimii pausen tavoin, eikä kuvaa voi katsoa hidastettuna. Fonttien ulkoasu poikkeaa IE:stä, kuten myös play- ja pause-symbolit. Play-symboli on sen verran kookas, että se muuttaa play-napin suuremmaksi kuin pause-napin. Kontrollien

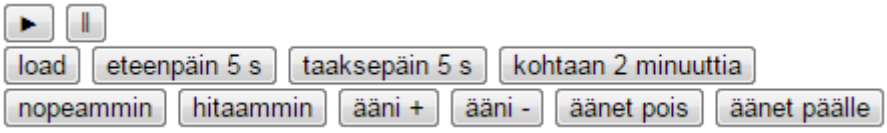
väliin jää oletuksena pystysuunnassa hieman tilaa, joka tekee niistä selkeämmän näköiset.



Kuva 7. Video-elementin itse tehdyt kontrollit Chromessa

- Selaimen versionumero, 38.0.2125.111 m
- Play, toimii
- Pause, toimii
- Load, toimii
- Eteenpäin 5 s, toimii
- Taaksepäin 5 s, toimii
- Kohtaan 2 minuuttia, ei toimi
- Nopeampi kuvan toisto, toimii
- Hitaampi kuvan toisto, toimii
- Äänen taso ylös, toimii
- Äänen taso alas, toimii
- Äänet pois, toimii
- Äänet päälle, toimii

Chromessa kontrollit toimivat päällisin puolin hyvin. Kuitenkin suoraan kahden minuutin kohdalle siirtyminen saa toiston katkeamaan ja videon katselu sen jälkeen vaatii sen uudelleen latauksen, joka luonnollisesti siirtää videon takaisin alkuun. Chromessakin kuvan toiston nopeuden alennus alle aloitustason toimii pausen tavoin ja kuvaa ei voi katsella hidastettuna. Kontrollien väliin ei oletuksena jää pystysuunnassa tilaa.



Kuva 8. Video-elementin itse tehdyt kontrollit Operassa

- Selaimen versionumero, 25.0.1614.68

- Play, toimii
- Pause, toimii
- Load, toimii
- Eteenpäin 5 s, toimii
- Taaksepäin 5 s, ei toimii
- Kohtaan 2 minuuttia, ei toimi
- Nopeampi kuvan toisto, toimii
- Hitaampi kuvan toisto, toimii
- Äänen taso ylös, toimii
- Äänen taso alas, toimii
- Äänet pois, toimii
- Äänet päälle, toimii

Operassa kontrollit toimivat kutakuinkin samalla tavoin kuin Chromessa eli tiettyyn kohtaan siirtyminen suoraan aiheuttaa videon toiston keskeytymisen ja video pitää ladata uudestaan. Lisäksi videolla ei voi siirtyä taaksepäin tiettyä aikamäärää enää sen jälkeen, kun on siirrytty eteenpäin. Kontrollien väliin jää pystysuunnassa mukavasti tilaa.

Kontrollien ulkoasussa on siis eroja selainten välillä. Kontrollien pystysuuntainen väli vaihtelee ja Firefoxissa ja IE:ssä myös samalla rivillä olevien kontrollien koko tai pystysuuntainen sijainti. IE:ssä ja Firefoxissa kaikki kontrollit toimivat, mutta Chromessa ja Operassa eivät. Pitkällisen selvittelyn jälkeen selvisi, että ongelman syy oli lopulta yksinkertainen. Ongelma aiheutui videotiedoston lataamisesta paikalliselta kovalevyltä ja ongelma poistui, kun lähteenä oli videontiedoston url-osoite. Oma osaamiseni ei riitä selvittämään syytä siihen, että miksi ongelma esiintyy nimenomaan paikallisen sisällön toistamisessa, mutta se on onneksi helppo kiertää. Ongelma voi tietenkin johtua käytetystä Visual Studio 2013 kehitysympäristöstä, mutta riippumatta ongelman perimmäisestä syystä se muodostaa kehittäjän kannalta mahdollisen ongelman, ellei siitä ole tietoinen.

4.5.3 Audio-elementin perustoiminnallisuus

Audio-elementti sijoitetaan sivulle nimeltä audio.html se sisältää audio-elementin sekä sen kontrollit ja itse tehdyt kontrollit. Perustoiminnallisuus testataan audio-elementin omilla kontrolleilla.

Kuva 9. IE:n audio-elementti selaimen kontrolleilla

- Selaimen versio, IE11 11.0.9600.17358
- Play, toimii
- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii

Kuva 10. Firefoxin audio-elementti selaimen omilla kontrolleilla

- Selaimen versio, 33.0.3
- Play, toimii
- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii

Kuva 11. Chromen audio-elementti selaimen kontrolleilla

- Selaimen versio, 38.0.2125.111 m
- Play, toimii

- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii

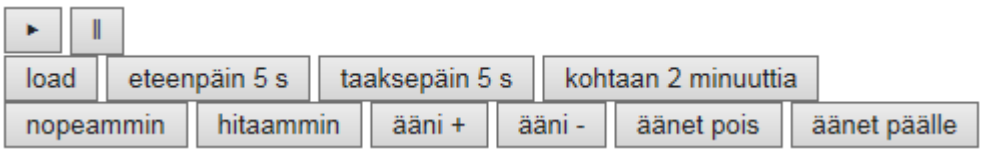
Kuva 12. Operan audio-elementti selaimen kontrolleilla

- Selaimen versio, 25.0.1614.68
- Play, toimii
- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii

Kaikissa neljässä selaimessa kontrollit toimivat moitteettomasti. Kaikki myös toistavat mp3-formaatissa olevia äänitiedostoja. IE:n kontrollit ovat selvästi suurimmat, jopa niin suuret, että ne varmastikin aiheuttavat ongelmia sivun asettelun suhteen.

4.5.4 Audio-elementin laajempi toiminnallisuus

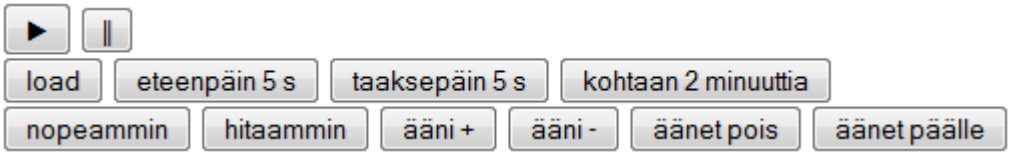
Audio-elementin laajemmassa testauksessa testataan play, pause, siirtyminen suoraan tiettyyn kohtaan toistettavassa tiedostossa, siirtyminen toistettavassa tiedostossa tietty sekuntimäärä eteenpäin ja taaksepäin, äänen toiston nopeuttaminen ja hidastaminen, äänen tason säätö ylös ja alas sekä äänen säätö pois päältä ja päälle.



Kuva 13. IE:n itse tehdyt kontrollit

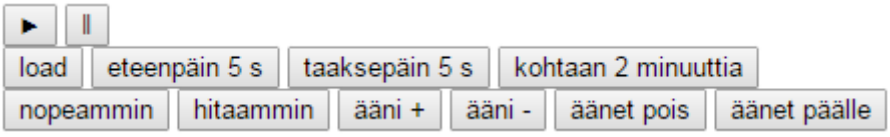
- Selaimen versio, IE11 11.0.9600.17358

- Play, toimii
- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Siirtyminen tiettyyn kohtaan, toimii
- Siirtyminen eteenpäin, toimii
- Siirtyminen taaksepäin, toimii
- Toiston nopeuttaminen, toimii
- Toiston hidastaminen, toimii



Kuva 14. Firefoxin itse tehdyt kontrollit

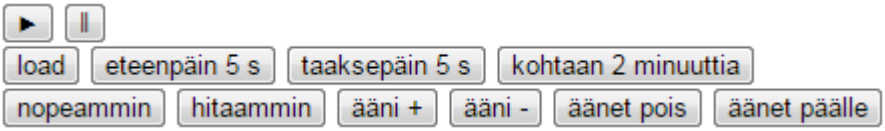
- Selaimen versio, 33.0.3
- Play, toimii
- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Siirtyminen tiettyyn kohtaan, toimii
- Siirtyminen eteenpäin, toimii
- Siirtyminen taaksepäin, toimii
- Toiston nopeuttaminen, toimii
- Toiston hidastaminen, toimii



Kuva 15. Chromen itse tehdyt kontrollit

- Selaimen versio, 38.0.2125.111 m

- Play, toimii
- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Siirtyminen tiettyyn kohtaan, toimii
- Siirtyminen eteenpäin, toimii
- Siirtyminen taaksepäin, toimii
- Toiston nopeuttaminen, toimii
- Toiston hidastaminen, toimii



Kuva 16. Operan itse tehdyt kontrollit

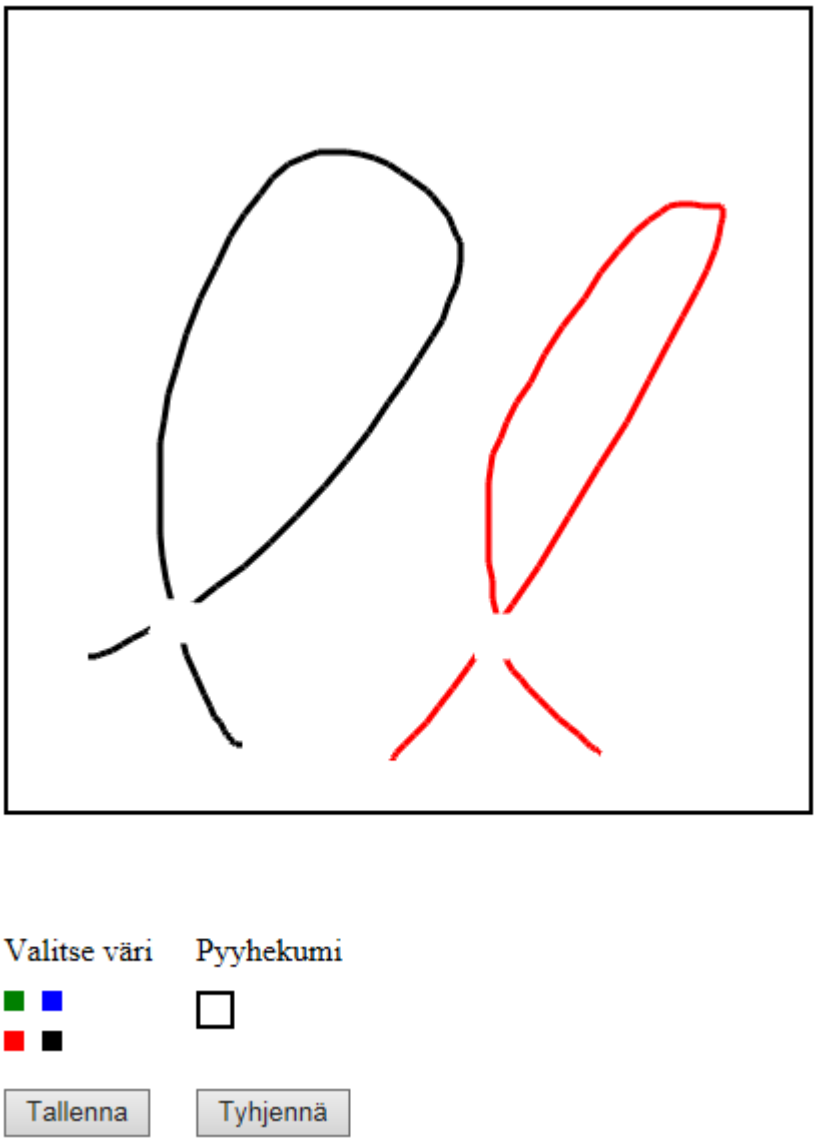
- Selaimen versio, 25.0.1614.68
- Play, toimii
- Pause, toimii
- Siirtyminen toiseen kohtaan audiotiedostossa, toimii
- Mute, toimii
- Äänen tason säätö, toimii
- Siirtyminen tiettyyn kohtaan, toimii
- Siirtyminen eteenpäin, toimii
- Siirtyminen taaksepäin, toimii
- Toiston nopeuttaminen, toimii
- Toiston hidastaminen, toimii

Itse tehdyt kontrollit toimivat kaikissa testatuissa selaimissa lähes moitteetta. Ainoa pieni kauneusvirhe on IE:n tapauksessa esiintyvä pieni hidastelu toiston nopeuden muuttamisessa. Kaikissa selaimissa toiston hidastaminen alle alkuperäisen tason toimii pausen tavoin eli äänitiedostoa ei voi kuunnella normaalia hitaampana. IE:ssä kontrollien väliin ei pystysuunnassa jää tilaa ja play- ja pausen napit ovat hieman eri paria. Firefoxissa kontrollien väliin jää pystysuunnassa riittävä tila, joskin Firefoxissakin play- ja pause-napit eivät ole saman kokoiset pystysuunnassa. Chromessa kontrollien väliin ei

jää pystysuunnassa tilaa, mutta kontrollit ovat sentään kaikki yhtä korkeita. Opera suoriutuu ulkoasuseikoista parhaiten. Kontrollien väliin jää pystysuunnassa tilaa ja kontrollit ovat samankorkuisia.

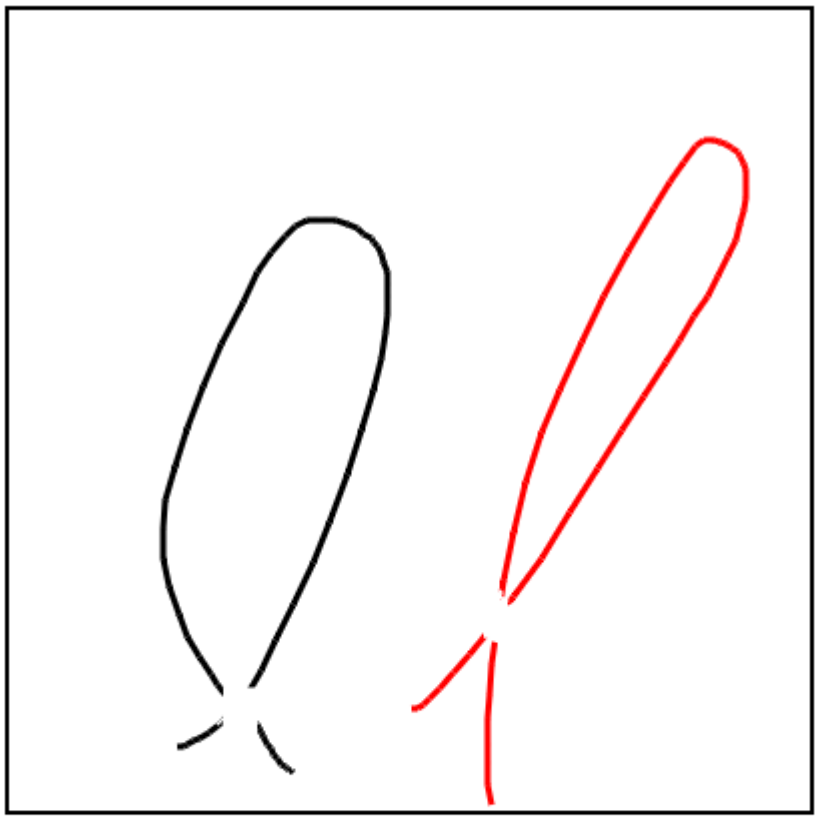
4.5.5 Canvas-elementin toiminnallisuus

Canvas-elementti sijoitetaan sivulla canvas.html. Sivu sisältää itse elementin sekä värinvaihto- kuvan pyyhkis-, sekä kopiointikontrollit.

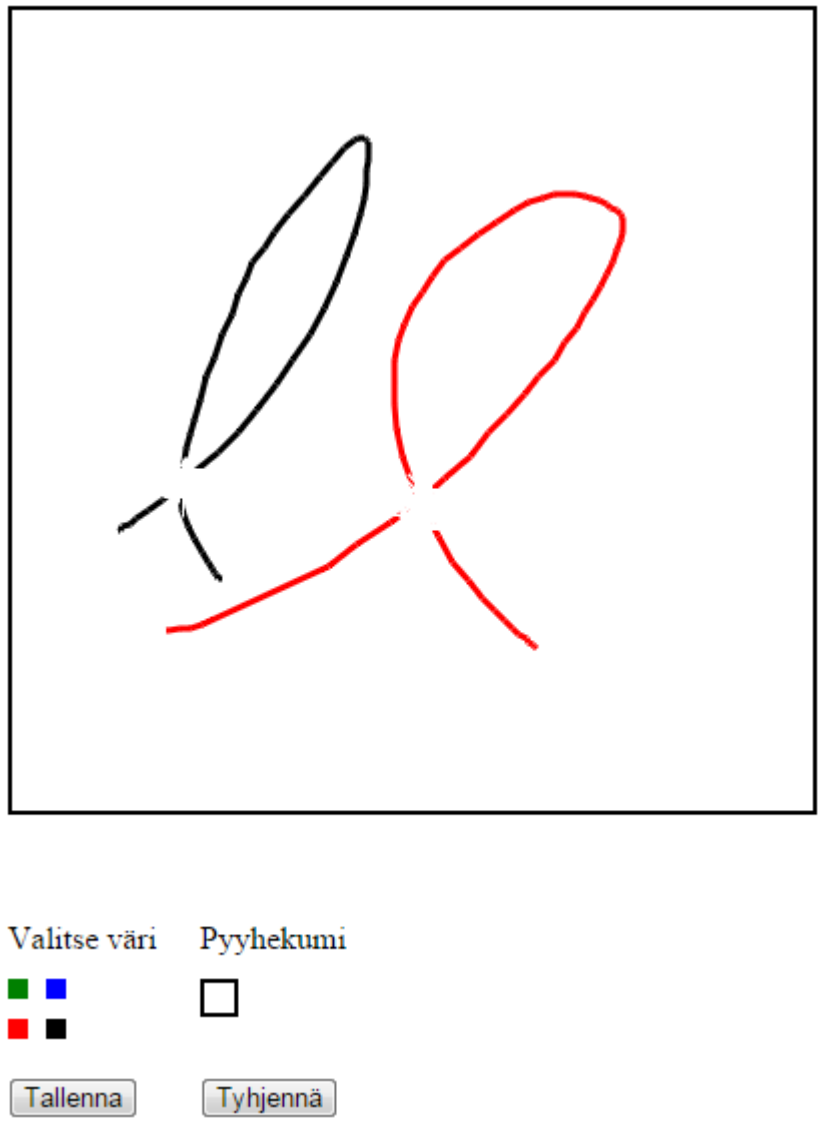


Kuva 17. IE:n canvas-elementti

Kuva 18. Firefoxin canvas-elementti



Kuva 19. Chromen canvas-elementti



Kuva 20. Operan canvas-elementti

Kaikissa neljässä selaimessa canvas-elementin toiminnallisuus ja ulkoasu ovat samanlaisia. Värin vaihto sujuu ilman ongelmia, kuten myös pyyhekumin käyttö. Pyyhekumi on käytännössä toki myös värin vaihto eli sikäli sen toiminta ei ole yllätys. Tallennus toimii kaikissa samalla tavalla. Se kopioi canvas-elementin sisällön toiseen canvas-elementtiin. Jos kuva haluaa tallentaa, sekin onnistuu kaikissa selaimissa hiiren oikealla napilla aukeavasta valikosta, jossa on ’tallenna kuva nimellä’ vaihtoehto. Myös canvaksen tyhjennys toimii moitteetta.

5 Yhteenveto

Yleisesti ottaen HTML5:n video- ja audio-elementit toimivat kaikissa neljässä selaimessa riittävän hyvin. Kaikki neljä selainta tukevat videon osalta H.264-formaattia, mutta varmuuden vuoksi kannattaa lähdetiedosto tarjota myös Ogg- ja WebM-formaatissa. Kaikki neljä selainta tukevat myös mp3-formaattia eli audion osalta voidaan aina käyttää mp3-formaattia lähdetiedostoina. Ulkoasun osalta selaimissa on huomattavia eroja selaimesta toiseen käytettäessä oletuskontrolleja ja tämä voikin aiheuttaa ongelmia sivujen ulkoasun suhteen. Oletuskontrollien aiheuttamat ulkoasuongelmat voidaan yrittää kiertää tekemällä kontrollit itse, mutta itse tehtyjen kontrollien toiminta on riippuvainen JavaScriptistä eli jos selain ei JavaScriptin toimintaa salli, eivät itse tehdyt kontrollit toimi. Tämän vuoksi onkin pakollista jättää aina mahdollisuus selaimen tekemien oletuskontrollien käytölle ja kuten Jukka Korpela HTML5 - Käsikirjassa toteaa, on oletuskontrollien pois päältä ottaminen viisainta tehdä JavaScriptillä. Tässä tapauksessa JavaScriptin toimimattomuus estää kontrollien piilottamisen ja oletuskontrollit jäävät näkyviin. Omien kontrollien ulkonäköeroja voi yrittää tasoitella tyylisivujen avulla, mutta oman kokemukseni mukaan sekään ei täydellisesti onnistu. Ulkoasun ongelmia ei voidakaan täysin luotettavasti kiertää.

Ihan kaikkia Flashin-ominaisuuksia ei voida toistaa HTML5:n avulla, mutta lista on varsin suppea. Ovaalien piirtäminen ei onnistu HTML5:llä toistaiseksi ja osa Flashin ominaisuuksista on hyvin raskasta toteuttaa HTML5:llä. Esimerkiksi osa tween-animaatioista ja hehku-efektit. Eroavaisuudet ovat kuitenkin sen verran pieniä, että Flashin käyttö ei ole pakollista niiden takia. Jos Flash on kuitenkin hyvin hallussa, niin EaselJS:n tapaiset välineet helpottavat siirtymistä HTML5:n pariin ja Flashin käyttöä kehitysalustana ei ole pakko kokonaan hylätä.

Jatkokehityksen kohteita on kaksi. Toisaalta canvas-elementti on niin laaja kokonaisuus, että sen mahdollisuuksien tutkiminen antaa aiheen hyvään ja laajaan opinnäytetyöhön. Tässä opinnäytetyössä raapaistiin vain vähän pintaa 2d-kontekstin osalta ja WebGL-kontekstia ei käsitelty lainkaan. 2d-kontekstilla voidaan toteuttaa ainakin yksinkertaisia pelejä ja WebGL antaa mahdollisuuden käsitellä 3d-grafiikkaakin. Toinen mahdollinen

jatkokehityksen kohde on Flash-sisällön kääntäminen HTML5:lle EaselJS:n tai muun vastaavaan työvälineen avulla.

6 Lähteet

Allen, J. 2011. Microsoft has abandoned Silverlight and All Other Plugins in Metro IE, InfoQ.com
Luettavissa: http://www.infoq.com/news/2011/09/Metro-Plug-ins. Luettu: 22.9.2014.

Aune, S. 2014. A brief history of Flash, TechnoBuffalo
Luettavissa: http://www.technobuffalo.com/2010/04/05/a-brief-history-of-flash/. Luettu: 30.7.2013.

Beal, V. HTML – HyperText Markup Language. Webopedia Luettavissa: http://www.webopedia.com/TERM/H/HTML.html. Luettu: 24.7.2014.

Bugnion, L. 2010, A Short History of Silverlight
Luettavissa: http://www.informit.com/articles/article.aspx?p=1645871&seqNum=4. Luettu: 22.9.2014.

DPCI. HTML5 Features and Benefits. Luettavissa: http://www.databasepublish.com/html5-features-and-benefits. Luettu 23.10.2014.

Garrett, J., 2005, Ajax: A New Approach to Web Applications, Adaptive Path
Luettavissa: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/. Luettu: 22.9.2014.

HTML Dog. Embedded Content: Video, Audio and Canvas.
Luettavissa: http://htmldog.com/guides/html/advanced/embeddedcontent/. Luettu: 18.11.2014.

HTML5 Test
Luettavissa: http://html5test.com/. Luettu: 22.10.2014

Korpela, J. 2014. HTML5 Käsikirja. Docendo.

Leiner, B, Cerf, V, Clark, D, Kahn, R, Kleinrock, L, Lynch, D, Postel, J, Roberts, L, Wolff, S. Brief History of the Internet. Luettavissa: http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet. Luettu: 8.10.2014.

Longman, Addison Wesley. 1998. A history of HTML. W3C. Luettavissa: http://www.w3.org/People/Raggett/book4/ch02.html. Luettu: 24.7.2014.

Manninen, P. & Marttila, J. 2006. Flash 8 ja ActionScript. Docendo.

McLellan, D. 2005, Very Dynamic Web Interfaces, xml.com
Luettavissa: http://www.xml.com/pub/a/2005/02/09/xml-http-request.html. Luettu: 22.9.2014.

Microsoft a. Manage add-ons in Internet Explorer. Luettavissa: http://windows.microsoft.com/en-us/internet-explorer/manage-add-ons#ie=ie-11. Luettu: 22.10.2014.

Microsoft b. Microsoft Silverlight Release History.
Luettavissa: http://www.microsoft.com/getsilverlight/locale/en-us/html/Microsoft%20Silverlight%20Release%20History.htm. Luettu: 22.9.2014.

Rousset, D. The Complete Guide to Building HTML5 Games Canvas & SVG, HTMLGoodies
Luettavissa: http://www.htmlgoodies.com/html5/client/the-complete-guide-to-building-html5-games-with-canvas-svg.html. Luettu: 21.10.2014.

Ryland Designs. The advantages and disadvantages of using Flash on a website. http://www.rylanddesigns.net/the-advantages-and-disadvantages-of-using-flash-on-a-website.php. Luettu 23.10.2014.

Skinner, G., Using the Flash Professional Toolkit for CreateJS, Adobe.com

Luettavissa: http://www.adobe.com/devnet/createjs/articles/using-flash-pro-toolkit-createjs.html. Luettu: 21.10.2014.

Ulanoff, L. 2011, It's Official: Flash Mobile Player Is Dead, Mashable Luettavissa: http://mashable.com/2011/11/09/its-official-flash-mobile-player-is-dead/. Luettu: 30.7.2014.

Warren, C. 2012, The Life, Death and Rebirth of Flash, Mashable Luettavissa: http://mashable.com/2012/11/19/history-of-flash/. Luettu: 30.7.2014.

W3C. Plan 2014. Luettavissa: http://dev.w3.org/html5/decision-policy/html5-2014-plan.html. Luettu 23.10.2014.

W3C. 1999. HTML 4.01 specification. Luettavissa: http://www.w3.org/TR/1999/REC-html401-19991224/. Luettu: 25.7.2014.

W3C. 2011. HTML5 differences from HTML4. Luettavissa: http://www.w3.org/TR/2011/WD-html5-diff-20110405/. Luettu: 30.7.2014.

W3C. 2012. W3C Candidate Recommendation. Luettavissa: http://www.w3.org/TR/2012/CR-html5-20121217/. Luettu: 23.10.2014.

W3C. 2010. HTML5 Video events and API. (Ei käytössä?) Luettavissa: http://www.w3.org/2010/05/video/mediaevents.html. Luettu: 22.10.2014.

Liitteet

Liite 1. Video-sivun lähdekoodi

```
<!DOCTYPE html>
<style>
    canvas {
        border: ridge 5px;
    }
</style>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>HTML5 Multimedia test page</title>
</head>
<body>
    <video id="video1" controls="" preload="auto" height="480" width="854">
        <source id="StarCitizenMP4" src="Videos/SC.mp4" type="video/mp4">
        <source id="StarCitizenOGG" src="Videos/SC.ogv" type="video/ogg">
        <source id="StarCitizenWEBM" src="Videos/SC.webmhd.webm" type="video/webm">
        <p>Selaimesi ei tue the HTML5:n Video-elementtiä.</p>
    </video>
    <div id="Video1Buttons">
        <!-- Tehdään videolle valmiiksi kontrollit, jotka jemmataan -->
        <button id="video1PlayButton" hidden="hidden" on-
click="video1.play()">&#x25b6;</button>
        <button id="video1PauseButton" hidden="hidden" on-
click="video1.pause()">&#x2016;</button>
        <br />
        <button id="video1LoadButton" hidden="hidden" on-
click="video1.load()">load</button>
        <button id="video1CTAddButton" hidden="hidden" onclick="video1.currentTime
+= 10">eteenpäin 5 s</button>
        <button id="video1CTSubstractButton" hidden="hidden" on-
click="video1.currentTime -= 10">taaksepäin 5 s</button>
        <button id="video1SkipButton" hidden="hidden" onclick="video1.currentTime =
120">kohtaan 2 minuuttia</button><br>
        <button id="video1RateAddButton" hidden="hidden" on-
click="video1.playbackRate++">nopeammin</button>
        <button id="video1RateSubstractButton" hidden="hidden" on-
click="video1.playbackRate--">hitaammin</button>
        <button id="video1VolumeAddButton" hidden="hidden" onclick="video1.volume +=
0.1">ääni +</button>
        <button id="video1VolumeSubstractButton" hidden="hidden" on-
click="video1.volume -= 0.1">ääni -</button>
        <button id="video1MuteOnButton" hidden="hidden" onclick="video1.muted =
true">äänet pois</button>
        <button id="video1MuteOffButton" hidden="hidden" onclick="video1.muted =
false">äänet päälle</button><br>
    </div>
    <script>
        //piilotetaan oletuskontrollit
        video1.controls = false;
        //laitetaan napit valmiiksi tehdyt napit näkyviin
        video1LoadButton.hidden = "";
        video1PlayButton.hidden = "";
        video1PauseButton.hidden = "";
        video1CTAddButton.hidden = "";
        video1CTSubstractButton.hidden = "";
        video1SkipButton.hidden = "";
        video1RateAddButton.hidden = "";
        video1RateSubstractButton.hidden = "";
        video1VolumeAddButton.hidden = "";
        video1VolumeSubstractButton.hidden = "";
        video1MuteOnButton.hidden = "";
        video1MuteOffButton.hidden = "";
```

```
    </script>
</body>
</html>
```

Liite 2. Audio-sivun lähdekoodi

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <audio id="dangerZone" controls="">
        <source id="audioFile" src="Audios/DZ.mp3" type="audio/mp3" />
    </audio>
    <div id="AudioButtons">
        <button id="DZPlayButton" onclick="dangerZone.play()" hid-
den>&#x25b6;</button>
        <button id="DZPauseButton" onclick="dangerZone.pause()" hid-
den>&#x2016;</button>
        <br />
        <button id="DZLoadButton" onclick="dangerZone.load()" hidden>load</button>
        <button id="DZCTAddButton" onclick="dangerZone.currentTime += 10" hid-
den>eteenpäin 5 s</button>
        <button id="DZPCTSubstractButton" onclick="dangerZone.currentTime -= 10"
hidden>taaksepäin 5 s</button>
        <button id="DZSkipButton" onclick="dangerZone.currentTime = 120" hid-
den>kohtaan 2 minuuttia</button><br>
        <button id="DZRateAddButton" onclick="dangerZone.playbackRate++" hid-
den>nopeammin</button>
        <button id="DZRateSubstractButton" onclick="dangerZone.playbackRate--" hid-
den>hitaammin</button>
        <button id="DZVolumeAddButton" onclick="dangerZone.volume += 0.1" hid-
den>ääni +</button>
        <button id="DZVolumeSubstractButton" onclick="dangerZone.volume -= 0.1" hid-
den>ääni -</button>
        <button id="DZMuteOnButton" onclick="dangerZone.muted = true" hidden>äänet
pois</button>
        <button id="DZMuteOffButton" onclick="dangerZone.muted = false" hidden>äänet
päälle</button><br>
    </div>
<script>
    //piilotetaan oletuskontrollit
    dangerZone.controls = false
    //laitetaan napit valmiiksi tehdyt napit näkyviin
    DZLoadButton.hidden = "";
    DZPlayButton.hidden = "";
    DZPauseButton.hidden = "";
    DZCTAddButton.hidden = "";
    DZPCTSubstractButton.hidden = "";
    DZSkipButton.hidden = "";
    DZRateAddButton.hidden = "";
    DZRateSubstractButton.hidden = "";
    DZVolumeAddButton.hidden = "";
    DZVolumeSubstractButton.hidden = "";
    DZMuteOnButton.hidden = "";
    DZMuteOffButton.hidden = "";
</script>
</body>
</html>
```

Liite 3. Canvas-sivun lähdekoodi

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Canvas</title>
</head>
<body onload="alusta()">
    <canvas id="piirtoAlusta1" width="400" height="400"
style="position:absolute;top:10%;left:10%;border:2px solid;"></canvas>
    <div style="position:absolute;top:57%;left:10%;">Valitse väri</div>
    <div
style="position:absolute;top:60%;left:10%;width:10px;height:10px;background:green;"
id="vihrea" onclick="vari(this)"></div>
    <div
style="position:absolute;top:60%;left:11%;width:10px;height:10px;background:blue;"
id="sininen" onclick="vari(this)"></div>
    <div
style="position:absolute;top:62%;left:10%;width:10px;height:10px;background:red;"
id="punainen" onclick="vari(this)"></div>
    <div
style="position:absolute;top:62%;left:11%;width:10px;height:10px;background:black;"
id="musta" onclick="vari(this)"></div>
    <div style="position:absolute;top:57%;left:15%;">Pyyhekumi</div>
    <div
style="position:absolute;top:60%;left:15%;width:15px;height:15px;background:white;bo
rder:2px solid;" id="valkoinen" onclick="vari(this)"></div>
    <img id="canvasimg" style="position:absolute;top:10%;left:52%;display:none;">
    <input type="button" value="Tallenna" id="btn" size="30" onclick="tallenna()"
style="position:absolute;top:65%;left:10%;">
    <input type="button" value="Tyhjennä" id="clr" size="23" onclick="tyhjenna()"
style="position:absolute;top:65%;left:15%;">

    <script>
    var canvas, konteksi, merkitse = false,
    previousX = 0,
    currentX = 0,
    previousY = 0,
    currentY = 0,
    maalaa = false;

    var piirtoVari = "black";
    var y = 3;

    function alusta() {
        piirtoalusta = document.getElementById('piirtoAlusta1');
        leveys = piirtoalusta.width;
        korkeus = piirtoalusta.height;

        konteksi = piirtoalusta.getContext("2d");

        piirtoalusta.addEventListener("mouseup", function (e) {
            etsiKoordinaatit('up', e)
        }, false);
        piirtoalusta.addEventListener("mousedown", function (e) {
            etsiKoordinaatit('down', e)
        }, false);
        piirtoalusta.addEventListener("mousemove", function (e) {
            etsiKoordinaatit('move', e)
        }, false);
        piirtoalusta.addEventListener("mouseout", function (e) {
            etsiKoordinaatit('out', e)
        }, false);
    }
```

```
function vari(obj) {
    switch (obj.id) {
        case "sininen":
            piirtoVari = "blue";
            break;
        case "musta":
            piirtoVari = "black";
            break;
        case "vihrea":
            piirtoVari = "green";
            break;
        case "punainen":
            piirtoVari = "red";
            break;
        case "valkoinen":
            piirtoVari = "white";
            break;
    }
    if (piirtoVari == "white") y = 21;
    else y = 3;

}

function piirra() {
    konteksi.beginPath();
    konteksi.moveTo(previousX, previousY);
    konteksi.lineTo(currentX, currentY);
    konteksi.strokeStyle = piirtoVari;
    konteksi.lineWidth = y;
    konteksi.stroke();
    konteksi.closePath();
}

function tyhjenna() {

    konteksi.clearRect(0, 0, leveys, korkeus);
    document.getElementById("canvasimg").style.display = "none";

}

function tallenna() {
    var dataURL = piirtoalusta.toDataURL();

    document.getElementById("canvasimg").style.border = "3px solid";
    document.getElementById("canvasimg").src = dataURL;
    document.getElementById("canvasimg").style.display = "inline";
}

function etsiKoordinaatit(res, e) {
    if (res == 'down') {
        previousX = currentX;
        previousY = currentY;
        currentX = e.clientX - piirtoalusta.offsetLeft;
        currentY = e.clientY - piirtoalusta.offsetTop;

        merkitse = true;
        maalaa = true;
        if (maalaa) {
            konteksi.beginPath();
            konteksi.fillStyle = x;
            konteksi.fillRect(currentX, currentY, 2, 2);
            konteksi.closePath();
            maalaa = false;
        }
    }
```

```
        if (res == 'up' || res == "out") {
            merkitse = false;
        }
        if (res == 'move') {
            if (merkitse) {
                previousX = currentX;
                previousY = currentY;
                currentX = e.clientX - piirtoalusta.offsetLeft;
                currentY = e.clientY - piirtoalusta.offsetTop;
                piirra();
            }
        }
    }
    </script>
</body>
</html>
```