HTML5:n uudet ominaisuudet. [WebGL (Three.js), Web Storage, Geolocation, Drag and Drop, Canvas]

3D demo-peli Sokkelo

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö
Tietotekniikan koulutusohjelma
HAMK Riihimäki
kevät, 2017

Semjon Patchine

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Koulutus Tietotekniikka
Kampus HAMK Riihimäki

Tekijä Semjon Patchine **Vuosi** 2017

Työn nimi HTML5:n uudet ominaisuudet. 3D demo-peli Sokkelo

Työn ohjaaja Petri Kuittinen

TIIVISTELMÄ

3D-grafiikka ja virtuaalinen todellisuus ovat olleet yhdessä yksi eniten tietokoneiden kehitystä eteenpäin vievistä voimista. Tänä päivänä netin sivustoilla on oltava MultiMedian käyttöä (kuvaa, ääntä, videota ja interaktiivisia dynaamisia sovelluksia, kuten esimerkiksi pelejä ja muita sisäohjelmia).

Opinnäytetyön teoriaosuudessa käydään läpi HTML5:n uudet ominaisuudet ja perustoiminnot.

Keskityin opinnäytetyössä peliohjelmointiin, koska se on monipuolisin tapa esitellä ja opiskella uutta teknologiaa. Opinnäytetyön suorittamista varten rakennettiin 3D Demo-pelin web-sivu, jolla käytetään näitä ominaisuuksia. Grafiikan ohjelmointi voi olla erittäin hidasta. Tämän vuoksi on hyvä harkita apukirjaston käyttöä nopeuttamaan kehitysprosessia. Tässä opinnäytetyössä käytin Three.js-kirjastoa. Kerroin askel askeleelta, kuinka oma pelinkehitysprosessini eteni ja mitä tuloksia sain aikaan. Näin pääsin käytännössä kokeilemaan lähempää tarkastelua varten otettuja ohjelmointirajapintoja.

Avainsanat 3D, HTML5, WebGL verkkoselain

Sivut 51 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology
Campus HAMK Riihimäki

Author Semjon Patchine **Year** 2017

Subject HTML5's new features 3D demo game labyrinth

Supervisors Petri Kuittinen

ABSTRACT

The purpose of the thesis was to introduce a new technology called WebGL and show how it could be used in software development.

The theoretical part of the thesis goes through the description of the implementation of the website.

In this thesis the focus was in game programming, as it is the most versatile way to introduce and study new technology. To accomplish this thesis, I build a 3D Demo Web site that uses these features.
Graphics programming can be very slow. Therefore, it is good to consider the use of auxiliary library to speed up the development process.
In this thesis I used the Three.js-library. I'm going to tell you step by step how the game development progressed and what results was achieved.

This will allow me to practice the programming interfaces for closer viewing.

Keywords 3D, HTML5, WebGL, web Browser

Pages 51 pages

SISÄLLYS

1 JOHDANTO 1

2 TEKNOLOGIAT JA HTML5:N KAKSOISMERKITYS 1

2.1 Mikä HTML on? 2
2.2 HTML5 5
2.3 HTML:n rakenteita ja käsitteitä 6
2.4 Dokumentin rakenne 7
2.5 Tagit ja elementit 8
2.5.1 Ylätunniste <header> 9
2.5.2 Navigaatio <nav> 10
2.5.3 Sivupalkki <aside> 10
2.5.4 Otsikkoelementit <h1>-<h6> 11
2.5.5 <section> ja <article> 11
2.5.6 Alatunniste <footer> 12
2.5.7 <ul>, <ol>, <li> 12
2.6 DOM-puu 13
2.7 Lomakkeet sivu 14

3 WEBGL 15

4 JAVASCRIPTIN PERUSTEITA 18

4.1 JavaScriptin rooli HTML:n yhteydessä 18
4.2 JavaScript ohjelmointikielenä 19
4.3 JavaScriptin perusrakenteita 19
4.4 Kirjastot 21

5 CSS3 23

5.1 CSS:n peruskäsitteitä 23

6 PAIKKATIEDOT (GEOLOCATION). GOOGLE MAPS API 25

6.1 Paikannuksen hyödyt 25
6.2 Geolocation – sovellusliittymä 25

7 VIDEO 26

7.1 Videomuodot 28

8 GRAAFINEN ESITYSALUE CANVAS – PIIRTOALUSTA 30

9 WEB STORAGE 31

10 VETÄMINEN JA PUDOTTAMINEN (DRAG AND DROP DND) 32

11 3D DEMO-PELI SOKKELO 33

11.1 Aloittaminen 33
11.2 Perusrunko 34

11.3 Työn kuvaus 34
11.3.1 Sokkelon algoritmi 36
11.4 labyrinth.html:n Init-funktion määrittelyt 43
11.4.1 Init-funktion rakenne 44

12 YHTEENVETO 47

12.1.1 Käyttöliittymänkehityksen haasteet 48

LÄHTEET 50

1 JOHDANTO

3D-grafiikka ja virtuaalinen todellisuus ovat olleet yhdessä yksi eniten tietokoneiden kehitystä eteenpäin vievistä voimista. Tänä päivänä netin sivustoilla on oltava MultiMedian käyttöä (kuvaa, ääntä, videota ja interaktiivisia dynaamisia sovelluksia, kuten esimerkiksi pelejä ja muita sisäohjelmia). Dynaamisuus Web-sivuihin on lisääntynyt, koska on alettu vaatia työpöytäsovellusten kaltaista käyttömukavuutta.

WebGL on OpenGL-grafiikkarajapinta, joka on sulautettu Internet-selaimeen. WebGL vähentää entistään ohjelmoinnin käännöstyötä eri alustoille, sillä WebGL-rajapintaa tukevia selaimia on kaikilla käyttöjärjestelmillä.

Opinnäytetyön teoriaosuudessa käydään läpi HTML5:n uudet ominaisuudet ja perustoiminnot.

HTML5-standardista rajattiin kehitystehtäviksi alkuvaiheessa muutama kokonaisuus, joita olivat semanttiset elementit (tag), <canvas> -elementti, raahaus ja pudotus (Drag and Drop) -rajapinta, lomakkeet ja videoiden näyttämiseen tarkoitettu <video> -elementti. Tietovarastojen (Web Storage) tuki auttaa säilyttämään web-sivun sisältöjä (esim. pelinhetkiä) talteen.
Geolocation (Google Maps APIs) on tällä hetkellä hyvin tuettu ja tarjoa tämän työn mielenkiintoisimman kohteen.

Keskityn opinnäytetyössä peliohjelmointiin, koska se on monipuolisin tapa esitellä ja opiskella uutta teknologiaa. Opinnäytetyön suorittamista varten rakennettiin 3D Demo-pelin web-sivu, jolla käytetään näitä ominaisuuksia. Grafiikan ohjelmointi voi olla erittäin hidasta. Tämän vuoksi on hyvä harkita apukirjaston käyttöä nopeuttamaan kehitysprosessia. Tässä opinnäytetyössä käytän WebGL-pohjaista Three.js-kirjastoa. Kerron askel askeleelta, kuinka oma pelinkehitysprosessini eteni ja mitä tuloksia sain aikaan. Näin pääsen käytännössä kokeilemaan lähempää tarkastelua varten otettuja ohjelmointirajapintoja.

2 TEKNOLOGIAT JA HTML5:N KAKSOISMERKITYS

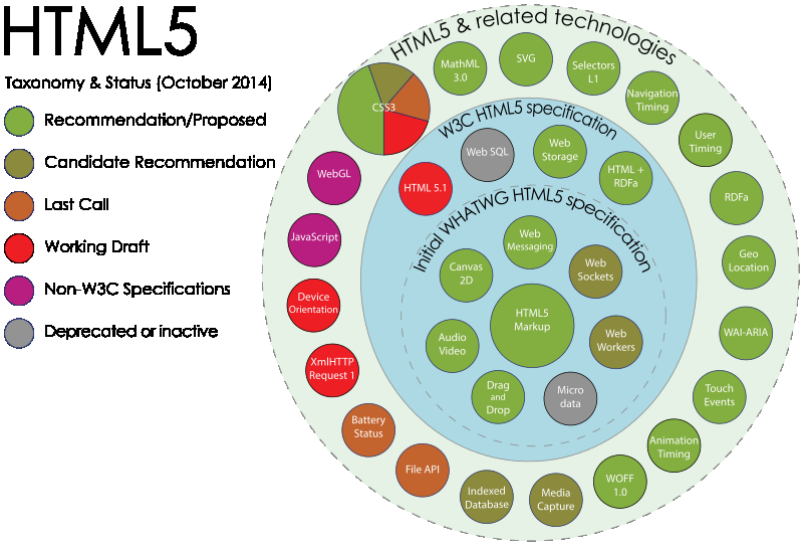
HTML5-kieli on uusi versio verkkosivujen tekemiseen yleisesti käytetystä HTML-merkintäkielestä. HTML5 viittaa nykyisin usein yleisesti moderneihin web-tekniikoihin. Niihin kuuluvat HTML5-kielen uutuuksien ohella esimerkiksi CSS:n uudet piirteet (CSS3) sekä sovellusliittymät kuten File API ja Geolocation API. Sovelluskehityksen alalla HTML5 tarkoittaa nykyi-

sin yleisesti sovellusten toteuttamista webin avoimilla tekniikoilla. Ohjelmointikielenä on tällöin JavaScript ja muotoilun perusvälineenä CSS.

Näiden lisäksi usein käytetään erilaisia JavaScript ja CSS lisäosia, kuten jQuery, CoffeeScript tai Sass. Nämä nopeuttavat työskentelyä ja auttavat tekemään sivuista moderneja sekä ulkonäöltään että toiminnoiltaan.

On hyvä mainita, että HTML-kieli ja sen uutuudet ovat melko pienessä roolissa HTML5-soveluksissa. Ilmauksella HTML5 on selvästi kaksi eri merkitystä.

Sovellusmarkkinoilla se tarkoittaa sovellusten toteuttamisen tapaa, joka perustuu web-tekniikoiden käyttöön.
Toinen merkitys on, että HTML5 tarkoittaa HTML-kielen ja sen määrittelyn (standardin) uutta kehitysvaihetta, jossa on uusia elementtejä ja muita piirteitä.
Sekaannus HTML5-sovelusten ja HTML5-kielen välillä on aika yleistä.



Kuva 1. HTML5:n rakenne

2.1 Mikä HTML on?

HTML on lyhyesti sanottuna eräs rakenteinen dokumenttien muoto, joka on käsiteltävissä mitä erilaisimmissa tilanteissa, kuten näytettävissä isolla tai pienellä kuvaruudulla, esitettävissä ääneen tai sokeainkirjoituksella ja analysoitavissa ohjelmilla, jotka laativat tietokantoja dokumenttien sisällöstä. HTML-dokumentti sisältää tekstiä, rakenteen osoittavaa merkkausta (esim. <h2> rakennetta mukaan (HTML)</h2> merkkaa tekstin 2. tason otsikoksi) sekä erityyppisiä viittauksia muihin dokumentteihin, jotka voivat olla myös esim. kuva- tai äänidataa. HTML:n yksityiskohtainen kuvaus sisältyy HTML5-spesifikaatioon.

HTML:ää kutsutaan usein kieleksi (language), tarkemmin sanoen merkkauskieleksi (markup language); itse nimi HTML johtuu alun perin sanoista HyperText Markup Language. ATK-alalla käytetyt kielten nimitykset ovat usein varsin harhaanjohtavia, etenkin, jos HTML sekoitetaan ohjelmointikieliin tms. Selvempi on siis puhua HTML:stä tiedostomuotona tai tarkemmin sanoen dataformaattina, jossa tekstin oheen on liitetty sen rakennetta osoittavia merkintöjä.

HTML on kieli web-sivustojen luomiseen. HTML ei ole ohjelmointikieli, vaan kuvauskieli, jonka avulla kuvataan sekä web-sivun rakenne että sivun sisältämä teksti. HTML-sivujen rakenne määritellään HTML-kielessä määritellyillä elementeillä, ja yksittäinen HTML-dokumentti koostuu sisäkkäin ja peräkkäin olevista elementeistä.

Sivujen rakenteen määrittelevät elementit erotellaan pienempi kuin (<) ja suurempi kuin (>) -merkeillä. Elementti avataan elementin nimen sisältävällä pienempi kuin (<) -merkillä alkavalla ja suurempi kuin (>) -merkkiin loppuvalla merkkijonolla, esim. <html>, ja suljetaan merkkijonolla jossa elementin pienempi kuin (<) -merkin jälkeen on vinoviiva, esim. </html>. Yksittäisen elementin sisälle voi laittaa muita elementtejä.

Suurin osa elementeistä tulee sulkea lopuksi. Osa HTML5:n elementeistä - esimerkiksi
 - on kuitenkin ns. tyhjiä ("void"), eikä niille kirjoiteta erillistä lopetusta. Halutessaan tyhjät elementit voi lopettaa X(HT)ML-tyyliseen /-merkkiin, esimerkiksi seuraavasti:
.

HTML5-dokumentti sisältää dokumentin tyypin ilmaisevan aloitustagin (<!DOCTYPE html>), dokumentin aloittavan html-elementin (<html>), otsake-elementin ja sivun otsikon (<head>, jonka sisällä <title> ja <meta charset="utf-8" />), sekä runkoelementin (<body>).
Metaelementti, jota käytetään lisävinkin antamiseen selaimelle: "dokumentissa käytetään UTF-8 -merkistöä". Tämä kannattaa olla HTML5 -dokumenteissa aina.
Elementit voivat sisältää attribuutteja ja niille voi antaa arvoja. Esimerkiksi `<img id="gal1" src="images/galery/bridge.jpg" title="this is a bridge in USA" />`. Esimerkissä <img> -elementille on määritelty erillinen attribuutti src, joka kertoo dokumentissa käytetystä kuvasta.
Nykyaikaiset web-sivut sisältävät paljon muutakin kuin sarjan HTML-elementtejä. Linkitetyt resurssit, kuten kuvat ja tyylitiedostot, ovat oleellisia sivun ulkoasun ja rakenteen luomisessa.

2.1.1 HTML:n historiaa

Seuraavassa kuvataan lyhyesti HTML:n historiaa.

Vuodesta 1980, on meillä ollut monia eri HTML-versioita

Versio	Vuosi
Tim Berners-Lee keksi internetin	1989
Tim Berners-Lee keksi HTML kielen	1991
Dave Raggett kirjoitti HTML+ kielen	1993
HTML yhdistys määritteli HTML 2.0 kielen	1995
W3C Valmisteli HTML 3.2 kielen	1997
W3C Valmisteli HTML 4.01 kielen	1999
W3C Valmisteli XHTML 1.0 kielen	2000
HTML5 kielen ensimmäinen julkinen esiintyminen	2008
HTML5 kielestä alettiin tehdä standardia	2012
HTML5 kielestä tuli standardi!	2014

2.1.2 ”Klassinen” HTML

HTML suunniteltiin ja sen ensimmäiset toteutukset tehtiin vuosina 1990–1991. Aluksi HTML:ää kehitettiin CERNissä, ja vasta v. 1995 laadittiin ensimmäinen kielen julkinen määrittely, spesifikaatio: IETF:n HTML2.0. Siitä jäi pois esimerkiksi <section> -elementti ja se tuli määrittelyihin ja selaimiin vasta HTML5:ssä.

HTML 3.2:sta alkaen HTML:n määrittelyä hoiti W3C-organisaatio. ”Klassinen” HTML:n kehitys lopetettiin v. 1998. Kehitystyötä suunnattiin muun muassa dokumenttioliomallien eli DOMien määrittelyyn. Yksi syy tähän oli se että johtavien selainten DOMeissa oli olennaisia eroja, jotka aiheuttivat hankaluuksia (Korpela 2014, 28).

2.1.3 XHTML

XHTML:n tausta on XML:ssä, joka on luonteeltaan yleinen merkkauskieli. Esimerkiksi HTML-tagista:
<input type=radio checked> tulee XHTML:ssä <input type="radio" checked="checked" />,

“puhtaassa” XML-kielissä olisi oikea muoto:
<input><type> "radio" </type> <checked> "checked" </checked> </input>.

XML levisi hyvin monenlaiseen käyttöön. Kieli jaettiin moduuleihin ja ajatuksena oli että moduuleista voi koota uusia versioita ja lisäksi eri ryhmät voisivat määritellä uusia moduuleja. Työ ajautui kuitenkin umpikujaan.

2.1.4 Uuden HTML:n synty. Yhä työn alla, eli ei "valmis"

Eri tahoilla kehiteltiin ajatusta HTML:n kehittämisestä vanhalta pohjalta, laajennusten kautta.

Eräs varhainen suunnitelma oli lomakkeiden toiminnollisuuden laajennus, Web Forms 2,0. Toinen, yleisemmin HTML:n kehitystä hahmotteleva

suunnitelma oli Web Applications 1.0. Myöhemmin suunnitelmat yhdistettiin ja kokonaisuudessa ruvettiin käyttämään nimitystä HTML5.

Tilanne on muodostunut seuraavanlaiseksi:
WHATWG jatkaa työtään ja käyttää nimitystä ”Living HTML Standard”. Sen sisältö muuttuu usein päivittäin. Tätä työtä kuvailee sivu wiki.whatwg.org/wiki/FAQ (WHATWG Wiki 2017).

W3C on määritellyt HTML5:n suositusehdokkaana. Vaikka joitakin asioita on jätetty pois, lähinnä siksi, että ne halutaan kuvata eri määrittelyissä. Saman aikaan W3C jatkaa työtä HTML5:n laajentamiseksi, toistaiseksi työnimellä HTML5.1.

2.2 **HTML5**

HTML5 on uusin HTML-kielen standardoitu versio, joka tuo useita interaktiota merkittävästi edistäviä ominaisuuksia HTML-kieleen. HTML5-termillä tarkoitetaan usein myös HTML5:n, CSS3:n ja JavaScriptin tekniikoiden yhdessä muodostamaa kokonaisuutta. Todellisuudessa silloin kyseessä on ”ohjelmointikieli” nimeltä HTML5. HTML5:tä käytetään yleisnimityksenä uusille avoimille webtekniikoille joita käytetään sivujen ja sovellusten luomisessa.

HTML5:n Canvas-rajapinta tarjoaa aiempaa huomattavasti kattavammat mahdollisuudet tehdä graafisia sovelluksia. <Canvas> mahdollistaa ensimmäistä kertaa alueen vapaalle piirtämiselle ilman liitännäisiä tai HTML-elementtien rajoitteita. Sitä hyödyntämällä voidaan tehdä näyttäviä pelejä ja animaatioita. <Canvas> -elementtiä on selaimesta ja laitteistosta riippuen myös mahdollista käyttää kolmiulotteisena versiona.

Yksi CSS-tyylimäärittelyiden kolmannen version merkittävimmistä uusista ominaisuuksista on responssivisuuden mahdollistava Media Query. Responssivisella sivulla määritellään erilaiset tyylit esimerkiksi käyttäjän päätelaitteen näytön leveyden tai tarkkuuden perusteella. Media Query -määrittelyiden avulla sama HTML-sivu voidaan sopeuttaa erilaisille näytöille, kuten työpöytätietokoneelle, tablet-laitteelle ja matkapuhelimelle, ilman erillisiä HTML-versioita samasta sisällöstä. Sitä varten käytetään seuraavan metaelementin: `<meta name="viewport" content="width=device-width, initial-scale=1.0">`.
Merkittäviä uusia CSS-ominaisuuksia ovat myös siirtymät (transition), muunnokset (transform) ja animaatiot, jotka mahdollistavat monipuoliset HTML-elementtien liikkeet tehokkaasti laitteiston näytönohjainta hyödyntäen.

Uusimpien selainten tarjoama Web Storage -rajapinta tarjoaa evästeitä (cookies) joustavamman tallennustilan käyttäjän tietokoneelle. Tilaa on enemmän ja sen käsittely tapahtuu vain pyydettäessä selaimen ja JavaScript-sovelluksen välillä. Kaikki evästeet siirrettiin aina jokaisen sivupyyn-

nön mukana luoden ylimääräistä kuormaa verkkoliikenteeseen. Web Storage koostuu kahdesta osasta: istunnon ajan säilyvästä Session Storge -tallennustilasta ja pidempiaikaisesta Local Storage -tallennustilasta.

Geolocation-rajapinta tarjoaa käyttäjän sijaintitiedot, mikäli laitteistossa on paikannusmahdollisuus.

2.2.1 Kumpi on oikea HTML5

W3C:n HTML:n rinnalla kehitetään jo kielen seuraavaa versiota, tällä hetkellä työnimellä HTML5.1. jonka kuvaus on sivulla HTML5.1 Nightly, [www.w3.org/TR/html51/]. Vaikka se muuttuu päivittäin, jopa siihenkin saatetaan viitata ”standardina” (W3C n.d.).

WHATWG:n HTML-määrittely [www.whatwg.org/html/] sisältää erittäin helppokäyttöisen kommentointi mahdollisuuden. WHATWG:n “Elävät standardit”-määrittelyä voi kommentoida julkisesti samalla kun lukee sitä (W3C n.d.).

2.3 HTML:n rakenteita ja käsitteitä

Seuraavassa on hyvin yksinkertainen HTML-sivu, joka on kirjoitettu HTML5:n mukaista HTML-syntaksia käyttäen:

```
<! doctype html>
<meta charset=utf-8 />
<title>Demo</title>
<h1>Yksinkertainen HTML-sivu</h1>
<p>Hei <em>kaunis</em> maailma!</p>
```

Merkkauksen lyhyys johtuu siitä, että useita tageja on jätetty pois tavalla, joka on sallittua HTML5-syntaksissa, mutta ei HTML:n aiemmissa versioissa. Vaikka yhteensopivuuden takia on kuitenkin parasta merkata myös <html>, <head> ja <body>. Kun nämä tagit lisätään, sivun merkitys, ulkoasu ja toiminnollisuus tai DOM-puu ei muutu mitenkään, mutta merkkauksesta ehkä näkyy selvemmin sisäinen rakenne:

```
<!DOCTYPE html>
<html
<head>
<meta charset="utf-8" />
    <title>Test</title>
</head>
<body>
    <h1>Yksinkertainen HTML-sivu</h1>
    <p>tässä työssä on luotu sokkelo 3d:ssä apuna käyttäen
<em>three.js</em>.</p>
</body>
</html>
```

2.4 Dokumentin rakenne

Ennen HTML5:ttä dokumentin rakenteen jaot määriteltiin pääosin käyttäen -elementtiä. Asettamalla -elementin id-attribuutille kuvaava teksti on jaoille saatu nimet ja merkitys, kuten `<div id="header">` jolla on merkitty otsikkoalue.

HTML5-spesifikaation dokumentaation ylläpitäjä Ian Hickson kävi vuonna 2004 Google Indexin avulla läpi lähes miljardi internetsivua. Hän halusi selvittää minkälaisia rakenteita internetsivut sisältävät ja osana selvitystä hän julkaisi listauksen suosituimmista luokan nimistä (class-attribuuteille annetuista arvoista). Vuonna 2009 Operan MAMA-hakurobotti kävi läpi noin 2 miljoonaa satunnaisesti valittua URL:ia tarkistaen mitä class-attribuutteja sivujen elementeille oli annettu sekä noin 1,8 miljoonaa URL:ia tallentaen sivustojen id-attribuutit. Näiden tietojen pohjalta saatiin tietoa rakenteista joita internetsivustoilla esiintyy. HTML5:n uudet rakenteelliset elementit ovat saaneet innoituksensa näistä tiedoista (Lawson, Bruce & Sharp, Remy 2012).

Standardin pyrkimyksenä ei ole ollut luoda täysin uusia dokumentin rakenteita pienen asiantuntijaryhmän sanelemana, vaan ennemmin mahdollistaa nykyiset käytössä olevat rakenteet suoraan omilla elementeillään.

Elementeillä <nav>, <header>, <article>, <section>, <aside> ja <footer> rakennetaan HTML5-sivuston ulkoasu, kun ennen yleisenä tapana oli käyttää -elementtiä kaikkeen sijoitteluun. -elementin käyttö perustui pitkälti CSS:n kanssa toimimiseen, muuten ei voitu sijoittaa rakennelohkoja CSS:lla sujuvasti, mutta nyt CSS voidaan liittää suoraan rakenne-elementeille.
<section>:in käyttö ei ole vielä vakiintunut web-sivuilla, mutta se ei ole tarkoitettu geneeriseksi sisältöastiaksi, kuten . Pelkkä <article> on hyvä, ilman ympäröivää <section> jos on tarkoitus kuvata semanttisesti yksi artikkeli. <section> voisi olla esim. hakumoottorin hakutulokset.

<embed> elementillä luodaan ’säiliö’ ulkopuoliselle sovellukselle. HTML5 ei tue enää seuraavia elementtejä: <acronym>, <applet>, <big> jne. Täydellinen ja viimeisin lista uusista ja poistuvista elementeistä löytyy W3C:n sivuilta (HTML5 Specification).

Nykyisin hyvänä käytäntönä pidetään sisällön ja ulkoasun erottamista erillisiin dokumentteihin
– HTML = sisältö
– CSS = ulkoasu
Elementti tarkoittaa yhden tagin ja sen lopputagin muodostamaa kokonaisuutta.
– <elementti></elementti>

• Etuna helppo päivitettävyys, ulkoasu voidaan muuttaa kerralla koko sivustolle.
• Voidaan määritellä ulkoasuja eri mediatyypeille käyttäen samaa HTML-dokumenttia (Pinja Hokkanen 2016).

2.5 Tagit ja elementit

Yleisesti HTML:ssä elementti koostuu alkutagista, elementin sisällöstä ja lopputagista. Alkutagissa on elementin nimi merkkien ”<”ja”>” välissä, esimerkiksi <h1>. Lopputagi on samaa muotoa, mutta nimen edessä on vinoviiva”/”, esimerkiksi </hl>. Lopputagi voidaan usein jättää pois elementtikohtaisten sääntöjen mukaan. (Eräät elementit kuten br koostuvat pelkästä alkutagista. Nämä ns. tyhjät elementit ovat HTML 4.0:ssa seuraavat: area, base, basefont, br, col, frame, hr, img, input, isindex, link, meta, param.)

Tavallisesti elementissä on erikseen aloitustunniste ja vinoviivalla merkitty lopetustunniste sekä niiden väliin jäävä sisältö (jossa voi olla tekstiä ja myös muita HTML-elementtejä): `<tunniste>elementin sisältö</tunniste>`

Aloitustunnisteeseen saattaa lisäksi sisältyä attribuutteja, jotka määrittävät elementin ominaisuuksia tarkemmin kuin tunnisteen nimeämä elementin tyyppi:

```
<tunniste attribuutti="arvo">elementin sisältö</tunniste>
```

Lopputagi ei useinkaan ole pakollinen. On kuitenkin helpompi opetella kirjoittamaan aina lopputagi kuin opetella melko mutkikkaat säännöt siitä, milloin sen saa jättää pois ja milloin ei. Sitä paitsi XHTML:ssä lopputagi on pakollinen.

Jossakin tapauksissa (mm. html-, head- ja body-elementti) myös alkutagi voidaan jättää pois (Korpela 2014, 46).

Block (lohko) tyyliset elementit

<address>, <article>, <aside>,
<blockquote>,
<dd>, , <dl>, <dt>, <details>,
<fieldset>, <figcaption>, <figure>, <footer>, <form>,
<h1>–<h6>, <header>, <hr>,
<iframe>,
, <legend>,
<nav>, <noscript>,
, <output>, <optgroup>, <option>,
<p>, <pre>,
<section>, <summary>,
<table>,

Inline tyyliset elementit

<a>, <area>,
<b>, <bdo>, <bdi>,
<cite>, <code>,
<dfn>, <del>,
,
<i>, <img>, <ins>,
<kbd>,
<label>,
<map>, <mark>,
<s>, <samp>, <small>, , , <sub>, <sup>,
<time>,

2.5.1 Ylätunniste <header>

<header> on uusi HMTL5:ssa ja se toimii kaikissa selaimissa. Sivun otsikkoalueen rajaamiseen käytetään <header> -elementtiä. <header> sisältää joukon sivustoa kuvaavia elementtejä (nimi, logo, kuva) sekä mahdollisesti navigaation. Elementti voi rajata sisälleen esimerkiksi otsikkoryhmän <hgroup> ja navigaatioelementin <nav>. Eräs huomionarvoinen seikka on, että <header> ei ole sama kuin <head>, joka on sivun perusrakenteeseen kuuluva elementti. Yleensä <header> vastaa internetsivuilta löytyvää ”yläbanneria”. <header> voi esiintyä html-sivulla monta kertaa. Esimerkiksi <article> -elementille voi asettaa otsikon <header> -elementillä. <header> ei voi sijaita <footer>-, <address>-, tai toisen <header> -tägin sisällä.

```
<header id="banner">
    <div id="logo">
        <img src="images/logo.png" width="250" alt="Sivuston Logo"  />
    </div>
    <div id="headline">OTSIKKO</div>
    <div id=headline2">ALIOTSIKKO</div>
    <nav>NAVIGAATIO</nav>
</header>
```

Ennen näki tämän nyt vain <header>.

2.5.2 Navigaatio <nav>

<nav> -elementissä esitetään sivuston päänavigaatio. Kaikkien sivun linkkien ei tarvitse olla <nav> -elementissä. Elementtiä voidaan käyttää useampaan kertaan sivun eri osioiden sisällä. Tällöin on kyseessä vain sen osion ”sisällysluettelo”, ei koko sivun. <nav> -elementin johdonmukainen käyttö mahdollistaa sen, että navigointiosat voidaan erottaa varsinaisesta sisällöstä. Liiallista <nav> -elementin käyttöä tulee välttää, sillä ei ole tarkoituksenmukaista merkitä kaikkia linkkejä vaan ne, jotka liittyvät sivun tai web-sovelluksen sisällä navigointiin.
Linkit voivat osoittaa muille sivuston sivuille, tai sivuun itseensä
• Päänavigaatio listamuodossa.
Nykyisin navigointilinkkien lista merkataan yleensä <ul> -elementiksi, mutta siellä voi olla muutakin, esimerkiksi useita <ul> -elementtejä, joita kutakin edeltää otsikko, tai vaikkapa kuva, jonka osat toimivat linkkien tavoin.

```
<nav>
    <ul>
        <li class="active"><a href="index.html">Etusivu</a>
        <li><a href="tuotteet.html">Tuotteet</a></li>
        <li><a href="jalleenmyyjat.html">Jälleenmyyjät</a></li>
    </ul>
</nav>
```

Elementin ideana on tehdä ero sivun navigointiosioiden ja muun sisällön välille. Tämä auttaa selaimia sivun esittämisessä. Esimerkiksi kannettavien laitteiden näytöillä joissa on yleensä vähän tilaa, selain voi esimerkiksi piilottaa <nav> -elementtien sisällön silloin kun käyttäjä ei halua navigoida sivulla.

2.5.3 Sivupalkki <aside>

<aside> -elementtiä käytetään merkitsemään reunahuomautusta tai oheistietoa. Tällaisia ovat esimerkiksi uutisartikkeliin liittyvä lisätieto, mainos, blogisivun osat, joissa on linkkejä esim. muihin blogeihin tai jotakin muuta pääsisältöä sivuava esitys. Usein on tarkoituksenmukaista asettaa pääsisällöstä erottuva tyyli <aside> -elementin sisällölle. Myös asettelulla johon elementin nimi viittaa saadaan <aside> -elementin sisältö erottumaan pääsisällöstä, <nav> -elementtien ryhmät.

<aside> -elementti kuvaa osan sivusta, jossa on välillisesti sivustoon liittyvää materiaalia.
• Tämän tyyppiset osiot esitetään usein sivupalkkeina painetussa materiaalissa.
• Esittää lohkoa joka voidaan esittää erotettuna pääsisällöstä.
Työssäni elementti <aside> käytin navigaattorina:

```
<aside>
    <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="HTML5_media.html">HTML5_media</a></li>
        <li><a href="Labyrint2D.html">Labyrint 2D</a></li>
        <li class="current">Labyrint 3D</li>
        <li><a href="Maps_APIs.html">Maps_APIs</a></li>
    </ul>
</aside>
```

2.5.4 Otsikkoelementit <h1>-<h6>

Teksteille, jotka loogisesti ovat otsikoita, kannattaa käyttää otsikkomerkkausta <h1>-<h6> -elementtien avulla, sitten, että <h1> tarkoittaa ylimmän tason (koko sivun) otsikkoa.

HTML5:ssä otsikon käsitettä määritellään tarkemmin kuvaamalla, miten määräytyy se sivun osa, jonka otsikosta on kyse. Esimerkiksi <h3> -elementti on sellaisen osan otsikko, joka ulottu siitä seuraavan <h3> -elementtiin tai ylemmän tason otsikkoelementtiin (<h2>,<h1>) asti.

h1 ensimmäisen tason otsikko (heading)
h2 toisen tason otsikko
h3 kolmannen tason otsikko p kappale (paragraph)

HTML5 CR:n mukaan elementtejä <h1>-<h6> ei saa käyttää alaotsikoille tai iskulauseille (subheadings, subtitles, alternative titles and taglines), ellei sellaisen ole tarkoitus olla uuden jako-osan otsikko.

Käytännössä otsikkoelementeillä on seuraavia vaikutuksia ulkoasumuotoilun lisäksi:

- Hakukoneet painottavat otsikoiden sisältöä suhteessa sivun sisältöön yleisesti; tämän yksityiskohdat ovat liikesalaisuuksia ja muuttuvat
- Selainten liitännäisillä ja apuohjelmilla voidaan tuottaa otsikoista sivun sisällysluettelo.
- Ruudunlukuohjelmissa yms. toimintatiloja, jossa luetaan vain otsikot, hypätään seuraavaan otsikkoon tms.

2.5.5 Sisältö <section> ja <article>

Leipäteksti ja siihen liittyvät kuvat sijoitetaan sektioiden ja artikkelien elementtien sisään. <section> määrittää alueen dokumentin sisällä. <article> yhtenäinen osa sivua, jossa on yleensä otsikko ja sisältö: esimerkiksi blogikirjoitus tai foorumiviesti. <article> määrittelee riippumattoman, itsenäisen sisällön.

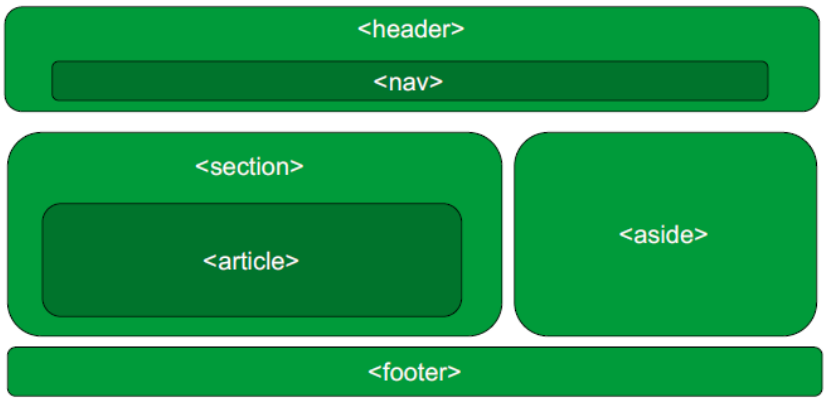
- Section-tagi yleisesti rakenteiselle sisällölle.

- Artikkeli-tagi on käyttökelpoinen esimerkiksi blogiteksteissä tai itsenäisissä artikkeleissa
  - Määrällisesti monta, jäsennettävää tietoa
  - Artikkeli voi sisältää omia <header> ja <footer> -elementtejä
- Mahdollista tehdä hierarkkisia rakenteita
  - Sektion alisektio
- Sijoiteltavissa ja jäsennettävissä vapaasti CSS:n avulla.

Sectionin ja articlen välillä on joskus vaikea valita. Sääntö on, että <article> on sellaista sisältöä, jota voisi käyttää muuallakin (julkaista uudestaan) kun <sectio> on enemmän vain kyseisellä sivulla esiintyvää sisältöä.

2.5.6 Alatunniste <footer>

Tähän tavallaan sijoitetaan
• Tietoa sivustosta
– Tekijät
– Tekijänoikeudet
– Linkki rekisteriselosteeseen
– Käyttöehdot
• Alatunniste on yleensä sama kaikilla sivuston sivuilla
– Yhtenäisyys
• Usein sivuston alalaidassa, mutta voidaan periaatteessa sijoittaa mihin tahansa.
– Joka sivulla samassa kohdassa
– Ei voi laittaa headerin tai toisen footerin sisälle.

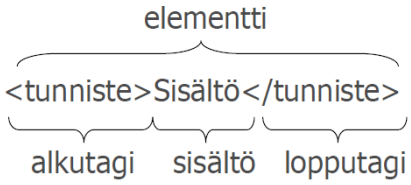


Kuva 2. Html:n sivun perus rakenne (Pinja Hokkanen 2016).

2.5.7 <ul>, <ol>,

 luetelma eli "järjestämätön" lista (unordered list)
 numeroitu eli "järjestetty" lista (ordered list)
 listan alkio (list item)

Kaikki nämä elementit ovat samaa muotoa, kuin edellä esitettiin: alkutagi, sisältö, lopputagi. Elementti alkaa alkutunnisteella eli tagilla, sitten tulee elementin sisältö ja lopuksi mahdollinen lopputunniste. Elementin määrittely kertoo millainen elementin sisältö voi olla, millaisia attribuutteja elementtiin voi liittää, mitä alielementtejä elementillä voi olla, pitääkö elementillä olla lopputagi jne.



Kuva 3. Elementti (Pinja Hokkanen 2016).

Näistä <ul> ja <ol> -elementit ovat sikäli muista poikkeavia, että niiden sisältönä ei ole tavallista tekstiä vaan sisältö koostuu <li> -elementeistä joiden sisällä sitten on tekstiä. Elementti voi olla myös tyhjä, ts. elementillä ei ole sisältöä eikä erillistä lopputagia. Lopputagimerkin voi halutessa kirjoittaa alkutagiin tunnisteen jälkeen, mutta se ei ole pakollista HTML5:ssa.

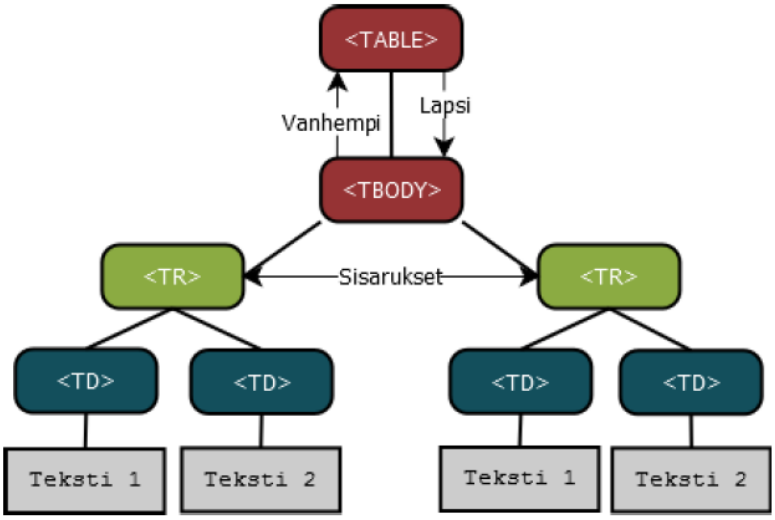
2.6 **DOM-puu**

HTML DOM eli Hypertext Markup Language Document Object Model (kirjaimellinen suomennos hypertekstimerkintäkielen dokumenttioliomalli) on w:ohjelmointirajapinta (Application Programmin Interface API), joka mahdollistaa HTML-tiedostojen sisällön, rakenteen ja tyylin dynaamisen muokkaamisen.

DOM kuvaa XML-dokumentin (XHTML on XML:ää, ja perinteinen HTML on tarpeeksi lähellä edellisiä, jotta sitä voidaan käsitellä samalla tavalla) elementtejä olioina, jotka muodostavat puurakenteen. Dokumenttioliomallin eli DOMin mukainen puurakenne koostuu solmuista (nodes), jotka vastaavat lähinnä elementtejä, ja niiden välisistä suhteista.

Yleisesti HTML-sivua vastaava DOM, ja siten itse sivun sellaisena kuin se on selaimen muistissa, on puurakenne, jossa on erityyppisiä solmuja: elementtisolmuja, tekstisolmuja, doctype-solmu ja eräitä muita. Muista tyypeistä tärkein on kommenttisolmu: kommentin sisältö tallentuu DOMiin ja on luettavissa JavaScriptillä.

Todellisuudessa puu on hiukan mutkikkaampi kuin selainten tutkintatoiminnot yleensä esittävät. Elementti ei varsinaisesti suoraan sisällä tekstiä, vaan puurakenteessa elementillä on lapsisolmu, joka on typiltään tekstisolmu.



Kuva 4. HTML-dokumentin taulukko-elementin puurakenne (DOM).

2.7 **Lomakkeet**

Tietojenkerääminen käyttäjiltä tapahtuu internetsivuilla yleensä lomakkeilla. HTML5 tuo uudistuksia lomakkeisiin liittyviin elementteihin. Tavoitteena on tarjota yksinkertainen ja yhtenäinen tapa luoda lomakkeita sekä niiden tarvitsemia toimintoja. Tietojen tarkistus ja erilaiset tiedon syöttämiseen tarkoitetut kentät ovat suurimmat HTML5:en tuomat muutokset lomakkeille.

Uudet, ajanilmauksiin liittyvät syötekenttien attribuutit (Date, time, datetime, month ja week); number-attribuutti; color; url… ovat <input> -elementin type-attribuutiksi annettavia arvoja. Niillä voidaan tarkemmin tyypittää tekstikenttään tulevaa tietoa, jolloin selain voi tarjota käyttäjälle tiedonsyöttämistä helpottavia ratkaisuja. Tällaisia ovat muun muassa kalenteri päivämäärän syöttämistä varten tai esimerkiksi numeronäppäimistö kosketusnäyttöpuhelimen näytölle, kun on kysymyksessä puhelinnumeron syöttäminen. Kirjoittamishetkellä vain osa selaimista tukee näitä uudistuksia ja selainten type-attribuuttien perusteella tarjoamien elementtien visuaalisen ilmeen muokkaaminen ei ole mahdollista (Korpela 2014, 431).

Lomakkeiden erityyppisille <input> -elementeille yhtenäisenä uutena ominaisuutena on placeholder-attribuutti. Sen arvoksi voi asettaa tekstin, joka väistyy elementin aktivoituessa. Tämän tarkoituksena on tuoda helppo tapa asettaa esimerkkiteksti joka häviää, kun käyttäjä alkaa täyttää esimerkiksi tekstikenttää. Selainten toteutukset ovat vaihtelevia ja vielä joudutaan asettamaan jokin vaihtoehtoinen vihje niille joiden selaimet eivät tunne vielä placeholder-attribuuttia (Korpela 2014, 445).

Input-elementeissä on useita muitakin määritteitä, jotka eivät ole välttämättömiä perustoiminnallisuuden kannalta, mutta parantavat käytettävyyttä. Esimerkiksi required-määrite tekee kontrollin pakolliseksi.

2.7.1 Lomakkeiden uudet piirteet, kuten uudentyyppiset tietokentät ja tiedon tarkistukset.

Sovellusvälimuisti (application cache), jolla voidaan ohjata selain lataamaan valmiiksi sovellukseen tarvittavat tiedostot offline-käyttöä varten.

- Rakenteiset elementit, joilla voidaan merkitä esimerkiksi navigointialue ja sivun jakautuminen osiin (header, nav, footer, aside, section...)
- Merkitysten kuvailun (semanttisen merkkauksen) keinot, kuten mikrotiedot ja mikromuodot.

3 WEBGL

HTML5-tekniikkaan sovelletaan useita eri teknologioita kuten canvas, SVG ja WebGL. WebGL on uusi tekniikka JavaScript API interaktiivisen 2D- ja 3D-grafiikan renderöintiin web-selaimella. Miksi pitäisi valita WebGL oman sovelluksen tai pelin graafisen ympäristön rakentamiseen? Eräs asiaan perehtynyt on Florian Boesch, joka blogikirjoituksessaan (Why you should use WebGL 2013) käy läpi WebGL-rajapinnan etuja. WebGL-rajapinnan kutsumiseen tarvitaan vähemmän työtä. Ei ole tarvetta tehdä tiedostoja, lisätä kirjastoja tai kutsua erikseen apuvälineitä. WebGL-rajapinnan kutsumiseen tarvitaan kolme riviä: <Canvas> -elementin luonti, sen lisääminen dokumenttiin ja kontekstin määritteleminen WebGL-rajapinnaksi (Boesch 2013.).

Esimerkki WebGL:n käyttö HTML-sivulla (WebGL Fundamentals 2015):

```
    <!-- vertex shader -->
    <script id="2d-vertex-shader" type="x-shader/x-vertex">
        attribute vec2 a_position;
        uniform vec2 u_resolution;
        void main() {
        …
        }
    </script>
    <!-- fragment shader -->
    <script    id="2d-fragment-shader"    type="x-shader/x-
fragment">
        precision mediump float;
        uniform vec4 u_color;

        void main() {
        …
        }
    </script>

        function getShader(gl, id) {
            var shaderScript = document.getElementById(id);
```

```
        …
            return shader;
        }

        function main() {
            // Get A WebGL context
            var canvas = document.getElementById("canvas");
            //var gl = canvas.getContext("webgl");
            var   gl   =   canvas.getContext("experimental-
webgl", { preserveDrawingBuffer: true });
            // setup GLSL program
            var   program   =   getProgram(gl,   "2d-vertex-
shader", "2d-fragment-shader")
        …
        }
```

WebGL on Khronos Groupin julkaisemaan OpenGL ES 2.0 -teknologiaan perustuva JavaScriptin kautta käytettävä ratkaisu 3D-grafiikan näyttämiseen suoraan selaimessa hyödyntäen HTML5:n <canvas> -elementtiä. WebGL on JavaScript API web-selaimelle ja yhteydessä tietokoneen grafiikkaprosessoriin renderöityessä HTML <canvas> -elementin sisällä (Kuryanovich 2016). WebGL toimii HTML5:n <canvas> -elementissä, ja se määrittää paikan, johon 3D-grafiikka piirretään. <Canvas> on HTML5:n uusi ominaisuus, joka mahdollistaa 3D-grafiikan piirtämisen suoraan selaimeen piirtokäskyjen avulla.

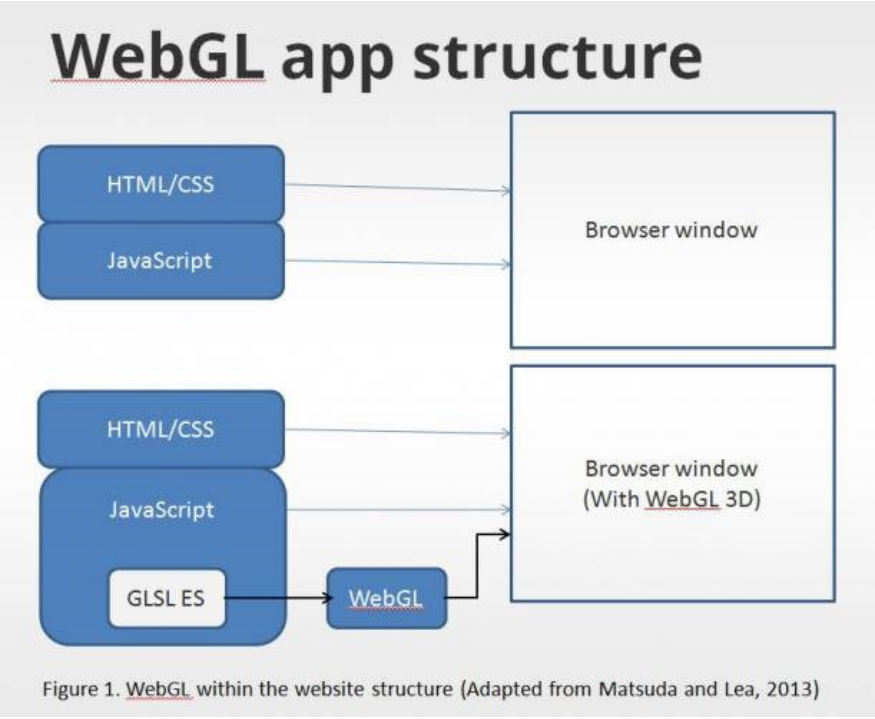
WebGL on alustariippumaton, ja siksi sen käyttö onnistuu eri päätelaitteilta. Katselemiseen tarvitaan ainoastaan WebGL:ää tukeva selain. Tällä hetkellä WebGL:lle on kuitenkin tuki kaikissa suosituimmissa selaimissa, esimerkiksi Google Chromessa, Mozilla Firefoxissa, Microsoft Internet Explorerissa ja Edgessä, Apple Safarissa ja Operassa. Mobiili Safari on yleisin mobiiliselain ja se on tukenut WebGL:ää iOS:n versiosta 8 lähtien. iOS ilmestyi jo 2014 eli toisin sanoen WebGL on nykyisin hyvin tuettu myös mobiililaitteissa, ellei sitten omista vanhoilla ohjelmistoilla varustettua laitetta, jossa todennäköisesti on jo muutenkin muita ongelmia. Vaikka WebGL ei ole virallisesti HTML5:n ominaisuus, siitä huolimatta se toimii kaikissa HTML5-kieltä tukevissa selaimissa.

Kuva 5. Kuvakaappaus WebGL-pelidemosta.

WebGL itsessään pystyy teknisestä näkökulmasta katsottuna toteuttamaan kaiken tarvittavan pelien ja visuaalisten sovellusten suorittamiseksi. Näitä ovat siis kuvasuhteet, eli resoluutiot ja skaalautuvuus, valaistukset ja varjostukset, vektoreiden, värien ja tekstuureiden piirto, fysiikat ja animaatiot sekä monia muita ominaisuuksia mitä peleissä useimmiten esiintyy (Danchilla 2012).

3D-grafiikka, jota WebGL piirtää, muodostuu kolmioista (triangle assembly). JavaScript luo vertekstitaulukot, jotka sisältävät tietoja jokaisista verteksistä, kuten niiden sijainnin 3D-maailmassa, tekstuurin, värin ja valaistuksen vaikutuksen. Tällaiset tiedot WebGL saa erikseen ladattavasta 3D-tiedostosta, tai ne voidaan luoda aivan alusta.

Useamman lähteen mukaan Three.js on vakuuttavin WebGL-ohjelmakirjasto 3D-pelille web-selaimeen. Noeticforce-nettisivulla kyseinen ohjelmakirjasto on ensimmäisten listalla (noeticsunil 2015.) Tietoperustan selvityksen myötä Three.js-17 ohjelmakirjastoon viittaavia kirjoja on kirjoitettu useampi, kuten esimerkiksi Jos Dirksenin LeaningThree.js: The JavaScript 3D library for WebGL sekä Isaac Sucinin Game Development with Three.js.



Kuva 6. WebGL app rakenne.

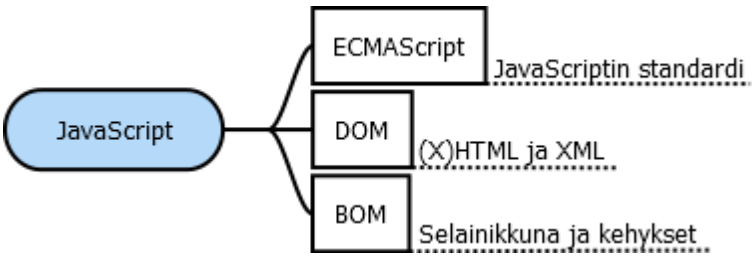
4 JAVASCRIPTIN PERUSTEITA

4.1 JavaScriptin rooli HTML:n yhteydessä

JavaScript on HTML:n yhteydessä pääasiassa selainskriptien tekemiseen käytetty kieli. Selainskripti eli lyhyesti skripti tarkoittaa ohjelmakoodia, jonka selain suorittaa HTML-sivun näyttämisen yhteydessä. Tällaisen selainohjelmoinnin (client-side scripting) lisäksi JavaScriptiä voidaan käyttää myös palvelimessa toimivana eli palvelinohjelmointiin (server-side scripting), mutta sitä ei käsitellä tässä työssä. JavaScriptin rooli HTML-sivulla vaihtelee suuresti sivun luonteen ja tarkoituksen mukaan (Korpela 2014, 55).

Silloin, kun JavaScriptiä käytetään apuvälineenä, sivu suunnitellaan sellaiseksi, että se on täysin toiminnallinen myös silloin, kun JavaScriptin suoritus on estetty selaimessa. tästä halutaan kuitenkin usein poiketa kertomalla käyttäjälle (noscript-elementin avulla), että tämän tulisi tai kannattaisi sallia JavaScript. Jos taas JavaScriptillä tehty osuus on aivan olennainen, ehkä niin, että koko sivu on luonteeltaan JavaScriptillä tehty sovellus, ei ”tunkeilematon JavaScript” tule kyseeseen.

Kuvassa 7 on esitelty kolme osaa, joista JavaScript muodostuu. Osat ovat ECMAScript, DOM ja BOM.
ECMAScript on JavaScriptin standardoitu versio ja tavaramerkki. Kielen on standardoinut ja rekisteröinyt eurooppalaisten tietokonevalmistajien liitto ECMA (European Computer Manufacturers Assosiation).
DOM (Document Object Model) on W3C:n määrittelemä ohjelmointirajapinta. DOM mahdollistaa (X)HTML- tai XML-dokumenttien sisällön lukemisen ja muokkauksen ajonaikaisesti. Yhdessä JavaScriptin kanssa sillä voidaan toteuttaa vuorovaikutteisia web-sivuja, jotka eivät vaadi jatkuvaa palvelinyhteyttä.
BOM (Browser Object Model) on pääasiassa vuorovaikutuksessa selainikkunan ja kehysten (frames) kanssa. Tavallisesti kaikki selaimet määrittelevät JavaScriptin laajennokset osana BOMia.



Kuva 7. JavaScriptin osat (Zakas 2012).

4.2 JavaScript ohjelmointikielenä

JavaScriptin käytön aloittaminen on erittäin helppoa, mutta kielen kokonaisuuden omaksuminen vie aikaa ja kielessä on joukko erikoisuuksia.

Vaikka JavaScript sinänsä ei ole mitenkään sidotuksissa HTML:ään, tässä työssä sitä käsitellään sellaisena, että sillä käsitellään HTML-sivua. Erityisesti, kun käytetään WebGL (Three.js) sokkelon rakentaessa.

4.3 JavaScriptin perusrakenteita

JavaScript koostuu muuttujista, operaattoreista (=, ==, <,>, &&, …), funktioista.
Ohjelman suorituksen aikana on usein tarve tallentaa erilaista tietoa muistiin. Esimerkiksi lomakkeen tekstikentän sisältö on saatava talteen jatkokäsittelyä varten. Tällaiset tiedot sijoitetaan ohjelmointikielissä muuttujiin, joihin viitataan yksilöllisellä nimellä. Jos taas on tallennettava useita toisiinsa liittyviä arvoja, voidaan käyttää taulukoita. Taulukossa saman nimen alle sijoitetaan useita arvoja, joihin viitataan indekseillä. JavaScriptissä uusi muuttuja luodaan komennolla var (tai ilman sitä) ja taulukko komennolla new Array() tai hakasulkujen [] avulla.

Tämän jälkeen, kun ECMAScript 2015 tuli voimaan, ei ole pakko käyttää komentoa var luodessaan muuttujia. Erityisesti JavaScriptin uusin variantti EcmaScript 6 sisältää huomattavan määrän funktionaalisten ohjelmointikielten ominaisuuksia.

4.3.1 Muuttujat

Muuttujalla ei ole tyyppiä, vaan muuttuja voi saada erityyppisiä arvoja, esimerkiksi `var lukumäärä = 0;` tai `var sana = "merkki";`
ECMAScript 2015 sisältää kaksi uutta tapaa ilmoittaa muuttujat: let and const ja ne soveltuvat paremmin muiden ohjelmointikielien kanssa.
”Let” toimii täsmälleen kuin var-muuttuja. Erona on, että let-määre voidaan jakaa uudelleen esim. silmukassa.

```
let apples = 5; // (*)

if (true) {
    let apples = 10;
    alert(apples); // tämä näyttää if lauseen sisällä
oleva let 10
}

alert(apples); // tämä antaa arvoksi 5
```

Const-määreellä esitellyn muuttujan sisältöä ei voi alustuksen jälkeen enää muuttaa.

```
const foo = 3; //foo is bound to the primitive 3.
foo = ["abc", "def"]; // TypeError exception!
```

Let- ja const-määreisiin liittyvä väliaikainen alue (engl. Temporal Dead Zone,TDZ) auttaa löytämään virheitä. Tällä tarkoitetaan ohjelmalohkon rivejä, jotka sijaitsevat ennen muuttujan let- tai const-määreellä.

Tietotyypit ovat: luku (sisältää sekä kokonais- että reaaliluvut), merkkijono (string), totuusarvo, määrittelemätön (undefined) ja oliot (objects). Esimerkkejä eri tyyppeihin kuuluvista vakioista: 42.1, ’huu’, true, undefined,{ a: 5, b: ’x’ },

Lauseet erotetaan toisistaan puolipisteellä, esimerkiksi ”x.moveTo(10, 50); x.lineTo(100, 100)”.

Sijoituslauseessa käytetään yhtäläisyysmerkkiä ”=”, esimerkiksi ”lukumäärä = 5”.

Yhtäsuuruusvertailu ilmaistaan kahdella tai kolmella yhtäläisyysmerkillä (jälkimmäinen tarkoittaa tyypin suhteen tiukkaa vertailua), esimerkiksi ”if(lukumäärä == 0)”.

Ehtolause on muotoa if(ehto) { lauseita } tai if(ehto) { lauseita } else { lauseita }. Aaltosulkeet saa jättää pois, jos niiden välissä on vain yksi lause. Esimerkki: ”if(i > 0) käsittele(i);”.

Toistolauseista tärkeimmät ovat tyyppiä while (ehto) { lauseita } ja for (alustus; ehto; lopputoimet) { lauseita }.
Esimerkki: for (var i = 0; i < s.lenght; i++) { muokkaa(s[i]) }. i++ tarkoittaa i:n arvon kasvattamista yhdellä.

4.3.2 Operaattorit

Ehdossa operaattori && tarkoittaa ’ja’, operaattori II ’tai’ ja lausekkeen edessä oleva operaattori ! (huutomerkki) kieltoa, ’ei’.
Esimerkki: ” if (i != 0 && i < 100) käsittele(i); ”.

Merkkijonovakiot kirjoitetaan yksinkertaisiin tai kaksinkertaisiin lainausmerkkeihin, esimerkiksi ’abc’ tai “abc” (ei merkityseroa).

Plusmerkki ”+” tarkoittaa merkkijonojen yhdistämistä silloin, kun sen operandit ovat merkkijonoja: ’ab’ + ’42’ on ’ab42’.
Yhdistetty operaattori ”+=”: a += b tarkoittaa samaa kuin a = a + b (paitsi että viittaus a:han käsitellään vain kerran). Muu esimerkki: a -= b (a = a - b), a *= b (a = a * b)

4.3.3 Funktiot

JavaScript-funktion määritteleminen.
Funktio pitää määritellä ennen funktion käyttämistä (kutsumista).

Funktio koostuu otsikkorivistä (ensimmäinen rivi) ja funktion rungosta. Otsikkorivillä on ensin funktion nimi ja sitten kaarisulkeet, joiden sisällä voi olla nollasta moneen parametria pilkulla toisistaan eroteltuna. Funktion runko alkaa { -merkillä ja päättyy } -merkkiin. Näiden aaltosulkeiden välissä voi olla yksi tai useampia lauserivejä (esimerkissä vain yksi rivi) (Korpela 2014, 57).

Funktion kutsussa on sulkeissa funktion argumentit, pilkulla erotettuina. HTML5:ssä määritellyt sovellusliittymät eli APIt ovat suurelta osin valmiiden funktioiden määrittelyjä, joiden yhteydessä kuvataan argumentit. Esim. x.moveTo(10, 50).

Funktion määrittely voi olla esimerkiksi seuraavanlainen:

```
function neliö(x) { return x * x }
```

Tunnuksissa kuten muuttujien ja funktioiden nimissä kirjaintaso on merkitsevä. Esimerkiksi x ja X ovat eri tunnuksia. Yleinen käytäntö on, että pitkissä tunnuksissa käytetään versaalikirjainta sananvälin sijasta (koska tunnus ei saa sisältää välilyöntiä) eli niin sanottua karavaanityyliä (camel case), esimerkiksi lukujenKeskiarvo ja getElementsByTagName.

Kommentti alkaa merkeillä // ja päättyy rivin loppuun.
Monta riviä ympäröidään näin /* … */.

ES6 tuo mukanaan uuden tavan määritellä funktioita joiden määrittelyssä on ensin mahdolliset parametrit, nuolimerkki (=>) ja lopuksi funktion sisältö. Parametrien ympäriltä voi jättää kaarisulkeet pois, jos parametreja on tasan yksi. Sisällön ympäriltä puolestaan voi jättää aaltosulkeet pois, mikäli funktion sisältönä on yksittäinen lauseke. Funktiot lyhentävät syntaksia ja selkeyttävät ohjelmakoodia.
Toisaalta pitkien funktioiden luettavuus saattaa heikentyä, mikä on puolestaan heikennys analysoitavuuden suhteen.

4.4 **Kirjastot**

Vaikka JavaScript on taipuisa ja muokattava ohjelmointikieli, sillä on muutama heikkous. Heikkouksiksi koetaan verkkoselainten pienet yksityiskohtaiset erot ja se, että monet ihmiset kokevat sillä ohjelmoinnin vaikeaksi. Sovelluksen optimointi eri verkkoselaimille voi vaatia paljon työtä. JavaScript-kirjastot ja automaatiotyökalut sisältävät erilaisia työkaluja ja kokoelman toimintoja kehityksen, ylläpidon ja suorituskyvyn nopeuttamiseksi. Lähes kaikki JavaScript-kirjastot on julkaistu copycenter- tai copyleft -lisenssillä. Ne mahdollistavat vapaan jakelun, käytön ja muokkauksen eikaupallisissa sovelluksissa. JavaScript-kirjastot toimivat hyvin websovelluksissa sovelluskehyksen tukena tai sellaisinaan (McFarland 2014).

Alaluvun tarkoitus on esitellä kaksi JavaScript-kirjastoa. JavaScript-kirjastojen ja automaatiotyökalujen valinta on vaikeaa, koska niitä on kymmeniä erilaisia. Monet valitsevat tarvittavat kirjastot ja työkalut tunnettavuuden, tarpeen, ominaisuuksien, dokumentaation tai vahvan yhteisön perusteella. Työn valitut pääasialliset kirjastot on esitelty alaluvuissa 4.4.1 ja 4.4.2.

4.4.1 Three.js

Three.js on apukirjasto grafiikan luomiseen WebGL-rajapinnalla. Tavallisen kuution luominen ilman apukirjastoja voi vaatia satoja rivejä JavaScript- ja varjostuskoodia, mutta apukirjaston avulla siihen tarvitaan vain muutama. Three.js:llä voi tehdä paljon muitakin asioita kuin vain pelejä. Three:n yhteisö on laaja ja siitä löytyy paljon demoja, opetusvideota, lähdekoodeja sekä kirjallisuutta.

Three.js kehitys alkoi GitHub-palvelussa itseään mrdoobiksi kutsuvan miehen toimesta vuonna 2010. Se on avoimen lähdekoodin kirjasto JavaScriptin kielellä, jonka kehittämisestä on tähän mennessä vastannut lähes 100 eri henkilöä. Three.js on keskittynyt täysin grafiikan luomiseen 3D-objekteista varjostimiin ja erilaisiin tehosteisiin. Siihen on kuulunut myös lisäosina esimerkiksi näppäimistön käsittelyä ja törmäyksen tunnistusta.

Lähteitä tutkiessa havaittiin, että Three.js ei sisältää pelimekanismiin liittyviä asioita, kuten pelimoottoria (engine) tai valmista näppäinkomentoja tai fysiikkaa (gravitaatio, kitkahäviö jne) ymmärtävä toteutusta.

Varsinaisen pelimoottorin sijasta Three.js on enemmänkin WebGL-tekniikkaa tukeva grafiikkakirjasto, jolloin varsinaiset pelimekaniikat on ohjelmoitava itse.
Täällä hetkellä on olemassa vastaava kirjasto nimeltä Tween.js, joka auttaa ratkaista ylemmäksi mainitut fysiikan ongelmat.

4.4.2 Modernizr-kirjasto

Modernizr on JavaScript-kirjasto, jolla voi testata selaimen tukea HTML5:n piirteille ja myös CSS3:n piirteille (Korpela 2014, 95). Sitä käytettäessä sivuntekijän ei tarvitse pohtia sitä, millainen testi on sopivin mikäkin piirteen osalta.

Kirjasto löytyy osoitteesta www.modernizr.com. Sitä ladataan samaan palvelimeen, jossa ovat omat sivut, nimellä modernizr.js. Sen jälkeen siihen voi viitata seuraavaan tapaan:
<script src=modernizr.js></script>
Tämä viittaa skriptiin, joka luo globaalin Modernizr-olion, joka sisältää tietoa HTML5-tuesta. Tämän jälkeen voi omassa JavaScript-koodissa testata esimerkiksi tukea <canvas> -elementille.

Modernizr-oliolla on seuraavat ominaisuudet, joiden arvot kertovat, onko selaimessa tuki vastaavalle HTML5:n piirteelle:

applicationcache (sovellusmuisti)
audio (<audio> -elementti), jolla on ominaisuudet ogg, mp3, wav ja m4a, joiden arvot kertovat tuesta vastaaville mediatyypeille
canvas (<canvas> -elementti) ja Canvas APIn pääosa)
canvastext (Canvas APIn tekstipiirtotoiminnot)
draganddrop (drag and drop eli viedä ja pudota)
geolocation (Geolocation API)
hashchange (hashchange-tapahtuma eli URLin fragmenttiosan muutos)
history (History API)
indexeddb (Indexed Database API)
localstorage (localStorage-sovellusvälimuisti)
postmessage (Messaging, dokumenttien välinen viestintä)
sessionstorage (sessionStorage-sovellusvälimuisti)
video (<video> -elementti), jolla on ominaisuudet ogg, webm ja h264, joiden arvot kertovat tuesta vastaaville mediatyypeille
websockets (Web Sockets API)
websqldatabase (Web SQL Database API)
webworkers (Web Workers, tausta-ajot).

Modernizr voi olla käytännöllinen ratkaisu etenkin silloin, kun on tarvetta testata monia selaintuen asioita.

5 CSS3

5.1 CSS:n peruskäsitteitä

CSS (Cascading Styling Sheets) on kieli verkkosivujen ulkoasun määrittelemiseksi. Sillä voi määritellä muun muassa värejä, sivujen taittoa ja fontteja. CSS:ää voi käyttää minkä tahansa XML-pohjaisen merkkikielen kanssa. CSS-tyylejä voi määritellä suoraan HTML-dokumentin sisällä tai vaihtoehtoisesti erillisessä tiedostossa, jota HTML-tiedosto kutsuu. (HTML & CSS 2016) CSS4 viimeinen versio kehittyy vuodelta 2011.

1990-luvun puolivälin paikkeilla Internetin suosio kasvoi räjähdysmäisesti. Tuolloin HTML oli ainoa tapa esitellä verkkosivua ja käyttäjät kaipasivat kipeästi parempaa kontrollia tyylien määrittelemiseen.
CSS-tyyliohje sisältää selaimelle annettavia ohjeita HTML-sivun tai XML-dokumenttien ulkoasusta. HTML:n tapauksessa selaimella on aina oletustyyliohje.

Perusteksti tyyli on leipätekstin kirjoittamista varten. Tyylit saat käyttöösi napsauttamalla haluttua tyyliä tyylivalikosta.

CSS-tyyliohje koostuu säännöistä (rules), jotka liittävät elementteihin ominaisuuksia (properties). Sääntö koostuu yhdestä tai useammasta selektorista (selectors) ja aaltosulkeissa olevista deklaraatiosta (declarations). Selektori ilmoittaa, mihin elementteihin deklaraatio koskee, ja se on yksinkertaisimmillaan vain elementin nimi. Kukin deklaraatio koostuu ominaisuuden nimestä, kaksoispisteestä ja ominaisuuden arvosta (Korpela 2014, 256).

Esimerkiksi seuraava sääntö asettaa body-elementin font-family -ominaisuudella arvon Cambria, mikä tarkoittaa, että sivun perusfontiksi asetetaan Cambria.

```
body {font-family: Cambria;}
```

Eri sektorit erotetaan toisistaan pilkuilla, eri deklaraatiot (css:n lausekkeet) taas puolipisteillä. Sääntö asettaa b- ja strong-elementeille keltainen taustaväri ja normaalin fontinvahvuuden. Jälkimmäinen kumoaa selaimen tyyliohjeen säännön, jonka mukaan näiden elementtien teksti on oletusarvoisesti lihavoitu

```
b, strong { font-weight: bold }
b, strong { background: yellow; font-weight: normal; }
```

Sivuun liitetyssä tyyliohjeessa esitetyt säännöt ajavat selaimen tyyliohjeen yli.

Tärkeä CSS-ominaisuus on display, joka määrittelee elementtien yleisen muotoilutavan. Display käytetään vain block (lohko) tyyliset elementit. Esimerkiksi `display: table-cell;` muotoilee elementin taulukon soluksi, `display: list-item;` listan alkioksi, `display: none;` estää elementin esittämisen kokonaan (missään näkyvässä muodossa) jne. Alkuasetus, jota sovelletaan, jos mikään tyyliohje ei muuta aseta, on `display: inline;`.

Se merkitsee, että elementti ladotaan kuten tekstiä kappaleen sisään: elementtiä peräkäin on niin paljon kuin samalle riville mahtuu.
Usein elementille asetettu oletusarvoksi `display: block;`. Se tarkoittaa esittämistä lohkona (block).
Tämä tarkoittaa, että ennen sitä ja sen jälkeen on rivinvaihto, selaimen asetetaan erikseen (margin-ominaisuudella)

5.1.1 Tyyliohje liittäminen HTML-sivuun.

CSS-tyyliohje voidaan liittää HTML-dokumenttiin link-elementillä, jossa on määrite tiedoston tyyli (rel="stylesheet"), tyyppi (type="text/css") ja sijainti (href="sijainti.css").

```
<link rel="stylesheet" href="css/styleWeb.css"
type="text/css" />
```

Vaihtoehtoisesti voidaan CSS-tyyliohje sisällyttää itse HTML-dokumenttiin. Tällöin kirjoitetaan style-elementin tai style-määritteen sisään sivun <head> -elementin sisään.

CSS3 tarjoaa käyttäjän laitteiston suorituskyvyn arvioimiseen näytön resoluution ja pikselitiheyden Media Query -ominaisuuksien avulla. Sen avulla voidaan käyttää kevyempää tyylitystä mobiililaitteille. Yleisen suorituskyvyn mittarina pikselitiheys ja näytön resoluutio on kuitenkin huono, koska sillä ei ole varsinaista yhteyttä laitteiston tehokkuuteen sivun esittämisessä: nykyaikainen mobiililaite voi olla tehokas, ja toisaalta vanha työpöytätietokone voi olla tehoton. CSS-tyyleistä erityisen raskaita ovat transform-animaatiot, border-radius -pyöristykset ja box-shadow -varjot.

6 PAIKKATIEDOT (GEOLOCATION). GOOGLE MAPS API

Paikkatiedot (geolocation), joiden avulla sisältö ja palvelut voidaan toteuttaa käyttäjän sijainnin mukaan.

Googlen karttapalvelussa maps.google.fi on toiminto, jolla voi tehdä omia linkkejä haluamiinsa paikkojen Googlen kartalla esitettynä. Linkkien lisäksi palvelusta saa myös valmiin koodin, jolla kartan voi upottaa sivuun <iframe> -elementillä. Koodissa oleva URL on hyvin pitkä, koska se suorittaa täsmällisen haun (mm. latitude ja longitude).

Upotettu kartta on googlen palvelun generoima HTML-sivu, joka toimii itsenäisesti omassa selailukontekstissaan. Siinä käyttäjä voi zoomata tai siirtää sen. Linkit avautuvat uuteen ikkunan tai välilehteen. Tämä riippuu siitä, miten sivu on tehty. <Iframe> -elementillä upotettava sivu voisi olla myös sellainen, että siinä olevat linkit avautuvat ”isäikkunaan” (target=_parent) eli siihen, jossa upottava sivu on.

6.1 **Paikannuksen hyödyt**

Käyttäjän fyysinen sijainnin selvittämisestä eli paikannuksesta (geolocation) käyttäjän sijainnin selvittämisestä on kahdentyypistä hyötyä:
Sijainnin perustella voidaan käyttäjälle esittää hänelle olennaista tietoa. Tarjouksia ja muuta sisältöä voidaan siis tehdä asiakaskohtaisemmiksi. Liikkeellä olevan käyttäjän sijainnin perusteella voidaan antaa esimerkiksi hyvinkin tarkkoja ajo-ohjeita, näyttää sijainti kartalla (Korpela 2014, 794).

6.2 **Geolocation – sovellusliittymä**

HTML5:n yhteydessä kehitelty sijaintitietojärjestelmä on kuvattu erillisessä W3C:n määrittelyssä Geolacation API Specification. Kyseessä on ohjelmointiliittymä, jota ei liity mitään uusia HTML-rakenteita.

Selaintuki on laaja, käytännössä on hyvä ehdolla if (navigator.geolacation) testata tuen olemassaolo ja antaa virheilmoitus tuen puutumisesta, jos se tekee sivusta hyödöttömän.

Paikannuksella saadut koordinaatit voidaan välittää Google Maps -palvelulle ja saada siitä kartta paikan ympäristöstä. Ei näytetä koordinaatteja käyttäjälle, vaan vältämme ne Google tarjoamaan JavaScript-kirjaston avulla käytettäviksi kartan luomisesta. Google Mapsin tuottama tulos näkyy oman sivumme sisältönä. (Korpela 2014, 795.)

Google Maps API

Tässä esimerkissä on luotu map-objekti `map = new google.maps.Map(document.getElementById('map-canvas'),` jolle on annettu omat koordinaatit, jotta kartalle voidaan lisätä karttamerkintöjä.

```
// This event listener will call addMarker() when the map is clicked.
map.addListener('click', function (event) {
    addMarker(event.latLng);
});
```

Luodaan kaksi eri funktiotta, jotta karttamerkintöjä voidaan lisätä ja poistaa kartalta:

```
// Sets the map on all markers in the array.
function setMapOnAll(map) {//2
    for (var i = 0; i < markers.length; i++) {
        markers[i].setMap(map);
    }
}
//Delete selected marker

function deleteMarker(id) {
    for (var i = 0; i < markers.length; i++) {
        if (markers[i].id == id) {
            markers[i].setMap(null);
        }
    }
```

Funktion InfoWindow:n avulla lisätään kaikille ”marker”:lle oma infoteksti.

```
var infowindow = new google.maps.InfoWindow({
    content: contentString
});
```

7 VIDEO

Videoiden esitys suoraan selaimessa osana sivua niin, että esitystä voi ohjata sivulla olevalla koodilla.

<video> ja <audio> -elementtien yhteiset IDL-määritteet.

<video> -elementti on tarkoitettu videon eli liikkuvan kuvan ja siihen mahdollisesti liittyvän ääneen esittämiseen. Se soveltuu myös tekstitetyn äänen esittämiseen, siis äänelle, johon liittyy ääneen synkronoituja kuvia, jossa on selitystekstejä.

Yksinkertaisimmillaan <video> -elementti voisi olla vain
`<video controls> <source src="abc.mp4" type="video/mp4"> </video>`
tai <video src="abc.mp4" type="video/mp4"> </video>.
Oletusarvoisesti videon kontrolleja ei ole eli videota ei voi käynnistää. Kontrollit saadaan mukaan controls-määritteellä.

Toinen vaihtoehto on, että kontrollit tehdään itse JavaScriptillä. Alkutilassa näkyy (kun selain on saanut videodatan) videon ensimmäinen ruutu, mutta haluttaessa voi erikseen asettaa poster-määritteellä sen tilalle muun kuvan, kuten videon kantta esittävän kuvan (Korpela 2014, 368).

Esitysasuun ei oletusarvoisesti tule reunaviivoja, selityksiä tms. Käytännössä videolle kannattaa yleensä kirjoittaa jonkinlainen selitysteksti kuten seuraavassa esimerkissä. Sen voi kirjoittaa <figcaption> -elementin sisään.

```
<figure>
    <video arc=orava.ogg width=300 height=400 controls
    poster=”orava.png“></video>
    <figcaption>Utelias harmaaorava.</figcaption>
</figure>
```

Videodatan lähteen ilmoittaminen. <video> -elementissä voi ilmoittaa videodatan lähteen joko elementin src-määritteellä tai sen sisällä olevissa <source> -elementeissä. Jälkimäisessä tapauksessa voidaan ilmoittaa useita lähteitä, jolloin ne ovat vaihtoehtoista: selain käyttää niistä ensimmäistä sellaista, jonka videodataan tyyppiä se tulee. Tämä on tärkeä siksi, että eri selaimet tukevat eri tyyppejä.

HTML5 toi yksinkertaisen ja kevyen tavan upottaa sivuille videoita. Nykyisin videot saadaan sivuille mutkattomasti. Esimerkin vuoksi videotiedosto ladataan omalta koneelta (HTML5_media.html):

```
<video width="400" controls>
    <source src="video/BigBuckBun.mp4" type="video/mp4">
    <source src="video/BigBuckBun.ogg" type="video/ogg">
    Your browser does not support HTML5 video.
</video>
```

Toisessa esimerkissä video ladataan JavaScriptin avulla youtube:sta käyttäen iframe API.

```
//3. this function creates an <iframe> and youtube player
//after the api code downloads
var player;
function onYouTubeIframeAPIReady() {
    player = new YT.Player('videoDiv', {
        height: '360',
        width: '640',
        videoId: ID_video,
        events: {
            'onReady': onPlayerReady,
            'onStateChange': onPlaerStateChange
        }
    })
}
```

Luodaan player-objekti, jolle annetaan oma korkeus ja leveys. Tälle esimerkille on luotu ul-lista, jossa voidaan vaihtaa videoitta [https://developers.google.com/youtube/iframe_api_reference].

7.1 **Videomuodot**

Nimi	Mediatyyppi	Tiedoston pääte
H.264 eli MPEG-4	video/mp4	.mp4 tai .mov
OGG	video/ogg	.ogg tai .ogx
WebM	video/webm	.webh
3GP	video/3gpp	.3gp tai .3gpp

Taulukko antaa karkean kuvan eri videomuotojen tuesta selaimessa (Korpela 2014, 371).

Jotta video-elementillä toteutettava esitys toimisi useimmissa selaimissa, pitää videodatan käytännössä olla tarjolla ainakin kahdessa eri muodossa, kuten H.264 ja OGG. Videomuotojen välisiin muunnoksiin on saatavilla ohjelmia, joiden laatu kuitenkin vaihtelee huomattavasti.

<Audio> ja <video> -elementin erityiset IDL- määritteet.

Ominaisuus	Tyyppi	Merkitys
addTextTrack	funktio	Lisää tekstiraidan
audioTracks	AudioTrackList	Saatavilla olevat ääniraidat
autoplay	totuusarvio	Käynnistyykö esitys automaattisesti
buffered	TimeRanges	Datan puskuroinnin tila
canPlayType	funktio	Testaa, onko argumenttina oleva mediatyyppi esitettävissä
controller	MediaController	Elementti, joka ohjaa median esitystä
controls	totuusarvo	Esittävätkö selain omat kontrollinsa
crossOrigin	merkkijono	CORS-asetukset
currentSrc	URL	Mediadatan nykyinen osoite; alussa tyhjä
currentTime	reaaliluku	Soittokohta sekunteina

defaultMuted	totuusarvo	Mykistyksen alkutila; vastaa sisältömääritettä muted
defaulPlaybackRate	reaaliluku	Toistonopeuden oletusarvo
duration	reaaliluku	Esityksen kesto sekunteina
ended	totuusarvo	Onko esitys päättynyt
error	MediaError	Virhekoodi
initialTime	reaaliluku	Aloituskohta sekunteina
load	funktio	Lataa mediadatan
loop	totuusarvo	Toistuuko esitys automaattisesti
mediaGroup	merkkijono	Mediaryhmä, johon elementti kuuluu
muted	totuusarvo	Onko äänimykistetty
networkState	kokonaisluku	Verkon tilaa kuvaava koodi
pause	funktio	Keskeyttää esityksen
paused	totuusarvo	Onko esitys pysähdyksissä
play	funktio	Käynnissä esityksen tai jatkaa sitä pysäytyskohdasta
playbackRate	reaaliluku	Toistonopeus
played	TimeRanges	Datasta esitetyt osat
preload	merkkijono	Vastaa preload-määritettä
readyState	kokonaisluku	Datan saatavuuden ilmoittava koodi
seekable	TimeRanges	Datasta haettavissa olevat osat
seeking	totuusarvo	Onko selain hakemassa kohtaa medi-asta
src	URL	Mediadatan osoite (URL)
startDate	Date	Alkukohdan aikasiirtymä (timeline offset)
textTracks	TextTrackList	Saatavilla olevat tekstiraidat
videoTracks	VideoTrackList	Saatavilla olevat videoraidat
volume	reaaliluku	Suhteellinen äänenvoimakuus 0…1

8 GRAAFINEN ESITYSALUE CANVAS – PIIRTOALUSTA

canvas-elementin tiivistelmä	
Merkitys	Luo piirtoalustan, johon voidaan piirtää JavaScriptillä Canvas SPI:n kautta.
Esimerkki	<canvas id=alusta></canvas>
Asema	HTML:n uutuus
Selaintuki	Usemmat selaimet, mutta IE vasta versiosta 9 alkaen (eikä välttämättä oikkutilassa), ja tuessa on aukioja. IE:n vanhemmille versioille saatavissa osittainen pakkelikoodi.
Ulkoasu	/*Laatikko, jossa piirtotoimintojen vaikutus näkyy. Oletuskoko 300x150 pikseliä*/
Missä	Siellä, missä fraasisisältö on salittu.
Sisältö	Läpinäkyvä sisältömalli: sisälösssä on sallittu ne elementit, jotka olisivat sallittuja canvas-elementin tilalla. Sisältö toimii varasisältönä, jonka selain selain esittää vain, jos sa ei tue canvas-elementtiä.
ARIA	Ei erityisiä sääntöjä.
Käyttö	Monenlaisten graafisten esitysten tuottamiseen
height	Elementin korkeus pikseleina (oletus 150)
width	Elemetin leveys pikseleina (oletus 300)

HTML5:n <Canvas> -elementti mahdollistaa dynaamisen, interaktiivisen 2D grafiikan ja animaation renderöinnin suoraan selaimessa (ilman multimedia plugineja).

- <Canvas> piirtotoiminnot
  - viivat, reunaviivat, täytteet
  - geometriset muodot
  - värit
- teksti <Canvas> -elementissä
- kuvat ja video
  - kuvien prosessointi
  - kuvaefektit
- animointi

8.1 <Canvas> -elementillä on lisäksi seuraavat metodit

getContext, joka luo piirtokontekstin (joka aina tarvitaan piirtämiseen)
toDataURL, jolla voidaan muuntaa piirros data-URLiksi
toBlob, jolla voidaan muuntaa piirros ”datamöhkäleeksi” eli BOB-olioksi.

Piirtoalustan luonti <canvas> -elementillä

<Canvas> -elementti on HTML5:n keskeisiä uutuuksia. Se tarjoaa ”piirtoalustan” graafisille esityksille, jotka voivat olla vuorovaikutteisia. Elementti itsessään on yksinkertainen, ja se luo pelkän piirtoalustan. Itse piirtämisen tehdään JavaScriptillä käyttäen tarkoitukseen tehtyä liittymää. Piirtämisen käsite on tässä laaja: se sisältää kuvioiden piirtämisen lisäksi mm. tekstin kirjoittamisen alustalle ja valmiiden kuvien sijoittamisen sinne.

Jotta <canvas> -elementtiin voi piirtää, siihen täytyy liittää piirtokonteksti eli lyhyesti konteksti, joka on CanvasRenderingContex2D- tai WebGLRenderingContext-olio.

Konteksti luodaan elementin getContex-metodilla, jonka ensimmäinen argumentti ilmoittaa kontekstin tyypin, ’2d’ tai ’webgl’. Ne viittaavat määrittelyihin HTML Canvas 2D Context ja WebGL Specification. Metodilla voi olla muitakin argumentteja, jos kontekstin tyyppiä vastaava API määrittelee niille merkitykset. (Korpela 2014, 395.)

Voi lisätä tekstin joukkoon pienen kuvana esitetyn symbolin. Elementin koon voi skaalata CSS:llä. Itse piirtäminen tehdään käyttäen pikselikoordinaatistoa, jonka elementin width- ja height-ominaisuus määräävät, mutta saman nimisillä CSS-ominaisuuksilla voi skaalata esimerkiksi fontin kokoon: `canvas { width: 1em; height: 1em; }`

<Canvas> -elementin toDataURL-metodi tuottaa elementin alueeseen piirretystä sisällöstä kuvan ja koodaa sen data-URLiksi, jonka metodi palauttaa arvonaan.
toDataUrl-metodin ensimmäinen argumentti on mediatyypin nimi. Oletusarvona on image/png (PNG-kuva), ja tätä arvo selain käyttää myös, jos argumentilla on muu arvo, mutta se ei viittaa selaimen tässä yhteydessä tuntemaan tyyppiin.

Esimerkiksi JavaScript-koodin loppuun voidaan kirjoittaa seuraavat lauseet, jotka lisäävät sivulle <img> -elementin, joka sisältää piirroksen PNG-kuvana. Kyseessä on tällöin täysin ”inline-kuva” eli <img> -elementin src-määrite sisältää itse kuvadatan koodatussa muodossa.

<Canvas> -elementin toBOB-metodi tuottaa piirroksesta tiedostomaisen Blob-olion, joka sisältää tehdyn piirroksen kuvana. Metodilla ei ole paluuarvoa, vaan tuloksen käsittely hoidetaan niin, että metodin ensimmäisenä argumenttina on takaisinkutsufunktio (callback functio) eli funktio, jota selain kutsuu asynkronisesti muodostettuaan Blob-olion. Funktio saa Blob-olion argumenttinaan. Toisena argumenttina toBlob-metodin kutsussa voi olla mediatyyppi. Sen oletusarvo on image/png.

9 WEB STORAGE

Web Storage API on käyttäjän selaimessa sijaitseva tietovarasto. Web Storagea käytetään evästeiden (cookies) tapaan käyttäjän henkilökohtaisen näkymän esittämiseen, mutta se on evästeitä kehittyneempi, turvallisempi ja nopeampi tekniikka. Web Storage tallentaa tietoja käyttäjän omalle tietokoneelle, eikä se kuormita sivustoa. Suorituskykyä parantaa

myös se, että tietoja ei lähetetä turhaan jokaisen palvelinpyynnön yhteydessä vaan ainoastaan niitä pyydettäessä (HTML5 Web 2017).

Web Storage sisältää kaksi oliota: localStoragen ja sessionStoragen. Istuntomuisti sessionStorage tallentaa tietoja vain istunnon ajaksi ja muisti tyhjennetään käyttäjän poistuessa sivulta. Paikallismuisti localStorage on tarkoitettu tiedon pitkäaikaiseen varastointiin, eikä muistille ole asetettu päättymisaikaa. Istuntomuisti sopii esimerkiksi säilyttämään käyttäjän lisäämiä tuotteita verkkokaupan ostoskorissa. Paikallismuistia voidaan käyttää hyödyksi esimerkiksi tallentamalla muistiin käyttäjän selaamia tuotteita ja tarjoamalla näitä kiinnostuksen kohteita myöhemmin (Zakas 2012).

10 VETÄMINEN JA PUDOTTAMINEN (DRAG AND DROP DND)

HTML5 tarjoaa mahdollisuuksia käyttää vetämistä ja pudottamista (drag and drop) sivuilla niin, että toiminnat liittyvät sivun kokonaisuuteen ja ovat skriptillä hallittavissa. Elementti voidaan määritellä vedettäväksi, jolloin se kokonaisuutena voidaan vettä haluttuun paikkaan ilman, että elementin sisältöä ensin valitaan esimerkiksi hiirellä maalaamalla. Näin voidaan toteuttaa graafisista käyttöliittymistä tuttuja toiminnollisuuksia.

HTML5:n vedä ja pudota -toiminnot poikkeavat monin tavoin edellä kuvatusta ”tavallisesta” toiminasta ja yleensä estävät tai sekoittavat sen. Siksi niiden käyttö kannattaa suunnitella huolellisesti. (Korpela 2014, 639.)

HTML5:n vedä ja pudota -toimintojen tuki selaimissa on laaja. Kosketusnäyttölaitteissa ei tukea yleensä vielä ole. Vedä ja pudota -toiminnot on suunniteltu käytettäviksi hiirellä tai muulla osoituslaitteella. Toiminnat voittaisin toteuttaa esimerkiksi puheohjauksella, jossa elementteihin viitataan nimillä.

Jos pudotusalueena on lomakkeen kenttä, voi siihen pudotettu sisältö tietenkin siirtyä lomakedatan mukana palvelimeen käsiteltäväksi. Pudotusalueesta voidaan sisältö kopioida lomakkeen piilokenttään, jolloin se myös saadaan palvelimen käsiteltäväksi. Lomakkeiden piilokentät (input type=hidden), periaatteessa toki muutkin kentät, joihin palvelin voi kirjoittaa tietoa, jonka selain lähettää eteenpäin lomakkeen lähetykseen yhteydessä.

Tietoa voidaan tallentaa selainmuistiin eli HTML5:n muistiin. Tämän ansiosta voitaisiin toteuttaa graafinen peli, jonka pelaamisen voi välillä keskeyttää. Selainmuisti eli HTML5:n muisti eli web-muisti tarkoittaa tiedon tallentamisen ja käyttämisen menetelmää, jossa HTML-sivu voi JavaScriptiä käyttäen tallentaa käyttäjän koneeseen tietoja. Evästeet (cookies) eli tiedot, jotka selain tallentaa käyttäjän koneeseen palvelimen pyynnöstä

ja antaa takaisin palvelimen pyynnöstä. Evästeitä voidaan luoda ja tutkia JavaScriptillä.

Verkkosivun osoitteeseen (URLiin), sen kyselyosaan (query part), sijoitetut tiedot. Ne saattavat johtua piilokentistä, mutta tekniikkaa voidaan käyttää myös itsenäisesti.

Vedä ja pudota -toimintojen koodaaminen HTML5:llä sisältää seuraavat asiat:

- elementtien merkitseminen vedettäviksi draggable-määritteellä
- ondragstart-tapahtumankäsittelijän liittäminen vedettäviin elementteihin niin, että vedettävä data saadaan käsiteltäväksi DatTrasfer-olioon
- elementin tai elementtien merkitseminen mahdolliseksi kohdealueeksi dropzone-määritteellä; käytännössä tällä on toistaiseksi vain dokumentoiva merkitys
- pudotuksen käsittely (esimerkiksi vedetyn datan lisääminen kohdealueeseen) ondrop-määritteellä.

11 3D DEMO-PELI SOKKELO

Opinnäytetyön suorittamista varten rakennettaan 3D Demo-pelin websivu, jossa on myös 2D-versio. Lisäksi opinnäytetyön teoriaosuudessa käydään läpi HTML5:n uudet ominaisuudet ja perustoiminnot.

Teoriaa varten tähän projektiin on luotu kaksi esimerkkiä siitä, miten HTML sivulle voidaan lisätä video, ja esimerkki Google Map API:sta.

Ennen kuin peliä voi alkaa ohjelmoida, täytyy tietenkin keksiä hyvä idea. Hyvän peli-idean tunnistaa siitä, että peliä saattaisi itsekin pelata mielellään. Toisaalta täytyy myös arvioida, onko pelin toteuttaminen mahdollista omilla taidoilla.

Kun idea sitten tulee, sitä kannattaa punnita jonkin aikaa mielessä ennen toimeen ryhtymistä. Sitä paitsi vasta pelin toteuttamisen aikana saattaa tulla vaikka mitä uusia ideoita.

11.1 Aloittaminen

Projekti aloitetaan luomalla projektikansio (tyo_P). Kansiossa olisi hyvä olla omat alikansiot CSS-tiedostoille, fonteille, kuville ja scripteille, jotta tiedostot saadaan pidettyä siististi järjestettynä.

Projekteissa on hyvä käyttää englanninkielisiä nimiä aina, kun mahdollista, jotta se olisi kansainvälisestikin ymmärrettävissä.

- tyo_P
  - CSS-kansion sisälle tulevat kaikki tyylimäärittelyt
  - Fonts-kansioon laitetaan kaikki projektin fontit, joita käytetään 3D-otsikon tekoon.
  - Textures-kansiossa säilytetään kaikkia kuvia, ja
  - Scripts(js) -kansioon laitetaan kaikki scriptit.

11.2 **Perusrunko**

Esimerkkisivun koodaus voidaan aloittaa luomalla uusi tyhjä HTML-tiedosto, jolle annetaan nimeksi labyrint.html.

Samalla luodaan uusi tyhjä CSS-tiedosto nimeltä ” styleWeb_3d (3D-otsikkoa varten) ja styleWeb” omia tyylejä varten ja sijoitetaan se CSS-kansioon. Kansioon js luodaan vastaavat skripti-tiedostot: maze2d.js, maze3d.js ja text3D.js sekä Three.js, Tween.js jne.

Tällöin koodista täytyy vain muokata CSS- ja script-tiedostojen tuontisijainnit vastaamaan projektikansion sijainteja.

Opinnäytetyön suorittamista varten rakennettaan 3D Demo-pelin websivu (labyrint.html) ja myös 2D-versio (labyrint2D.html).

Tähän projektiin on luotu teoriaa varten kaksi esimerkkiä (HTML5_media.html ja Maps_APIs.html) siitä, miten HTML sivulle voidaan lisätä videoita sekä Google Maps API:n malli.

Sivun koodaus aloitetaan kirjoittamalla HTML5:n-mukainen ”!doctypeja” perus HTML runko <html>,<head> {headin sisälle<title> tag} ja <body>. <head> -tagin sisälle määritellään metatageilla charsetiksi UTF-8, CSS- ja script-tiedostojen tuontisijainti.

Sivun <body> -elementtiin lisätään <header>,<footer> -tagit ja keskeiset <aside> ja <article> -tagit. Tämä on kuvattu projektin HTML-osassa.

11.3 **Työn kuvaus**

Päätyöksi luodaan kevyt 3D-sokkelopelin demo, joka koostuu eri komponenteista: seinät, pohjakenttä ja pelikuutio, jotka yhdistettiin yhdeksi kokonaisuudeksi eli sokkeloksi. Sen jälkeen ne aseteltiin paikoilleen.

Pelin prototyyppiä varten toteutin JavaScriptin Three.js-kirjaston avulla.

```
<script src="https://code.jquery.com/jquery.js"></script>
<script src="js/three/three.js"></script>
<script src="js/libs/TrackballControls.js"></script>
<script src="js/libs/tween.js"></script>
```

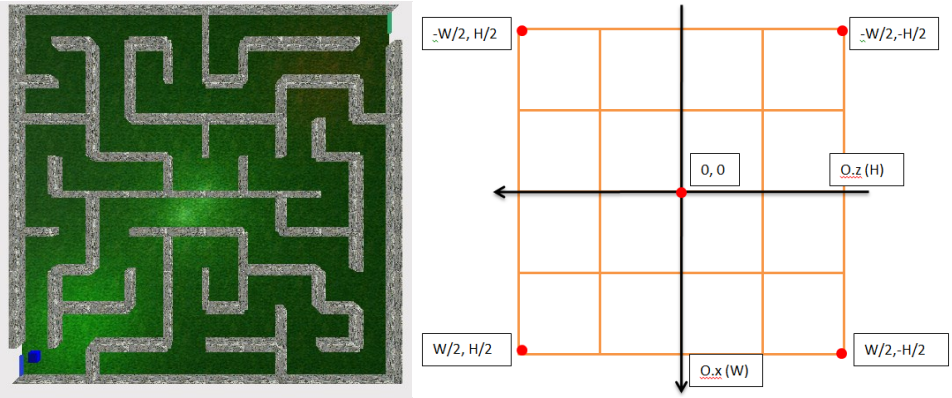
Yhdessä ne muodostavat kokonaisuuden, jossa pelaaja ohjaa pelihahmon (kuution) sokkelon läpi.
Tavoitteena on luoda yksinkertainen selaimessa toimiva peli, jota ohjataan näppäimistön avulla.

Tässä opinnäytetyössä käytetään Three.js-apukirjastoa. WebGL-sovelluksen luontiin Three.js-apukirjaston versio on R67. Testaus on suositeltavaa suorittaa samalla versiolla, koska avoimeen lähdekoodiin perustuvan Three.js-kirjaston olioiden ja funktioiden syntaksi voi vaihdella versioittain.

Three.js-sovellukselle voidaan tehdä yleinen pohja, josta on hyvä lähteä rakentamaan omaa sovellusta.

Tässä esitellään Three.js-sovelluksen pohja, Three.js-kirjaston 3D-objektit, materiaalit, scene, valot ja kamera. Lisäksi esitellään yksinkertaisen olion luonti JavaScriptilla sekä sen liikuttaminen näppäimistön avulla.

Varsinaisen pelimoottorin sijasta Three.js on enemmänkin WebGL-tekniikkaa tukeva grafiikkakirjasto, jolloin varsinaiset pelimekaniikat on ohjelmoitava itse. Tämä tuotti jonkin verran työtä, mutta kirjaston suosion ansiosta sille on saatavissa runsaasti erilaisia ohjeita ja vinkkejä.



Kuva 8. Kuvassa on sokkelon esimerkkinäkymä. Oikealla: pohjakentän koordinaatin sijainti, browserin ruudulla.

Pelaaja ohjaa pelikuutiota sokkelon läpi maaliin. Peli alkaa vasemmasta alakulmasta ja maali on oikeassa yläkulmassa. Objektin törmäyksen käsitellään Raycasting-tekniikan avulla. Aina, kun pelikuutio törmää seinään, se pysyy paikallaan. Raycasting-tekniikka on esitetty myöhemmin tarkemmin. Sokkelon näkymää voi zoomata hiiren rullan avulla. Sokkelon rakenne muuttuu, kun koko sivu päivittyy tai pelaaja saapuu maaliin ja tulee näkymään uusi ikkuna jossa sanotaan, että olet voittanut.

Työ luotiin Visual Studio 2015 ohjelmointiympäristöllä. Sokkelon koodimallina käytin c#:ssa tehtyä koodia, jonka käänsin JavaScriptin muotoon.

Se toimii WebGL:ää (Three.js) tukevissa verkkoselaimissa ja käyttöliittymissä. Liikkuvan objektin (pelikuutio) ohjaavia näppäimiä ovat nuolinäppäimet.

Perusajatuksena oli, että 3D-mallit pysyisivät melko yksinkertaisina, mutta kuitenkin riittävän monimuotoisina, jotta niillä saataisiin kokeiltua erilaisia tuettuja toimintoja ja mahdollisia ongelmia WebGL:n kanssa.

11.3.1 Sokkelon algoritmi

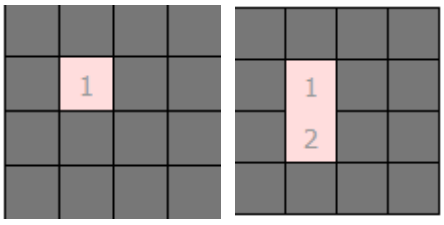
Algoritmi on tehty tiedostossa nimeltään ”maze3d.js”. Kuvaus toteutuksesta löytyy alta.

Ensin luodaan solujen taulukko [n x n], esimerkiksi

```
var width = 160;
  var cellsN = 10;
  //cells count by width (160/10=16px one cell)
  var cellWidth = width / cellsN;
  //solun koko   pikselina
  var wall_H = 40;// height of wall
 var cells = [cellsN, cellsN];
```

Kun ruudun taulukko on luotu, jokaisessa solussa kaikki neljää seinää on suljettu.

Algoritmin ydin on se, että lähtösolu valitaan sattumanvaraisesti function generateMaze(…) -funktion avulla. Lähtösolu laitetaan muistiin, ja sokkelon tiemuodon rakentaminen alkaa.



Koska lähtösolun, jolla operoidaan, seinät on suljettu, avataan yksi solun neljästä seinästä sattumanvaraisesti ja siirrytään viereiseen naapurisoluun. Siellä toistamme saman tapahtuman: avaamme yhden seinän kolmesta ja niin edelleen, kunnes ennemmin tai myöhemmin saavumme soluun [0,0] tai [n,n] tai umpikujaan.

Ohessa on function generateMaze(…):n osa, jonka avulla sokkelo rakennetaan.

```
var cell = new Cell(startX, startY, this);//ensimmäinen valittu random-solu
cell.visited = true;
var path = new Stack();//tiemuoto
while (path.length() > 0) {
    _cell = path.peek();
```

```
var nextStep = new List();//list not visited cells //
```

Tällä katsotaan viereisiä soluja (oikealla, vasemmalla, ylhäällä, alhaalla). Kun avaamaton solu löytyy, se lisätään ”path”-luetteloon, jotta seuraava naapurisolu voidaan valita.

```
if (_cell.x > 0) {//left of selected cell
            var x1 = _cell.x - 1;
            var y1 = _cell.y;
            xyI = numPiste(x1, y1);
            if (!cells[xyI].visited) {
                nextStep.append(cells[xyI]);
            }
        }
        if (_cell.x < nWidth - 1) {//right of selected cell
            x1 = _cell.x + 1;
            y1 = _cell.y;
            xyI = numPiste(x1, y1);

            if (!cells[xyI].visited) {
                nextStep.append(cells[xyI]);
            }

        }

        if (_cell.y > 0) { //top of selected cell
            x1 = _cell.x;
            y1 = _cell.y - 1;
            xyI = numPiste(x1, y1);

            if (!cells[xyI].visited) {
                nextStep.append(cells[xyI]);
            }

        }
        if (_cell.y < nHeight - 1) {//bottom of selected
cell
            x1 = _cell.x;
            y1 = _cell.y + 1;
            xyI = numPiste(x1, y1);

            if (!cells[xyI].visited) {
                nextStep.append(cells[xyI]);
            }
        }
```

Jos on olemassa naapurisoluja, jotka ei vielä ole käytetty, valitsemme seuraavan solun random-toiminnon avulla. Nyt selvitämme, mikä naapurisolun seinistä avataan (randomin avulla). [CellState[0]=-1 (closed); CellState[1]= 1 (open);]

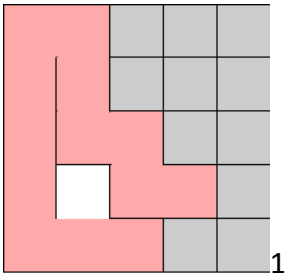
Jos naapurisolua ei löydy, tämä solu poistetaan ”path” -luettelosta (path.pop() -funktion avulla).

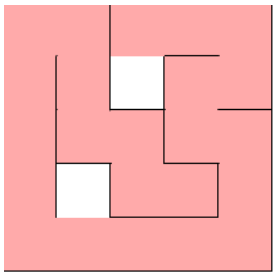
```
        var len = nextStep.List_length();

        if (nextStep.List_length() > 0) {
            var randcell = Math.random();
            var nextcell = Math.floor(randcell * len);
            var next = nextStep.listData[nextcell];
            // neighbor cell to open
            if (next.x != _cell.x) {
                if (_cell.x - next.x > 0) {
                    _cell.Left = CellState[1];
                    next.Right = CellState[1];
                }
                else {
                    _cell.Right = CellState[1];
                    next.Left = CellState[1];
                }
            }
            if (next.y != _cell.y) {
                if (_cell.y - next.y > 0) {
                    _cell.Top = CellState[1];
                    next.Bottom = CellState[1];
                }
                else {
                    _cell.Bottom = CellState[1];
                    next.Top = CellState[1];
                }
            }

            next.visited = true;//visited cell
            path.push(next);
        }
        else {
            path.pop();
        }
    }
```

Ohjelma yksinkertaisesti kierittää soluja, kunnes listan (eli path) kaikki solut on käytetty läpi. Sen jälkeen tämä ”path” vapautetaan ja jos on mahdollista, rakennetaan uusi. Tämä jatkuu kunnes, tämän toiminnan seurauksena muodostettu sokkelo.


1. path


2. path tyhjenys

Kuva 9. Kuvassa on esitetty path:n tyhjennyksen alku. Seinät ja aukiot on tehty.

Tässä käytin Stack() ja Lista() luokat. Kaikki käytetyt solut sijoitetaan Stackin avulla listaan.

```
function Stack() { /*from  http://www.java2s.com/ */
    this.dataStore = [];
    this.top = 0;
    this.push = push;
    this.pop = pop;
    this.peek = peek;
    this.length = length;
    this.clear = clear;
    //  }
    //push() can push a new element onto a stack.
    function push(element) {
        this.dataStore[this.top++] = element;
    }
    //The pop() function does the reverse of the push()
function.
    //It returns the element in the top position of the
stack and then decrements the top variable:
    function pop() {
        return this.dataStore[--this.top];
    }
    //The peek() function returns the top element of the
stack by accessing the element at
    //the top-1 position of the array:
    function peek() {
        return this.dataStore[this.top - 1];
    }
    //The length() function returns this value by returning
the value of top:
    function length() {
        return this.top;
    }
    //we can clear a stack by simply setting the top varia-
ble back to 0:
    function clear() {
        this.top = 0;
    }
};
//****** end  Stack() ******************************

//******  List ***********************************
```

```
function List() { /* www.java2s.com*/
    this.listSize = 0;
    this.pos = 0;
    this.listData = []; // initializes an empty array to
store list elements
    this.clear = clear;
    this.find = find;
    this.toString = toString;
    this.insert = insert;
    this.append = append;
    this.remove = remove;
    this.front = front;
    this.end = end;
    this.prev = prev;
    this.next = next;
    this.List_length = List_length;
    this.currentPosition = currentPosition;
    this.moveTo = moveTo;
    this.getElement = getElement;
    this.contains = contains;

    //Adding an Element to a List
    //appends a new element onto the list at the next
available position,
    //which will be equal to the value of the listSize var-
iable:
    //After the element is appended, listSize is increment-
ed by 1.
    function append(element) {
        this.listData[this.listSize++] = element;
    }
    //find() for finding the element to remove:
    //The find function simply iterates through listData
looking for the specified element.
    function find(element) {
        for (var i = 0; i < this.listData.length; ++i) {
            if (this.listData[i] == element) {
                return i;
            }
        }
        return -1;
    }
    //Removing an Element from a List we use the splice()
mutator function.
    //The remove() function uses the position returned by
find() to splice the listData
    //array at that place.
    //After the array is modified, listSize is decremented
by 1 to reflect
    //the new size of the list.
    //The function returns true if an element is removed,
and false otherwise. Here is the code:
    function remove(element) {
        var foundAt = this.find(element);
        if (foundAt > -1) {
```

```
            this.listData.splice(foundAt, 1);
            --this.listSize;
            return true;
        }
        return false;
    }
    //Determining the Number of Elements in a List
    //The length() function returns the number of elements
in a list:
    function List_length() {
        return this.listSize;
    }
    //Retrieving a List's Elements
    function toString() {
        return this.listData;
    }
    //Insert: Inserting an Element into a List
    function insert(element, after) {
        var insertPos = this.find(after);
        if (insertPos > -1) {
            this.listData.splice(insertPos + 1, 0, ele-
ment);
            ++this.listSize;
            return true;
        }
        return false;
    }
    //Clear: Removing All Elements from a List
    function clear() {
        delete this.listData;
        this.listData = [];
        this.listSize = this.pos = 0;
    }
    //Contains: Determining if a Given Value Is in a List
    function contains(element) {
        for (var i = 0; i < this.listData.length; ++i) {
            if (this.listData[i] == element) {
                return true;
            }
        }
        return false;
    }
    //Traversing a List
    function front() {
        this.pos = 0;
    }
    function end() {
        this.pos = this.listSize - 1;
    }
    function prev() {
        if (this.pos > 0) {
            --this.pos;
        }
    }
    function next() {
```

```
        if (this.pos < this.listSize - 1) {
            ++this.pos;
        }
    }
    function currentPosition() {
        return this.pos;
    }
    function moveTo(position) {
        this.pos = position;
    }
    function getElement() {
        return this.listData[this.pos];
    }
};
//****** end  List **********************************
```

Nyt kaikkiin ruudun soluihin on luotu seinät ja aukot ja voimme aloittaa kuvan piirtämisen.
Tähän tarvitaan Three.js:n ’scene’, johon luodaan wall:t, jotka esittävät sokkelon seiniä.
Tätä varten käytetään function drawMaze(scene) -funktiota, jonka alla on toinen funktio function drawLine(x1, y1, x2, y2, scene), joka piirtää seinät. Funktion argumentit ilmaisevat, mistä mihin piirrämme viivan (2D) tai seinän (3D) sokkeloon.

Ongelman ratkaisu – kahden seinän välinen kulma ei ollut täysin ehjä:

ennen korjausta 

korjauksen jälkeen 

```
if (cells[stI].Top == CellState[0]) { //top border line
          X1 = cellX * x - _Thick / 2;
          Y1 = cellY * y;
          X2 = cellX * x + cellX + _Thick / 2;
          Y2 = cellY * y;

          drawLine(X1, Y1, X2, Y2, scene);
      }
  }

  if (cells[stI].Bottom == CellState[0]) {
                        // bottom  border     line
          X1 = cellX * x - _Thick / 2;
          Y1 = cellY * y + cellY;
          X2 = cellX * x + cellX  + _Thick / 2;
          Y2 = cellY * y + cellY;

          drawLine(X1, Y1, X2, Y2, scene);
      }
  }
```

Koodista nähdään, miten korjaus on tehty.

Objekteille voidaan määritellä tekstuuriksi myös kuva. Ensin on varmistettava, että tekstuuriksi käytettävä kuva on neliö, jonka sivut ovat luvun kaksi potensseja. Muuten tuloksena voi olla tekstuureiden venymistä tai toistumisen katkeamista.

Yksinkertaisten objektien teksturointiin riittää normaali THREE.MeshBasicMaterial. Ensin kuvat ladataan three.js-rajapinnan kuvatyökaluilla, määritellään tekstuuri koko objektin kietovaksi ja lopuksi määritellään, kuinka monta kertaa se toistetaan objektissa.

```
    // create a box to represent the wall segment
    var wall_H = 20;// height of wall (if x2-x1=0 tai y2-
y1=0)
    var wallGeom = new THREE.BoxGeometry(wall_X, wall_H,
wall_Y);

    var texture =
THREE.ImageUtils.loadTexture("textures/wall.jpg");
    var wallMaterial = new THREE.MeshBasicMaterial({ map:
texture });
    // and create the complete wall segment
    var wallMesh = new THREE.Mesh(wallGeom, wallMaterial);
```

11.4 **labyrinth.html:n Init-funktion määrittelyt**

Pelissä käytin selaimessa ajettavaa WebGL-liitännäistä, joka alustetaan seuraavasti.
Annetaan sivulle `<div id="game"></div>` -elementti, joka toimii säiliönä (container), johon sokkelo sijoitetaan.

Pääfunktiona on `function init()`. Tällä määritellään sovelluksen asetukset. Three.js-sovellus tarvitsee vähintään kolme elementtiä: scene, camera ja renderer.

Nämä elementit ovat kaikki Three.js-kirjaston olioita, ja ne määritellään funktiossa init.

Scene on olio, jolle lisätään kaikki 3D-objektit.

Camera-olio on nimensä mukaisesti kamera, jonka kautta käyttäjä näkee scene-oliolle määritellyt objektit. Kameran näkyvyyden leveydeksi on asetettu 45 astetta, kuvasuhde leveydestä ja korkeudesta, lähin objekti 0,1 yksikön päähän ja kauimmainen objekti 1 000 yksikön päähän.
Kameran paikkaa on myös muutettu: korkeusakselilla eli y-akselilla 200 yksikköä ja syvyysakselilla eli z-akselilla 0 yksikköä taaksepäin. Leveysakselilla eli x-akselilla arvo on 0 ja tarkoittaa, että kamera näyttää nykyisen ylhäältä päin kuvatun näkymän.

Renderer-olio piirtää kaikki scene-olion koordinaatistolla olevat objektit näkyviin. Renderer luo määritellyn kokoisen canvas-elementin. Renderer-olion luoma canvas-elementti on erikseen määriteltävä kuuluvaksi HTML-dokumenttiin. Canvas voidaan myös liittää esimerkiksi erilliseen HTML-elementtiin, kuten div-elementtiin.

11.4.1 Init-funktion rakenne

Ensin määritän globaaliset muuttujat:

```
  // global variables
            var renderer;
            var scene;
            var camera;
            var control;
            var isTweening = false;
            var controls;

            var collidableMeshList = [];//A list of cells
in which the wall and there will be a collision (calls for
tween.js)

            var width = 160;  //150!!!
            var  cellsN  =  10;//cells  count  by  width
(160/8=20px one cell)
```

Luodaan liikkuva **pelikuutio**.

```
     function createCube() {
                var cellWidth = width / cellsN
                var boxEdge = cellWidth / 4;
                var        cubeGeometry        =        new
THREE.BoxGeometry(boxEdge, boxEdge, boxEdge);
                var        cubeMaterial        =        new
THREE.MeshPhongMaterial({  color:  0x0000ff,  metal:  true,
shininess: 10 });
                var  cube  =  new  THREE.Mesh(cubeGeometry,
cubeMaterial);
                cube.castShadow = true;
                cube.name = 'cube';
                //cube.position = new THREE.Vector3(width /
2 - 3, 1, width / 2 - 3);  !!!
                cube.position = new THREE.Vector3(width / 2
- cellWidth * 2 / 5, 2, width / 2 - cellWidth * 2 / 5);
                scene.add(cube);
                return cube;
            }
```

<renderer>, <scene>,< camera>.

Init()-funktiossa luodaan ensin **renderer**-objekti.

```
      // create a renderer, sets the background color and
the  size
      renderer = new THREE.WebGLRenderer();
      renderer.setClearColor(0xebe9e9, 0.5);//0xe5ffff
      renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
      renderer.shadowMapEnabled = true;
      renderer.antialias = true;
```

Nyt muuttujan renderer avulla voidaan luoda HTML5:n elementti, johon itse peli piirretään.

Sitten luodaan **scene**:

```
          // create a scene, that will hold all elements
such as objects, cameras and lights.
      scene = new THREE.Scene();
```

Pohjakenttä:

```
                //create the ground plane
                var planeGeometry = new
THREE.PlaneGeometry(width, width, 40, 40);
                var planeMaterial = new
THREE.MeshPhongMaterial({ color: 0xffffff });
                planeMaterial.map =
THREE.ImageUtils.loadTexture("textures/grass.jpg");
                planeMaterial.map.wrapS = planeMateri-
al.map.wrapT = THREE.RepeatWrapping;
                planeMaterial.map.repeat.set(4, 4);
                var plane = new THREE.Mesh(planeGeometry,
planeMaterial);
                // add the plane to the scene
                scene.add(plane);
```

Tämän jälkeen luodaan itse sokkelo:

```
                // generate a maze:  width x width pixcels;
cellsN x cellsN cells
                generateMaze(width, cellsN, cellsN);
                drawMaze(scene);
```

Peli tarvitsee myös kameran, johon asetin kolmiulotteisen perspektiivin.

```
                // create a camera, which defines where
                we're looking at.
                SCREEN_WIDTH = container.clientWidth;
                SCREEN_HEIGHT = container.clientHeight;
                var aspect = window.innerWidth / win-
dow.innerHeight;
                aspect = SCREEN_WIDTH / SCREEN_HEIGHT;
                camera = new THREE.PerspectiveCamera(45,
aspect, 0.1, 1000);
```

Tässä “aspect” on kuvasuhde, joka saadaan jakamalla containerin leveys sen pituudella. Lopuksi kamera täytyy asettaa oikeaan kohtaan. Tällä hetkellä kuvataan näkymä ylhäältä päin katsottuna.

Peli tarvitsi vielä kevyen valaistuksen, jonka avulla voidaan luoda varjot ja selkeä näkyvyys.

```
            // add spotlight for the shadows
    var spotLight = new THREE.SpotLight(0xff0000);
    scene.add(spotLight);
    var directionalLight = new THREE.DirectionalLight
         ({                color: 0xaaaaaa });
    directionalLight.castShadow = true;
    directionalLight.position.set(0, 50, 50);
    directionalLight.intensity = 0.6;
            scene.add(directionalLight);
```

Lopuksi kaikki lisätään containeriimme.

```
   container = document.getElementById("game");
    // add the output of the renderer to the html element
    container.appendChild(renderer.domElement);
```

Ohjaavina näppäiminä käytetään nuoli-näppäimiä, joille tehdään oma funktio.

```
          function setupKeyboardControls() {
                document.onkeydown = checkKey;
                function checkKey(e) {
                    e = e || window.event;
                    if (e.keyCode == '37') {
                        // left
                        takeStep-
Left(scene.getObjectByName('cube'), 0, 0.5 * Math.PI, 100);
                    }
                    if (e.keyCode == '38') {
                        // up

takeStepForward(scene.getObjectByName('cube'),   0,   0.5   *
Math.PI, 100);
                    }
                    if (e.keyCode == '39') {
                        // right

takeStepRight(scene.getObjectByName('cube'),   0,   0.5   *
Math.PI, 100);
                    }
                    else if (e.keyCode == '40') {
                        // down
                        takeStepBack-
ward(scene.getObjectByName('cube'), 0, 0.5 * Math.PI, 100);
                    }
```

```
        }
    }
```

Three.js ja sen 3D-objektit ovat käteviä funktioita olion liikuttamiseen ja animointiin. Nämä siirtävät objektia sen omassa koordinaatistossaan riippumatta ulkopuolisista kierroista. Olion liikuttamisfunktio siirtää kappaletta eteen, taakse, vasemmalle tai oikealle.

Pelikuution liikkuminen on kuvattu funktioissa:

```
function takeStepRight(cube, start, end, time){}
function takeStepLeft(cube, start, end, time){}
function takeStepBackward(cube, start, end, time){}
function takeStepForward(cube, start, end, time){}
```

Objektin törmäyksiä käsitellään Raycasting-tekniikan avulla. Raycasting on tietokonegrafiikassa käytetty menetelmä (tween.js), jolla voidaan tehdä laskelmia sädettä käyttäen. Raycasting-menetelmällä voidaan esimerkiksi laskea varjoja objekteihin käyttämällä valonlähdettä lähtöpisteenä tai mitata etäisyyksiä.

Se, mitä tapahtuu, kun pelikuutio törmää seinään, on kuvattu funktiolla function detectCollision(). Kun törmäyksen edellytykset THREE.Raycaster:lla täyttyvät, lasketaan kuution koordinaateista, missä solussa törmäys tapahtuu, ja sitten siirretään kuutio tämän solun keskelle. Tilanteessa, jossa kuutio törmää seinään, Raycastingiä käytetään muun muassa etäisyyksien mittaamiseen

Kun sokkelon raja on ylitetty viimeisessä solussa pelin lopussa (finish), siirretään uuteen peliin.

12 YHTEENVETO

Opinnäytetyön tavoitteena oli tutkia HTML5:n ja uusien web-teknologioiden mahdollisuuksia sekä toteuttaa kevyt 3D-sokkelopelin demo.

HTML5:n sisällön keskeisimmät asiat ja suunnitellun tavoitteet tulevat esiin tässä työssä.

Digitaalisten palveluiden tarjoajille HTML5-sovelukset tarjoavat olennaisia etuja. Palveluiden tarjoajien ongelmana on laitekannan jatkuva muuttuminen ja monimuotoistuminen. HTML5-sovelluksilla päästään paremmin hoitamaan käyttöliittymän ja käytettävyyden kysymyksiä. HTML5-sovelusten kehitystapa on luonnostaan sellainen, että se tuo käyttäjäkokemuksen mukaan alkuvaiheesta alkaen. Lisäksi käyttöliittymän muutta-

minen ja muuntelu on olennaisesti helpompaa, koska käyttöliittymä on toteutettu selkeän mallin pohjalta omana kokonaisuutenaan.

Tietotekniikan opiskelijoille ja harrastajille HTML5 tarjoaa kiinnostavia ja sopivan haastavia mahdollisuuksia.

HTML5-sovellusten perustekniikat HTML, CSS ja JavaScript tulevat säilymään, vaikka niissä varsinkin CSS:ssä tapahtuu kehitystä. Uudet mahdollisuudet eivät pakota luopumaan vanhoista, vaan selainmoottoreissa säilyy vanhentuneiksi julistettavien piirteiden tuki.
Mahdollisesti tullaan näkemään HTML5-sovelluksia joissa ei ole lainkaan HTML:ää. Toisaalta XML-tagien käyttö HTML:ssä on jo nyt mahdollista. CSS ja JavaScriptin rinnalle tuskin on lähivuosina tulossa uusia kieliä.

Tulevaisuuden ennustaminen on tunnetusti epävarmaa, mutta ainakin muutamia kehityssuuntia voidaan ennakoida. Puheen tunnistaminen ja syntetisointi ovat jo käytössä ja tekniikoiden parantuessa ne voivat muodostua hyvinkin tärkeiksi. Kehitys tulee siis luomaan uusia käyttöliittymien ja vuorovaikutuksen ongelmia ja mahdollisuuksia.
HTML5-sovellukset nousevat valtavirraksi. Vuorovaikutteisuutta sisältävät palvelut verkossa tullaan toteuttamaan HTML5-sovelluksina. Verkkopalveluja uusitaan eri syistä muutamaan vuodenvälein ja yhä useammin huomataan HTML5-sovellus parhaaksi vaihtoehdoksi.

12.1.1 Käyttöliittymänkehityksen haasteet

HTML5-sovelusten käyttöliittymäkehityksen keskeisiä ongelmia ovat seuraavat:

Yhteistyö toteuttajien, suunnittelijoiden, tuotteen omistajan ja asiakkaan kesken. Keskeistä on, että esille nousevat ongelmat voidaan käsitellä nopeasti ja että työn etenemistä arvioidaan riittävästi suhteessa tavoitteisiin.

Laitteisiin mukautumisen (responsive design) toteuttaminen niin, että eri vaihtoehdot otetaan huomioon riittävästi.

Työn rajaaminen niin, että saadaan toimiva hyvä tuote ajoissa, sen sijaan, että tavoitellaan täydellisyyttä.

Demo Peli

Pelin merkitys ei perustu vain viihteeseen, vaan peliä käytän oppiakseni uusia menetelmiä suunnittelussa ja kokeilussa.
Pelin suunnittelua varten luotiin tarkat ohjeet kohta kohdalta, siitä miten se luotiin ja mitä missäkin kohdassa tapahtui. Tarkoituksena oli antaa itselleni ja harrastajille runko ja esimerkki, joka opettaa, innostaa ja antaa kosketuksen konkreettiseen tuotokseen.

Usein sanotaan, että pelit on tehokkuussyistä toteutettava natiiveina sovelluksina. Totta on, että vahvasti grafiikkaa ja animaatiota käyttävissä peleissä tehokkuudella voi olla merkitystä. Suuri osa peleistä perustuu johonkin muuhun kuin raskaaseen grafiikkaan.

Peli voi olla luonteeltaan täysin paikallisesti käytettävä tai siihen voi liittyä tiedon haku verkosta tai vuorovaikutus verkon kautta. HTML5-sovellus ei aina ole parasta toteutustapa, mutta sitä kannattaa aina harkita ja tutkia.

Pelin kehitysprojekti innosti minua opiskelemaan three.js-kirjastoa. Sen avulla olen luonnut tämän 3D demo-pelin, joka sisältää <renderer>, <scene>,< camera>, valot sekä oliot visualisointia varten (pohjakenttä, pelikuutio, sokkelon seinät).

Olen kehittänyt kolme eri peliversiota.
Perusversio, jota käytän minun työssäni pääversiona. Tässä versiossa on käytetty valmis kontrolli TrackballControls.js (https://github.com/mrdoob/three.js). Kamera näyttää sokkeloa suoraan ylhäältä päin katsottuna. Ongelmana oli se, että pelikuutio jäi välillä jumiin osumalla seinään. Ratkaisu tähän löytyi Raycasting-tekniikan avulla. Raycasting on tietokonegrafiikassa käytetty menetelmä (tween.js), jolla voidaan tehdä laskelmia sädettä käyttäen.

Toinen versio.
Kamera seuraa pelikuutiota ja itse kameran ohjaus tapahtuu FirstPersonControls.js:n avulla. Tässä tapauksessa on käytetty nuoli- ja WASD-näppäimiä, jotka ohjaavat pelikuution liikkumista ja RF nostaa kameraa ylös-alas suunnassa. Ongelmana on, että kuutiota on vaikea navigoida. Liikunnan ohjaamisessa suunnat menevät välillä sekaisiin.

Kolmas versio.
Kolmas versio löytyy osoitteesta:
(http://rusfin.org/labyrinth/Labyrinth_3D_FPC_shooter.html)
Tässä ei ole kuutiota vain pelkkä kamera, joka on ensimmäisen persoonan näkökulmassa. Sokkelossa liikkuminen tapahtuu nuoli- ja WASD-näppäimien avulla, sekä hiiren näppäimillä. Kameran näkymä muutetaan liikuttamalla hiirtä. En pystynyt ratkaisemaan, miksi tämä versio toimii serverillä ja joillakin kannettavilla tietokoneilla, mutta ei pöytäkoneella.

Prosessina opinnäytetyön tekeminen oli haasteellinen ennen kaikkea sen aiherajausten suhteen. Laajan aihealueen läpikäynti ja olennaisimpien asioiden löytäminen olivat suurin osa opinnäytetyöprosessista. Ammatillisesta näkökulmasta asioiden selvittäminen ja tutkiminen toivat uusia näkökulmia selaimesta sovelluskehitysalustana.

Opinnäytetyö sisältää useita aihealueita, joista voisi aloittaa jatkotutkimuksen.

LÄHTEET

Aalto-yliopisto. Pinja Hokkanen. (2016). Verkkosivujen Rakenne (HTML). Perustuu Antti Tolppasen, Pia Tukkisen ja Hannu Järvisen kalvoihin. Haettu 04.03.2017 osoitteesta https://mycourses.aalto.fi/pluginfile.php/189950/mod_resource/content/1/HTML-luento.pdf

Boesch F. (2013). Why you should use WebGL. Blogijulkaisu 02.02.2013. Haettu 25.3.2017 osoitteesta http://codeflow.org/entries/2013/feb/02/why-you-should-use-webgl/

Buck J. (2011).The Buckblog, Mazes for programmers. Haettu 04.03.2017 osoitteesta http://weblog.jamisbuck.org/2011/1/27/maze-generation-growing-tree-algorithm

Danchilla,B. (2012). Beginning with WebGL for HTML5. Haettu 04.03.2017 osoitteesta http://www.books24x7.com/books24x7.asp

Dirksen J. (2014). Three.js Essentials. Chapter-03 Raycasting- tekniikka.

Heikniemi J. Mitä XML on? Haettu 25.3.2017 osoitteesta http://www.heikniemi.fi/kirj/moxml.html

java2s.com. Stake and list function haettu 04.03.2017 osoitteesta http://www.java2s.com/

Korpela J. (2014). HTML5 käsikirja. Jyväskylä: Docendo.

Korpela J. Web-julkaisemisen opas. (2009). (index.html; all.html) Haettu 04.03.2017 osoitteesta http://www.cs.tut.fi/~jkorpela/webjulk/

Korpela, J. (2011). HTML5 – uudet ominaisuudet. Jyväskylä: Docendo.

Lawson, Bruce & Sharp, Remy. (2012). Introducing HTML5 second edition. Berkeley.

McFarland, D. S. (2014). JavaScript & jQuery: The Missing Manual, 3rd Edition. Publisher: O'Reilly Media.

Simpson, K. (2015). You Don’t Know JS: ES6& Beyond. Sebastopol, California: O’ReillyMedia, Inc.

Three.js. Haettu 04.04.2017 osoitteesta https://threejs.org/

W3C. Haettu 04.03.2017 osoitteesta https://www.w3.org/standards/

W3C HTML5.1 Nightly. Haettu 04.03.2017 osoitteesta www.w3.org/TR/html51/

W3Schools. THE WORLD'S LARGEST WEB DEVELOPER SITE. Haettu 25.3.2017 osoitteesta https://www.w3schools.com/

WebGL Fundamentals. Haettu 14.4.2017 osoitteesta https://webglfundamentals.org/webgl/

WHATWG HTML Living Standard — Last Updated 2 May 2017. Haettu 3.5.2017 osoitteesta www.whatwg.org/html/

WHATWG Wiki. (2017). Haettu 13 March 2017 osoitteesta wiki.whatwg.org/wiki/FAQ

Zakas N. Fluent. (2012). Maintainable Javascript. Haettu osoitteesta http://www.tikalk.com/maintainable-javascript-nickolas-zakas-fluent-2012/