

Janne Törmänen

HYBRIDISOVELLUKSEN JA OPPIMISPELIN TOTEUTUS MOBIILIALUSTOILLE

Opinnäytetyö
Tietojenkäsittelyn koulutus

2018



**Kaakkois-Suomen
ammattikorkeakoulu**

Tekijä/Tekijät	Tutkinto	Aika
Janne Törmänen	Tradenomi (AMK)	Helmikuu 2018
Opinnäytetyön nimi Hybridisovelluksen ja oppimispelin toteutus mobiilialustoille		49 sivua
Toimeksiantaja Sanoma Pro		
Ohjaaja Janne Turunen		
Tiivistelmä <p>Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa sähköinen kasvistosovellus hybridi-mobiilisovelluksen muodossa. Opinnäytetyön toimeksiantajana oli Sanoma Pro. Toimeksiantaja määritteli sovellusta kohtaan etukäteen vaatimuksia ja kohdeyleisön.</p> <p>Sovelluksen kohdeyleisönä toimi alakoululaiset. Sovelluksen vaatimuksiin lukeutuivat esimerkiksi kasvien opiskelu sovelluksen kautta tekstien ja kuvien muodossa, kasvien hakeaminen niiden nimellä ja kasvien jaottelu luokka-asteiden mukaisesti. Sovelluksen vaatimukseen sisältyi myös pelillinen ominaisuus, jonka avulla sovelluksen käyttäjät voisivat testata kasveista oppimaansa tietoa.</p> <p>Opinnäytetyöraportissa käsitellään tekniikat, ohjelmistokehykset ja työkalut, joita sovelluksen ohjelmointiin käytettiin. Näihin lukeutuu muun muassa Apache Cordova, AngularJS ja Ionic Framework. Opinnäytetyössä tutkittiin myös pelillisyyttä, pelillistämistä ja oppimispelisiä. Pelillisuus aihe-alueessa selvitetään esimerkiksi mistä elementeistä peli koostuu ja miten käsitteet kuten pelillisuus tai pelillistäminen voisi määritellä. Aihe-alueessa myös sivutaan aiheita kuten pelillisyyden ja pelillistämisen hyödyntäminen ohjelmistokehittäjän näkökulmasta. Oppimispelisiä opinnäytetyö tutkii niin pedagogisessa kuin teknisessä merkityksessä.</p> <p>Teoriaosuuden jälkeen käydään läpi opinnäytetyöprosessin käytännön toteutusta eri vaiheet esitellen. Käytännön toteutuksen eri vaiheita ovat esimerkiksi rautalankamallit, haakuominaisuuden toteutus, layoutit, kasvientunnistuspelin ohjelmointi, responsiivisuus ja äänet. Opinnäytetyön lopussa arvioidaan sovelluksen lopputulosta ja käsitellään mahdollisia kehityskohtia. Lopussa käsitellään myös opinnäytetyön toimeksiantajalta saatua palautetta ja arvioidaan sitä.</p>		
Asiasanat mobiilisovellukset, pelillisuus, mobiilioppiminen, oppimispelit		

Author (authors)	Degree	Time
Janne Törmänen	Bachelor of Business Administration	February 2018
Thesis title		49 pages
Creating a hybrid application and learning game for mobile platforms		
Commissioned by		
Sanoma Pro		
Supervisor		
Janne Turunen		
Abstract		
<p>The objective of this thesis was to create and design plant identification application for mobile platforms. The aim was to create application in the form of hybrid application. The purpose of the plant identification application was to teach plant identification to its target audience, which was primary school students in this case. The requirements of the plant identification application included features such as: learning about different plants with photos and short text descriptions, browsing plants by grade and searching plants by name. The application was also required to have a game like element where the students could test their knowledge of what they had learned about plants.</p> <p>The thesis contained three major parts, as follows: techniques, gamification and practical execution of the application. The first part of the thesis introduced all major techniques that were used in creating the plant identification application. The techniques included essential mobile development software, libraries and frameworks such as Apache Cordova, AngularJS and Ionic Framework. The second part of the thesis included a deep dive to concepts such as gamification and gamifying. The second part also explored topics such as how software developers could benefit from using gamification in applications and how to define a term "videogame". The third part of the thesis contained information on how the application was created in practice. The third part was divided in to different stages of the application creation, which were named as follows: wire-frame models, the implementation of search feature, visual layouts, programming the plant identification game, responsive execution and game sound effects.</p> <p>The final part of the thesis reflected how the thesis process went overall and if the results matched the requirements set before starting the application creation. This part of the thesis also went through and evaluated feedback from the commissioner. The final part of the thesis also consisted of thoughts about what elements of the application could have been improved or made differently in the process.</p>		
Keywords		
mobile applications, gamification, mobile learning, learning games		

SISÄLLYS

1	JOHDANTO.....	5
2	TEKNIIKAT	6
2.1	Apache Cordova	6
2.2	AngularJS	8
2.3	Mobiilikehityksen frameworkit ja Ionic.....	11
3	PELILLISYYS	13
3.1	Mikä on peli	14
3.2	Pelillisuus ja pelillistaminen ohjelmistokehityksessä	16
3.3	Oppimispelit.....	17
4	KÄYTÄNNÖN TOTEUTUS	19
4.1	Lähtökohdat.....	20
4.2	Kasvienselailuosa	20
4.3	Kasvientunnistuspeli	27
5	PÄÄTÄNTÖ	46
	LÄHTEET.....	48

1 JOHDANTO

Toimeksiantajana tässä työssä toimii Sanoma Pro ja tarkemmin Sanoma Pron Mikkelin toimistolla työskentelevä Raija Komppa-Rannaste. Hän ehdotti tätä kyseistä aihetta työharjoittelujaksoni aikana ja päätin tarttua mahdollisuuteen.

Toimeksiantona oli toteuttaa alakoululaisille suunnattu kasvistomobiilisovellus. Sovelluksen päämääränä oli se, että alakoululaiset pystyisivät selaamaan kasveja luokka-asteen mukaisesti ja siirtyä harjoittelemaan lukemiaan tietoja kasvientunnistuspeliin. Pelissä on kolme tasoa, jotka skaalautuvat luokka-asteiden mukaisesti, pelissä on tunnistettavan kasvin kuva ja kirjalliset vaihtoehdot joista käyttäjä yrittää valita oikeaa. Kasvien selailupuolella oppilaat pystyisivät opiskelemaan tietoa kasveista kuvien ja lyhyehköjen tekstien avulla, sovelluksesta löytyy myös nimihaku, jonka avulla kasveja voi hakea kaikkien luokkien kesken.

Tämän opinnäytetyöraportin päämääränä on esitellä tekniikat, joita käytin toimeksiannon toteuttamiseen. Raportista selviää myös mitä ohjelmistoja hyödyntäen toimeksiannon mukainen sovellus syntyy. Opinnäytetyössä käydään myös läpi käytännön toteutuksen eri vaiheet sovelluksen osa-alue kerrallaan, aina sovelluksen aloituksesta julkaisuun asti.

Opinnäytetyön luvussa kaksi esitellään erilaiset tekniikat, joita kasvistosovelluksen luomiseen käytettiin. Näihin sisältyivät muun muassa Apache Cordova, AngularJS ja Ionic framework. Luvun eri alaluvuissa käydään esimerkein läpi, miten tekniikoita voi hyödyntää. Esitellyt tekniikat käsittävät mobiilisovelluksen kehitysalustan, JavaScript-frameworkin ja visuaalisen framework-kehiksen. Työssä käytettyjä ohjelmointikieliä, tekstieditoria tai kuvankäsittelyohjelmia ei ole perusteellisesti selitetty opinnäytetyöraportissa.

Opinnäytetyön luku kolme käsittelee pelillisyyttä ja siihen liitoksissa olevia lieveilmiöitä. Luvun ensimmäisessä alaluvussa käydään läpi käsitteen ”peli” määritelmä. Luvun toisessa alaluvussa syvennytään pelillisyyttä käsitteeseen ja tutkitaan miten ohjelmistokehittäjä voi hyödyntää pelillistämistä omissa sovelluksissaan. Luvun kolmas alaluku käsittelee oppimispelejä, jollainen kuului myös tämän työn toimeksiantoon. Oppimispelejä tarkastellaan niin tilastolli-

sesta kuin myös pedagogisesta näkökulmasta. Alaluvun lopussa sivutaan myös oppimispelien tulevaisuutta ja sitä ovatko tavalliset oppimismetodit ka-
toamassa.

Luvussa neljä käydään läpi toimeksiannon asettamat vaatimukset ja sovelluk-
sen toteuttaminen käytännössä. Luku neljä on jaettu kahteen osaan, jotka
ovat kasvienselailuosa ja kasvientunnistuspeli. Näiden kahden luvun alaluvut
käsittelevät kokonaisuuksia kuten esimerkiksi rautalankamallit, layoutit, ha-
kuominaisuus, pelin ohjelmointi, äänet ja responsiivisuus. Luku viisi on pää-
töntö, jossa käydään läpi toimeksiantajan palautetta sovelluksesta. Päättä-
nössä pohditaan myös kehityskohteita ja asioita mitä olisi voinut tehdä toisin.

2 TEKNIIKAT

Tässä luvussa esitellään tekniikat, joita käytettiin opinnäytetyön ohjelmoinnis-
sa ja sovelluksen kääntämisessä mobiililaitteille. Näihin tekniikoihin sisältyy
esimerkiksi Apache Cordova ja AngularJS. Vertailen myös erilaisia mobiilike-
hityksen framework -ohjelmistoja ja syvennyn tarkemmin niistä yhteen, Ionic
frameworkiin.

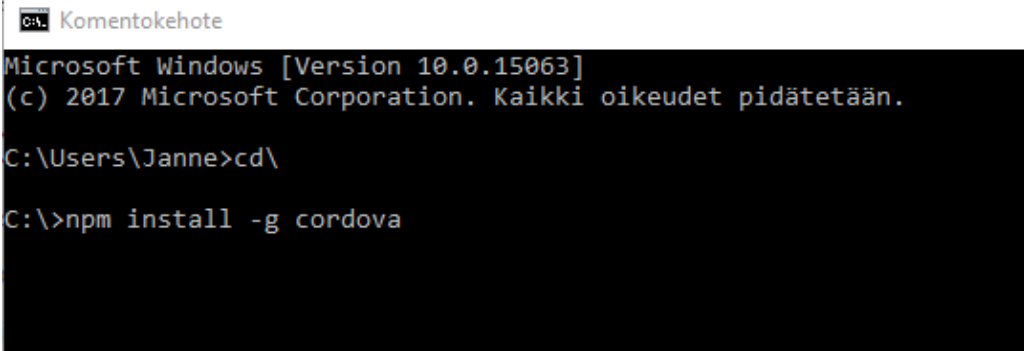
2.1 Apache Cordova

Apache Cordova on avoimeen lähdekoodiin perustuva mobiilikehitysympäris-
tö. Apache Cordova syntyi vuonna 2011 ja se luotiin toisen mobiilikehitysym-
päristön pohjalta, jonka nimi on PhoneGap. Apache Cordovasta on saatavilla
useita eri versioita, uusin versio on 7.1 (Apache Cordova 2017). Omassa
työssäni käytin versionumeroa 6.0.0.

Apache Cordova on suunnattu mobiilikehittäjille, jotka haluavat luoda sovel-
luksia useammille alustoille mahdollisimman helposti. Tämänkaltaisia sovel-
luksia kutsutaan yleisesti hybridisovelluksiksi. Apache Cordova mahdollistaa
tämän tunnettujen web-tekniikoiden avulla, joita ovat HTML5, CSS3 ja Ja-
vascript. (Cordova Documentation 2015c.)

Apache Cordova kehitysympäristön käyttöönotto

Ennen kuin Apache Cordova -kehitysympäristöä voi alkaa käyttämään, se pitää asentaa. Cordovan toiminta vaatii myös muiden ohjelmien asennusta ja käyttöönottoa. Yksi näistä ohjelmista on Node.js, joka on Javascript-kieltä hyödyntävä palvelinkehitysympäristö. Node.js tarjoaa npm-komennon, jota tarvitaan Cordovan CLI:n asennuksessa. Cordova asennetaan komentokehoteen (cmd) kautta (kuva 1). (Cordova Documentation 2015b.)



```

C:\> npm install -g cordova
  
```

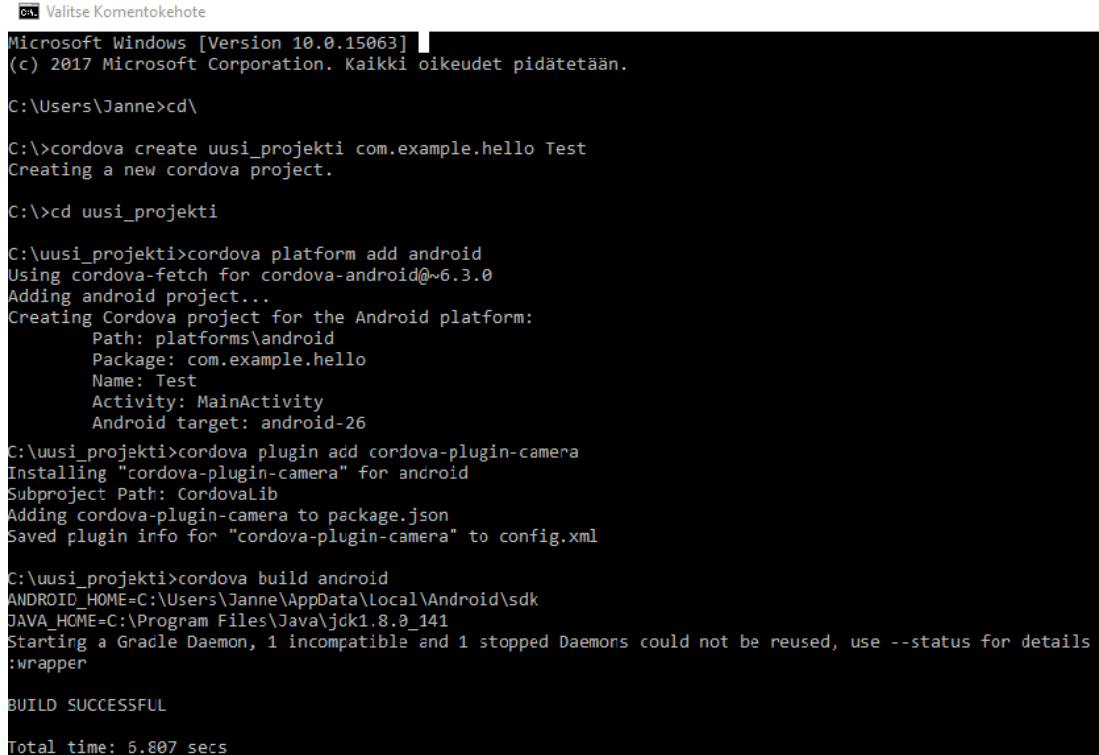
Kuva 1. Cordovan asennuskomento Windowsin komentokehoteen kautta

Cordovan asennuksen yhteydessä tarvitaan myös Android SDK Manager -niminen ohjelma, android-laitteille kehittäessä. Android on suosittu mobiilikäyttöjärjestelmä ja SDK lyhenne sanoista Software Development Kit. SDK on kehityspaketti tietylle alustalle, joka sisältää yleensä ohjelmointityökaluja, rajapintoja ja dokumentaatiota. Android SDK Manager vaatii toimiakseen JDK:n eli Java Development Kitin. Android SDK Managerin kautta voi ladata eri API eli rajapintatasoja, joiden avulla määrittyy mikä rajapintataso vastaa mitään Cordovan versiota ja vastaavaa Android versiota. Cordovan asennus vaatii myös järjestelmän ympäristömuuttujien polkujen asettamista. (Cordova Documentation 2015a.) (Kuva 2).

cordova-android Versio	Tuetut Android API-tasot	Vastaava Android-versio
6.xx	16 - 25	4.1 - 7.1.1
5.XX	14 - 23	4.0 - 6.0.1
4.1x	14 - 22	4.0 - 5.1
4.0.x	10 - 22	2.3.3 - 5.1
3.7X	10 - 21	2.3.3 - 5.0.2

Kuva 2. Taulukko yhteensopivista versiotasoista (Cordova Documentation 2015a)

Kuten myös asennus, muitakin Cordovan ominaisuuksia hallitaan käskylauseilla komentokehotteen kautta. Komentokehotteen kautta Cordovaan voi luoda esimerkiksi uusia projekteja, lisätä niihin alustoja ja luoda projektistasi koontiversion eli apk-tiedoston (kuva 3).



```

Valitse Komentokehote
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. Kaikki oikeudet pidätetään.

C:\Users\Janne>cd\

C:\>cordova create uusi_projekti com.example.hello Test
Creating a new cordova project.

C:\>cd uusi_projekti

C:\uusi_projekti>cordova platform add android
Using cordova-fetch for cordova-android@6.3.0
Adding android project...
Creating Cordova project for the Android platform:
  Path: platforms\android
  Package: com.example.hello
  Name: Test
  Activity: MainActivity
  Android target: android-26
C:\uusi_projekti>cordova plugin add cordova-plugin-camera
Installing "cordova-plugin-camera" for android
Subproject Path: CordovaLib
Adding cordova-plugin-camera to package.json
Saved plugin info for "cordova-plugin-camera" to config.xml

C:\uusi_projekti>cordova build android
ANDROID_HOME=C:\Users\Janne\AppData\Local\Android\sdk
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_141
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could not be reused, use --status for details
:wrapper

BUILD SUCCESSFUL

Total time: 5.807 secs
  
```

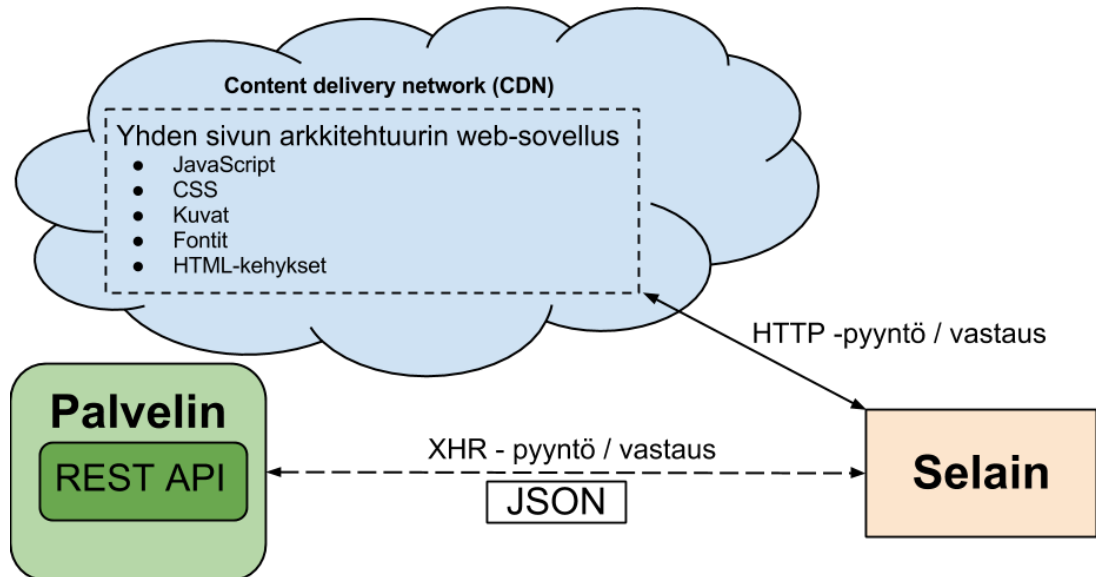
Kuva 3. Cordova projektin create ja build -komentojen demonstrointia.

Apk-tiedosto vaaditaan, että sovelluksen voi siirtää mobiililaitteeseen. Kun puhutaan Apache Cordovasta, yhtenä isona asiana esille nousee pluginit eli liitännäiset. Plugineilla voi lisätä sovellukseen ominaisuuksia, jotka voivat vaikuttaa esimerkiksi mobiililaitteen kameraan tai lisätä liikkeentunnistusominaisuuksia. Pluginit asennetaan myös käskylauseella komentokehotteen kautta (kuva 3).

2.2 AngularJS

AngularJS on vuonna 2012 julkaistu avoimeen lähdekoodiin perustuva front-end framework. AngularJS sai alkunsa, kun Googlella työskentelevä Misko Hevery alkoi kehittää sitä vuonna 2009. AngularJS on Googlen julkaisema ja tukema. AngularJS perustuu JavaScript-ohjelmointikieleen. Angularista on julkaistu kaksi erillistä versiota AngularJS ja Angular2, nämä versiot eroavat

toisistaan huomattavasti. Oman opinnäytetyöni käytännön osuudessa käytin AngularJS-versiota Angularista. Angulariin liittyvät vahvasti käsitteet kuten yhden sivun sovellus, direktiivit, lausekkeet, filtrit, moduulit ja kontrollerit. (AngularJS Documentation 2015.)



Kuva 4. Yhden sivun arkkitehtuurilla toteutettu web-sovellus havainnollistettuna kaavioon (Lyytinen 2013)

AngularJS tukee yhden sivun sovelluksia, englanniksi "Single Page Applications" (SPA). Yhden sivun sovelluksena tai yhden sivun arkkitehtuurina tunnetut tekniikat tarkoittavat teknisesti selitettynä sitä, että web-sovelluksen toiminnot ja logiikka siirretään asiakasselaimen vastuulle. Tämä mahdollistaa sen, että selaimen ei tarvitse ladata kaikkea uudelleen käyttäjän navigoidessa sovelluksessa tai sivustolla. Tämä taas tekee sovelluksesta nopeamman ja interaktiivisemman. Hyvä esimerkki yhden sivun arkkitehtuuria käyttävistä verkkosivuista ja palveluista on esimerkiksi Gmail-sähköpostipalvelu. Yhden sivun arkkitehtuuri hyödyttää myös sovelluksen yleistä tehokkuutta ja suorituskykyä koska sovelluspalvelin vapautuu kerrallaan palvelemaan vain tarvittavia datapyyntöjä. Tämä tarkoittaa yleensä JSON-muotoista kommunikointia, joka tapahtuu REST-rajapinnan kautta. Yhden sivun arkkitehtuuri on tehokas ja moderni tapa luoda sovelluksia nykypäivänä. (Lyytinen 2013.)

AngularJS-direktiivit ja lausekkeet

Direktiivit (engl. *Directives*) ovat tärkeä osa AngularJS:n toiminnallisuutta. AngularJS-kirjastoon kuuluu useita valmiita direktiivejä, joiden avulla pääsee hy-

vin alkuun. Direktiivejä voi kuitenkin joutua usein luomaan omia tai vähintään muokkaamaan valmiita direktiivejä tarpeidensa mukaiseksi saavuttaakseen optimaalisen lopputuloksen. Direktiivit ovat osa niin sanottua "Document Object Modelia", eli DOM-elementtiä. Direktiivit voisi selittää seuraavanlaisesti, ne luovat tietyn toiminnallisuuden sovellukseen ja "opettavat" tämän toiminnallisuuden sovelluksen HTML-osalle. (Panda 2014.)

Yleisimpiä AngularJS-direktiivejä ovat esimerkiksi ng-app, ng-model, ng-repeat, ng-hide ja ng-show. Näitä edellä mainittuja direktiivejä käytin myös itse ohjelmoidessani kasvisovellusta. Ohjelmointikoodissa AngularJS:n omat direktiivit tunnistavat siitä, että niiden edessä käytetään ng-etuliitettä. Direktiivit siis luovat sovellukseen jonkin toiminnallisuuden, esimerkiksi direktiivi nimeltään ng-repeat luo toistolauseen, ng-show näyttää tietyn elementin ja ng-hide piilottaa sen.

Lausekkeet (engl. *Expressions*) ovat toinen AngularJS:n keskeisistä osalueista. Lausekkeita voi yleensä AngularJS:n tapauksessa löytää direktiivien sisältä tai kirjoitettuna kaksoisaaltosulkujen sisälle. Lauseketta usein ympäröiviä kaksoisaaltosulkuja ei kuitenkaan pidä käsittää osana lauseketta vaan AngularJS-lausekkeeksi kutsutaan vain sitä osaa mikä löytyy sulkujen sisältä. Angularin lausekkeet ovat vähän kuin pieniä JavaScript-koodin osia, jotka laitetaan kuitenkin sovelluksen HTML-puolelle. Lausekkeet siis sisältävät toiminnallisuuksia eivätkä toimi pelkkänä tulostusmuotona. Tästä hyvänä esimerkkinä toimii esimerkiksi se, että jos Angularia käytettäessä kaksoisaaltosulkujen sisään kirjoittaa lausekkeen "2 + 2", sovellukseen ei tulostu "2 + 2" vaan tuon kyseisen laskutoimituksen vastaus eli numero neljä. (Marisi 2015.)

Yksi Angularin lausekkeiden ominaisuuksista on filttareiden eli suodattimien lisääminen lausekkeisiin. Suodattimet lisätään lausekkeen oikealle puolelle pystyviivamerkillä erotettuna. Suodattimia tai filttareita käytetään vaikuttamaan siihen, missä muodossa lauseke tulostuu sovelluksessa. Suodattimien avulla voi sovellukselle kertoa esimerkiksi, että haluaa sovelluksen käsittelevän tulostettavat numerot valuuttana tai esimerkiksi kahden desimaalin tarkkuudella (Marisi 2015). Itse käytin hyväkseni Angularin suodatusominaisuutta kasvisovelluksen hakuominaisuuden toteutuksessa.

2.3 Mobiilikehityksen frameworkit ja Ionic

Mobiilikehitys kannattaa aloittaa pohtimalla, millaista alustaa tai frameworkia kannattaa käyttää. Frameworkeja on monenlaisia, ne voisi jaotella sellaisiin alustoihin jotka kääntyvät natiivikomponenteiksi ja web-pohjaisiin mobiiliframeworkeihin tai mobiilialustoihin. Natiivikomponenteiksi kääntyvien ryhmään kuuluu esimerkiksi sellaisia frameworkeja kuten Xamarin, NativeScript ja React Native. Web-pohjaisista frameworkeista mainittakoon tunnetuimpina esimerkiksi Apache Cordova ja Ionic framework.

”Professional Mobile Application Development” -e-kirja jakaa eri kolmannen osapuolen frameworkit kolmeen ryhmään, jotka ovat tulkatut (*interpreted*), käännetyt (*translated*) ja verkko-frameworkit (*web*). Käännetyt-frameworkit käyttävät yhtä ohjelmointikieltä korvatakseen natiivikielen. Verkko-frameworkit eroavat muista esimerkiksi siinä, että ne käyttävät plugineita toimiakseen. (Gowell & McWherter 2012, 9-10.) E-kirja sisältää hyvää analyttistä tietoa mobiilisovelluksen kehittämisestä, mutta täytyy pitää mielessä, että kirja on julkaistu vuonna 2012. Tämän vuoksi kirjassa ei esimerkiksi voida vertailla suosittua vuonna 2013 julkaistua Ionic-frameworkia muihin frameworkeihin.

”How to choose the best mobile app development framework” -blogipostaus painottaa sitä, että frameworkin valitsemisessa kannattaa miettiä sovelluksen käyttämisen helpoutta ja nopeutta. Heard kirjoittaa blogissaan, että natiivikomponenteilla luodut sovellukset toimivat nopeammin kuin web-pohjaisilla frameworkeilla tehdyt. Hän myös kertoo, että mobiilisovelluksen olisi hyvä pyöriä 60 kuvaruutua (*framea*) sekunnissa, mutta web-pohjaisilla frameworkeilla luodut sovellukset eivät usein tähän yllä. Hitaudesta huolimatta Heard suosittelee käyttämään monia alustoja tukevia web-pohjaisia frameworkeja yli natiiviratkaisujen niiden helppouden takia. (Heard 2017.)



Kuva 5. Kaavio havainnollistaa, kuinka käyttötuntuma ja sovelluksen nopeus muuttuvat mennessä kauemmas natiivitoteutuksesta (mukaillen Heard 2017)

Eriäviäkin mielipiteitä toki löytyy, muun muassa yksi lähteenäni käyttämä suomenkielinen blogipostaus nimeltään ”Kuinka toteutustapa vaikuttaa mobiilisovelluksen käytön sujuvuuteen?”. Blogissa kerrotaan, että paras käyttötuntuma saadaan natiivitoteutuksella, mutta myös useat natiiviksi kääntyvät alustat kuten React Native, NativeScript ja Xamarin ovat päässeet suhteellisen lähelle natiivisovelluksen tuntumaa. Web-pohjaiset ja Webview-näkymiin pohjautuvat toteutukset eivät vedä vertoja natiivitoteutuksen käyttötuntumalle tai nopeudelle (kuva 5). Blogissa kuitenkin myös mainitaan, että web-pohjaiset toteutukset voivat olla itse ohjelmoijalle helpompia toteuttaa. (Ristola 2016.)

Ionic framework

Ionic on yksi esimerkki mobiilikehityksen hybridi-frameworkeista. Ionic tukee HTML5-tekniikoita ja sen avulla voi esimerkiksi luoda käyttöliittymän mobiilisovellukseen. Ionic on rakennettu AngularJS:n pohjalta ja niitä yleensä käytetään yhdessä. Toimiakseen kunnolla Ionic tarvitsee myös niin sanottua ”natiivia wrapperia” tai käärettä, jotta sovellus voisi toimia omana sovelluksenaan. Tämä wrapper on Apache Cordova. Yhdistettynä nämä kolme komponenttia eli Apache Cordova, AngularJS ja Ionic framework muodostavat tehokkaan mobiilikehityspaketin. Tässä paketissa Ionicin rooli on tarjota mobiilisovellukselle, se mitä sovelluksen käyttäjä voi itse nähdä ja tuntea eli käyttöliittymän. Ionicin avulla pystyy luomaan esimerkiksi responsiivisia valikkoja, painikkeita ja esimerkiksi animaatioita. (Ionic Documentation 2016.)

Ioniciista on olemassa tällä hetkellä kolme versiota, Ionic v1, Ionic v2 ja uudempana Ionic v3. Ionic v1 on huomattavasti erilainen kuin Ionic v2. Mainitsin aikaisemmin, että Ionic on rakennettu Angularin pohjalta. Angular on myös syy siihen miksi Ionic v1 ja Ionic v2 poikkeavat toisistaan niin paljon. Ionic v1 on nimittäin rakennettu AngularJS:n pohjalta ja Ionic v2 taas uudemman Angular2:n pohjalta. Ionic v3 on käytännössä samalla pohjalla toimiva päivitys Ionic v2 malliin. (Renaux 2017.) Tässä tekstissä puhuessani Ionicista ilman mitään loppupäätettä tarkoitan nimenomaan Ionic v1 -versiota, jota käytin myös omassa työssäni.

Ionic sisältää valikoiman erilaisia valmiiksi tyyliteltyjä CSS-komponentteja, kuten esimerkiksi painikkeita, otsikkopalkkeja, lomakkeita ja listoja. Näiden komponenttien avulla pystyy luomaan siistin ja responsiivisen ulkoasun mobiilisovellukselle. Ionicin CSS-komponentit ovat lähinnä front-end -visuaalisia elementtejä, jotka eivät vaadi välttämättä JavaScript-koodia. Ionicin tarjoamia valmiita CSS-komponentteja voi myös itse muokata CSS- tai SASS-tiedoston kautta. Ionicista löytyy myös AngularJS-laajennuksia, jotka yhdistävät Ionicin tarjoamat komponentit Javascript-koodiin. Tällä tavalla Ionicin komponentteihin saa lisättyä vaativampia toiminnallisuuksia. Laajennukset määritellään yleensä niin sanotun kontrollerin (engl. *controller*) kautta. Kontrolleri sisältää Javascript-funktioita, jotka lisäävät toiminnallisuutta käyttöliittymässä näkyviin elementteihin. Ionicin AngularJS -laajennuksiin sisältyy esimerkiksi valikot, pop-up -ikkunat ja erilaiset näkymät. (Ionic Documentation 2016.)

3 PELILLISYYS

Tässä luvussa tutkin ja määrittelen pelillisyyden käsitettä ja asioita mitä se pitää sisällään. Näitä ovat esimerkiksi se, mikä tekee pelistä pelin ja mitä on pelillisyyden. Sivuan myös asioita, kuten oppiminen peleissä ja sitä miten pelillisyyden voidaan yhdistää oppimiseen. Koen myös tärkeänä mainita, että kun puhun tässä luvussa peleistä ja pelillistämisestä, tarkoitan nimenomaan sähköisiä pelejä, enkä esimerkiksi lauta- tai pihapelejä.

3.1 Mikä on peli

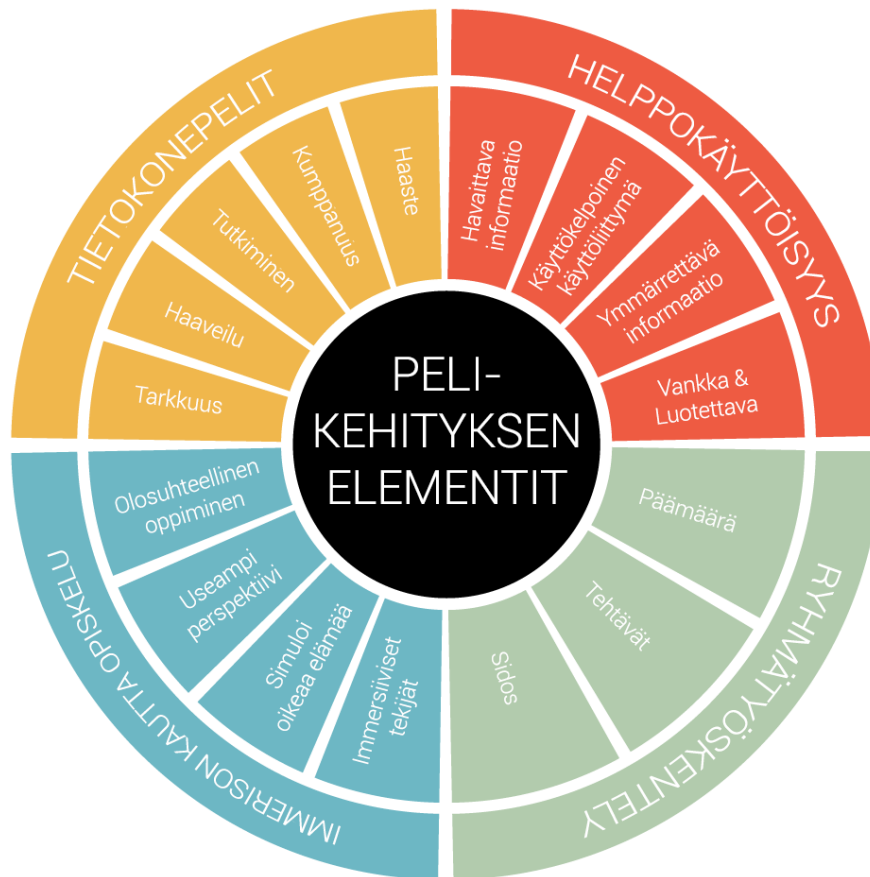
Ennen kuin siirrymme analysoimaan pelillisyyttä tarkemmin, on syytä selvittää miten pelin voisi ylipäättään määritellä, näin saamme helpommin myös määritettyä pelillisyyden käsitteen sisällön myöhemmin. Esimerkiksi yksi pelisuunnittelusta kertova artikkeli lähtee määrittelemään peliä pohtimalla, mitä yhdistäviä tekijöitä lähes kaikista peleistä voi löytää. Näitä yhdistäviä tekijöitä ovat muun muassa säännöt, peliympäristö, päämäärä, kilpailu, aktiivisuus ja palaute.

Säännöt ja peliympäristön voisi niputtaa oikeastaan samaan kategoriaan, koska peliympäristö määrittää säännöt. Peliympäristöä voisi ajatella aidattuna alueena, jonka sisälle säännöt määritellään. Säännöt taas määrittelevät miten pitää edetä saavuttaakseen päämäärän. Päämäärä on jokin ehto, jonka avulla peli ratkeaa. Päämäärä ei aina ole välttämättä positiivinen, sillä sen saavuttaminen voi myös tarkoittaa pelin häviämistä. Tärkeintä kuitenkin on, että pelissä on jokin päämäärä, alku ja loppu. Päämäärä tai pelin tavoite ei myöskään välttämättä aina ole ennalta määriteltä. Pelaaja itse voi myös määritellä päämäärän, esimerkiksi vaikka yrittämällä saada enemmän pisteitä kuin viime pelikerralla. (Mononen 2003.)

Tästä pääsemmekin siirtymään toiseen pelin määrittelevään termiin eli kilpailuun. Yleensä kaikista peleistä löytyy jonkinlainen kilpailullinen elementti, vastustaja vain muuttuu pelistä riippuen. Vastustaja voi olla esimerkiksi toinen pelaaja, tekoälypelaaja, pelin asettamat rajoitukset tai pelaaja itse. Kilpailun olemassaolo ei kuitenkaan tarkoita sitä, että peleissä pitäisi olla selvä häviö tai voittaja. Pelit vaativat myös yleensä jonkinlaista aktiivisuutta, aktiivisuus voi esiintyä motorisena liikkeenä, esimerkiksi napin painalluksena. Aktiivisuutta voi myös esiintyä henkiselä tasolla, henkisen tason aktiivisuudella tarkoitetaan esimerkiksi päätöksentekoa, keskittymistä tai muistamista. (Mononen 2003.)

Aktiivisuus on myös suuri osatekijä siihen miksi pelit ja pelinomaisuus on helppo yhdistää oppimiseen. Aktiivisuus saa aikaan sen, että jokainen pelikerta opettaa pelaajalle jotain. Se on pelaamisen sivutuote, peli saa pelaajan ajattelemaan ja tätä kautta ymmärtämään. Pelin opetus ei välttämättä aina ole hyödyllinen tai järkevä ”opettava”, mutta silti se on olemassa. Palaute on myös

tyypillinen piirre pelien keskuudessa, siitä pelaajalle selviää yleensä, miten hyvin tai huonosti hän onnistui saavuttamaan asetetun päämäärän. Palautetta pelaajalle voi ilmaista kesken pelin tai sen jälkeen. Palaute on myös todella tärkeä osa oppimiskokemusta peleissä, sillä se antaa suuntaa siitä mitä pelaaja oppi pelaamisen aikana. (Mononen 2003.)



Kuva 6. Kaavio esittää ominaisuuksia, mitä pelit sisältävät, niin videopelien, opetuksen ja työskentelyn saralla (mukaillen Cheek 2015)

Tämä alaluku on myös esimerkki siitä, että käsitettä peli on vaikea määritellä yksimielisesti, se voi olla tiettyssä mittakaavassa jopa mahdotonta. Asiasta tietoa etsiessä esille nousee jatkuvasti yksi nimi, 1900-luvulla elänyt filosofi Ludwig Wittgenstein. Hän väitti, että pelejä ei voi määritellä yksiselitteisesti, vaan ne määrittävät itsensä toisten pelien kautta. Eli ainoa tapa määritellä peli olisi kuvailla muita pelejä yleisellä tasolla (Anat 2014). Joku voisi pitää hieman outona, että tämä 1900-luvun alun jälkeen syntynyt määritelmä on vielä tänä päivänäkin monien mielestä se "oikea", ehkä tämä seikka yksinään jo todistaa jotain. Kaikki eivät tähän kyseiseen määritelmään kuitenkaan tyydy, artikkelissa nimeltään "What is a game?" lainataan pelisuunnittelijoiden Katie Salen ja

Eric Zimmerman kehittelmää määritelmää pelistä. He kuvailevat peliä järjestelmänä, jossa pelaajat kohtaavat sääntöjen asettamia keinotekoisia ristiriitoja ja päätyvät jonkinlaiseen tulokseen lopussa (Ruben 2009.).

3.2 Pelillisyyys ja pelillistäminen ohjelmistokehityksessä

Aiemmassa alaluvussa kävimme läpi mistä elementeistä peli koostuu ja mitä voi kutsua peliksi. Pelillisyyys (engl. *gamification*) on ikään kuin noiden kyseisten elementtien lainaamista pohjautuen siihen, että lopputulosta ei kuitenkaan välttämättä kutsuta peliksi. Otetaan vain osa pelillisistä elementeistä, päämäärä silti pysyy samana. On yleistä yhdistää pelit viihteeseen, pelillisyyys taas ei välttämättä liity viihdyttämiseen millään tavalla. Pelillisyyttä käytetään usein ikään kuin verhona tai pehmusteena. Tavoitteena olisi saada joukko ihmisiä oppimaan uusia asioita, innostumaan jostain tai perehtymään uusiin asioihin. Lisäämällä tämän kaltaisiin ”pakollisiin” toimenpiteisiin pelinkaltaisia elementtejä voi tehdä niiden omaksumisesta kevyempää ja mieluaisampaa.

Millä keinoin käytännössä ohjelmakehittäjä voi hyödyntää pelillisyyttä ja pelillistämistä omilla verkkosivuillaan tai sovelluksissa ja miten pelillisten elementtien lisääminen voi vaikuttaa sovelluksen käyttäjäkokemukseen positiivisesti? Esimerkiksi yksi pelillisistä elementeistä, jolla voi kartuttaa sovelluksen käyttäjämääriä on satunnaisuuden hyödyntäminen. Satunnaisuutta voi hyödyntää esimerkiksi erilaisilla arvonnoilla. Käyttäjä kokee arvonnän kaltaisen satunnaisuuden jännittävänä ominaisuutena, palkinnonkaan ei tarvitse olla kummoinen. Pelimäisen satunnaisuuden voi yhdistää visuaaliseen efektiin kuten pyörivään ”onnenpyörään”, joka pyörähtää, kun käyttäjä painaa painiketta. Visuaaliset efektit ja animaatiot lisäävät jännityksen tuntua. Toistuvat esimerkiksi kuukausittaiset arvonnät lisäävät myös mahdollisuutta siihen, että käyttäjät palaavat takaisin sovelluksen pariin. Pitkäaikaiset käyttäjät ovat niitä mitä kaikki ohjelmistokehittäjät haluavat saavuttaa. (Rabkina 2015.)

Toinen pelillinen elementti, jota nykyään esimerkiksi useat verkkokaupat hyödyntävät on mahdollisuus antaa sivuston käyttäjien vaikuttaa tuotepalautteisiin. Tällä tarkoitetaan esimerkiksi ominaisuutta, jossa käyttäjät voivat antaa muiden käyttäjien jättämälle palautteelle esimerkiksi ylä- tai alapeukalon. Periaatteessa siis käyttäjät jättävät palautetta, jonkun toisen jättämälle palaut-

teelle. Ohjelmistokehittäjälle tai sovelluksen ylläpitäjälle tämä tarkoittaa sitä, että sovelluksen käyttäjät periaatteessa moderoivat palautteita ohjelmistokehittäjän puolesta. Toisaalta, sovelluksen käyttäjien näkökulmasta tekniikka kannustaa jättämään herkemmin palautetta, koska se että muut sovelluksen käyttäjät pitävät palautteestasi tuottaa käyttäjälle mielihyvää. Hieman samaan tyyliin kuin esimerkiksi facebookissa niin sanottujen ”tykkäysten” vastaanottaminen tuntuu palkitsevalta. Tämä aspekti muodostaa myös eräänlaisen kilpailun siitä, kuka käyttäjistä jättää hyödyllisimmän palautteen, koska yleensä esimerkiksi verkkokaupat nostavat eniten tykkäyksiä saaneen palautteen palautteviestien kärkeen. (Rabkina 2015.)

Kolmas pelillinen elementti, joka nauttii nykyään suurta suosiota eri sovellusten markkinointitekniikoissa, ovat eräänlaiset tehtävät. Tehtäviä suorittamalla sovelluksen käyttäjä vastaanottaa pienen bonuksen tai palkinnon. Tunnettu pelisuunnittelun kulmakivi on käännetty juonikkaasti toimivaksi markkinointikeinoksi. Klassinen esimerkki tämänkaltaisesta pelillistämisestä on palkita käyttäjä, jos hän jakaa sivustosi linkin tunnetuilla sosiaalisen median sivustoilla. Palkintona yleensä on esimerkiksi alennuskuponki tai jonkinlainen etuoikeus verkkosivustolle tai sovellukseen. Ohjelmistokehittäjälle tai sovelluksen ylläpitäjälle tämä on halpa keino lisätä markkinointia ja levittää sanaa sovelluksesta laajemmin. (Rabkina 2015.)

Pelillistämistä hyödynnetään nykyään valtaosassa verkkosivustoja ja sovelluksia, keinolla tai toisella. Pelillistäminen on tärkeä osa käyttäjien interaktiota ja sitä mikä tekee sovelluksesta tai verkkosivusta koukuttavan. Se täytyy kuitenkin osata integroida mukaan oikein, pelillisellä elementillä täytyy olla jokin päämäärä ja tarkoitus. Liika pelillisyyden kanssa kikkailu voi myös aiheuttaa päinvastaisen efektin ja ärsyttää sovelluksen käyttäjiä. Onnistuessaan pelilliset elementit ovat oiva lisä sovellukselle tai verkkosivustolle.

3.3 Oppimispelit

Kun pelillisyyttä käsitteestä etsii tietoa, saattaa huomata nopeasti, että käsite pelillisyyttä yhdistetään usein oppimiseen, perehdyttämiseen tai osaamisen kehitykseen. Tästä seikasta ja siitä, että myös yksi osa opinnäytetyöni teknistä

toteutusta oli eräänlainen oppimispeli niin aion tässä luvussa käsitellä varsin-kin tätä pelillisyyden ja oppimispelien yhteyttä.

Lapset ja nuoret seuraavat teknologian kehitystä, mediaa ja pelejä yhä kasvavissa määrin. Tutkimus nimeltään ”Mobiilimuksut” paljasti, että nuorten ja lasten teknologian päivittäisessä käytössä on tapahtunut huomattava muutos niin käyttöajassa kuin myös siinä, että mobiililaajakaistojen yleistymisen jälkeen lapset ja nuoret voivat olla jatkuvasti yhteydessä verkkoon. Tutkimuksen osoittamat tulokset ovat myös siirtyneet koulumaailmaan. Tulokset ovat huomattavissa lisääntyneinä älytauluina, tablet-tietokoneina ja pelien opetuskäytön lisääntymisestä koulujärjestelmässä. (Kangas & Koskinen, Krokfors 2014.)

Oppimispelejä on tutkittu suhteellisen paljon viime vuosikymmenen aikana. Tutkimukset ovat esimerkiksi kertoneet, että oppimispelejä hyödynnetään eniten matematiikan opiskelussa, jopa yksi kolmasosa kaikista oppimispeleistä ovat matematiikan opiskeluun suunnattuja. Suosiossa matematiikan taakse jäivät esimerkiksi aihe-alueet kuten maantieto, luonnontieto, terveystieto, historia ja kielet. Yksi teoria siihen, että juuri matemaattiset oppimispelit ovat suosituimpia, on yksinkertaisesti se, että tietokoneet ja ohjelmointikielet pohjautuvat yleensä matemaattisiin ratkaisuihin. Tietokoneet ja puhelimet ovat periaatteessa kumminkin vain kehittyneitä laskimia, joten matemaattisten pelien kehittäminen voi tuntua selvimmältä vaihtoehdolta markkinoille, jotka ovat kuitenkin vielä suhteellisen uudet. (Mylläri & Vesterinen 2014.)

Oppimispelien määrittelyyn ei ole olemassa yhtä kaavaa. Oppimispelit voivat poiketa toisistaan paljonkin siinä miten opittava asia on sisällytetty oppimispeliin. Jotkut oppimispelit voivat olla juurikin niitä toistoa vaativia matematiikka harjoitteita, joissa ainoa tavoite on saada mahdollisimman paljon pisteitä. Toiset taas voivat esimerkiksi visuaalisen ilmeen tai tarinan avulla vedota enemmän pelaajan tunteisiin ja ilmaisutaitoon.

Opetushallituksen ylläpitämä ”Edu.fi” -verkkosivusto kertoo, että oppimispelit voidaan jakaa pedagogisesti kolmeen alaryhmään, eli behavioristisiin-, kognitiivisiin- ja konstruktionistisiin oppimispeleihin. Behavioristiset oppimispelit ovat esimerkiksi juuri niitä matemaattisia pelejä, joita kuvailin tämän alaluvun alussa. Tämän alaryhmän oppimispelit perustuvat toistoon ja muistin kehittämi-

seen. Pelaaja vastaa kysymykseen ja saa välittömän palautteen pisteiden välityksellä. Tämä on suoraviivaisin ja selkein alaryhmä oppimispelien joukosta.

Kognitiivisien oppimispelien alaryhmä onkin hieman monimutkaisempi, nämä oppimispelit nojaavat enemmän visuaalisuuteen, kuten kuviin, tekstiin ja ääniin. Kognitiiviset oppimispelit vetoavat enemmän pelaajan tunteisiin ja aktiivisuuteen. Hyvinä esimerkkeinä kognitiivisista oppimispeleistä ovat esimerkiksi erilaiset simulaatiopelit, jotka harjoittavat pelaajaa oikeaa vastaavanlaista tilannetta varten. Viimeinen eli kolmas oppimispelien alaryhmä on konstruktivistiset oppimispelit, näissä pelaaja reflektoi omaa oppimistaan ja ne turvautuvat enemmän ”tekemällä oppii” -mentaliteettiin. Tätä kutsutaan kokemukselliseksi oppimiseksi. Ongelmanratkaisu taidon kehittäminen on iso osa konstruktivistisia oppimispelejä. (Opetushallitus, Edu 2016.)

Herää siis kysymys, perustuuko tulevaisuudessa oppiminen ja koulutus täysin oppimispeleihin ja sähköisiin ratkaisuihin. Ainakaan kunnioitetun ”Scientific American” -julkaisun oppimispelejä koskeva artikkeli ei usko moiseen. Artikkelin esittää näkökannan, joka paljastaa, että vaikka oppimisen ja pelaamisen yhteyttä onkin tutkittu suhteellisen runsaasti, niin harva tutkimus on löytänyt selvää yhteyttä oppimispelien vaikutuksesta luokahuoneen suorituskäyttöön tai akateemisiin saavutuksiin. Toisaalta on tieteellisesti todistettu, että oppimispelit tai pelit ylipäänsä kehittävät muistia, joka on yksi kognitiivisista taidoista jota oppiminen vaatii. Osa oppimispelien vastaanottamasta ylistyksestä saatetaan siis olla eräänlaista markkinointia. Artikkelin myös ennustaa, että oppimispelien pelaamisen sijaan oppilaat voisivat hyötyä enemmän siitä, että kouluissa opetettaisiin pelisuunnittelua ja ohjelmointia jo varhaisimmilta luokkasteilta alkaen. Loppukaneettina voisi siis sanoa, että oppimispelit ovat hyvä lisä tukemaan opetusta ja niiden hyödyntäminen tulee lisääntymään opetuksessa, mutta myöskään tavalliset opetusmenetelmät eivät ole vaarassa kuolla sukupuuttoon. (Malykhina 2014.)

4 KÄYTÄNNÖN TOTEUTUS

Tässä luvussa käsitellään, miten käytännössä toimeksianto on toteutettu. Tämä luku on jaettu kahteen isompaan kokonaisuuteen, jotka käsittelevät kasvinsielailuosa- ja kasvintunnistuspeliä. Molemmista osa-alueista käydään

läpi muun muassa rautalankamallit, layoutit, ohjelmalliset elementit ja pelilliset toteutukset.

4.1 Lähtökohdat

Sanoma Pro toimeksiantajana antoi minulle teknisen toteutuksen puolesta vapaat kädet, kunhan sovellus vain toimisi mobiililaitteilla ilman ongelmia. Sovelluksen visuaalisesta puolesta ja käyttöliittymästä vaatimuksina mainittiin termejä kuten puoleensavetävä, selkeä ja helppokäyttöinen. Oli myös tärkeää, että sovelluksen kohderyhmä (alakoululaiset) oli otettu huomioon.

Ominaisuuksien puolesta vaatimukset olivat seuraavanlaiset: kasveja pitää pystyä selaamaan luokka-asteittain, sovelluksesta pitää löytyä nimihaku, jonka avulla kasveja voi hakea ja sovelluksessa pitää olla pelillinen elementti, jossa käyttäjä voi testata kasvientunnistustaitojaan. Pelilliseksi elementiksi kaavailimme kasvientunnistuspelin, jossa käyttäjä yrittää tunnistaa eri kasveja kuvan ja tekstivaihtoehtojen perusteella. Kasvientunnistuspelin tasojen vaikeus skaalautuisi alakoulun luokka-asteiden mukaisesti.

Kehitin sovellusta ajatuksentasolla eteenpäin ja suoritin samalla ”Johdatus opinnäytetyöhön” -opintojaksoa. Opintojakson lopuksi kävin esittelemässä opinnäytetyöni aiheen koululla ja aihe hyväksyttiin. Tämän jälkeen aloitin sovelluksen toteuttamisen.

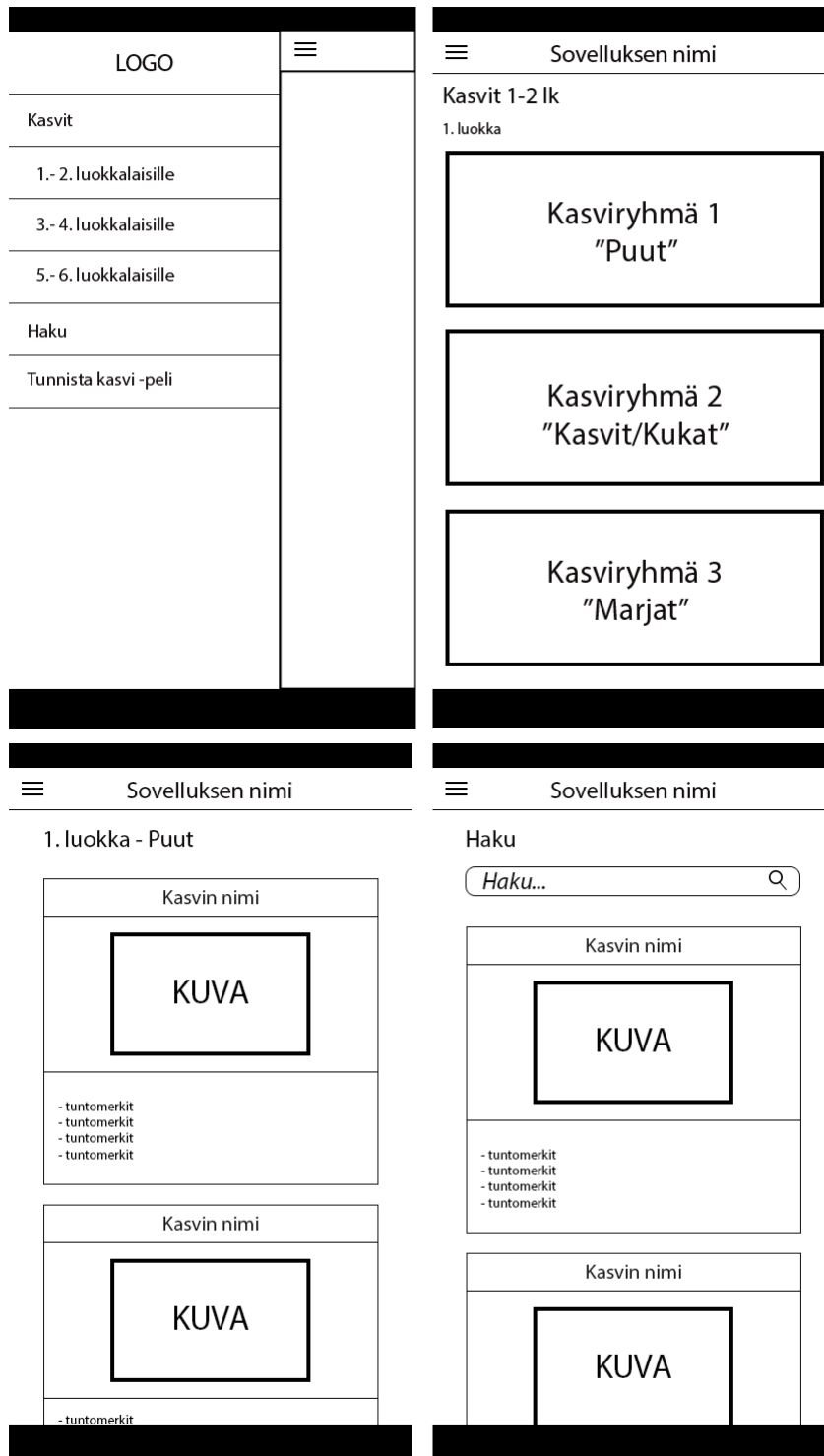
4.2 Kasvienselailuosa

Päätin toteuttaa sovelluksen Apache Cordova, AngularJS ja Ionic framework tekniikoita käyttäen. Työskentely alkoi sillä, että asensin Cordova-sovellusympäristön ja muutkin tarvittavat työkalut. Mobiilisovelluksen ohjelmointiin päätin käyttää NetBeans-editoria. Tässä luvussa käyn läpi, miten käytännössä toteutin kasvienselailuosaan rautalankamallit, eri näkymät, valikon ja hakuominaisuuden.

Rautalankamallit

Toteutin rautalankamallit käyttöliittymästä Adobe Photoshop -ohjelmistolla. Ohjelmistoja rautalankamallien toteuttamiseen on toki muitakin, itsekin aloitin

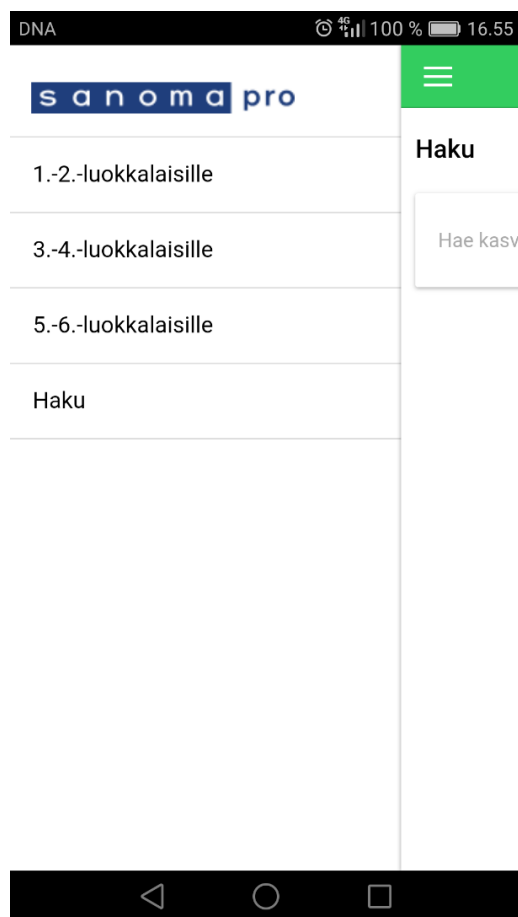
työskentelyn selainpohjaisella draw.io-sovelluksella, mutta lopuksi päädyin itselleni tuttuun ja turvalliseen Photoshopiin. Rautalankamallit toteutin valkoiselle pohjalle, johon piirsin käyttöliittymän ääriviivat Photoshopin viivatyökalua ja viivoittimia apuna käyttäen. Rautalankamalleista ilmenee esimerkiksi käyttöliittymän painikkeiden paikat, osa otsikoista ja eri elementtien asettelu sovelluksessa.



Kuva 7. Rautalankamallit kasvienselailusovelluksen näkymistä

Sivuvalikon ja näkymien toteutus

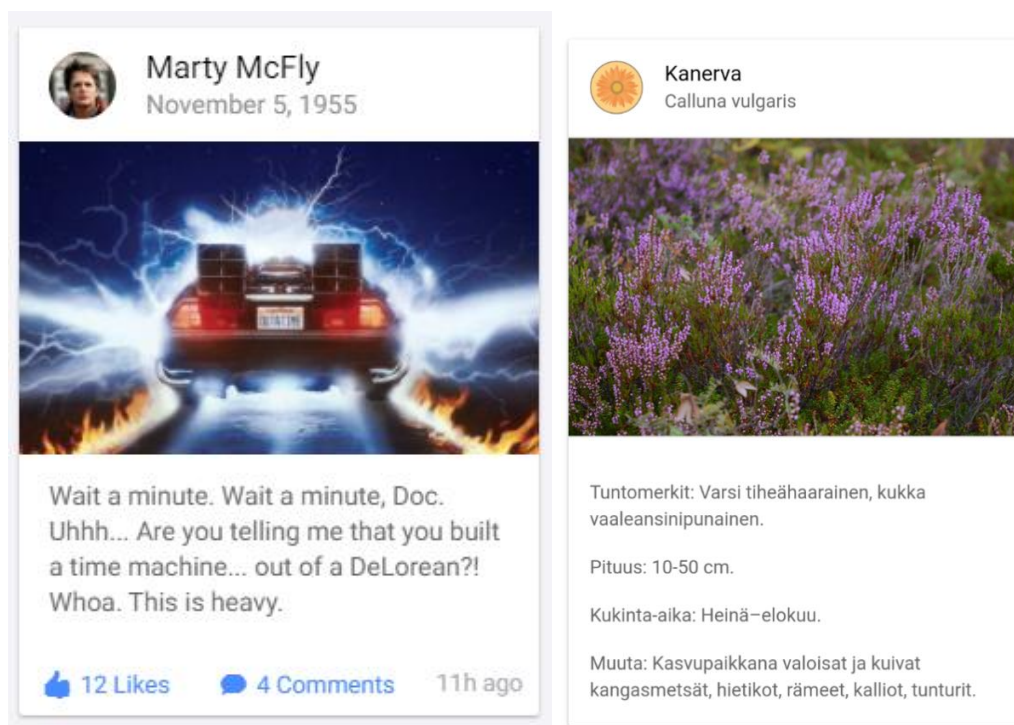
Ohjelmoinnin aloitin rakentamalla sovellukseeni rautalankamallin mukaisen valikkorakenteen. Päätin käyttää valikkona Ionicin tarjoamaa ion-side-menu -komponenttia. Tämä vaati myös tutustumista Angularin ngRoute -moduuliin. Laitoin valikon aukeamaan sovelluksen vasemmalta laidalta, joko käyttäjän painaessa "hampurilaismenu" -ikonia tai raahaamalla sormea näytöllä oikealle. Valikkoon hahmottelin otsikoiksi sovelluksen eri osat, jotka menevät seuraavanlaisesti: 1.-2.-luokkalaisille, 3.-4.-luokkalaisille, 5.-6.-luokkalaisille ja Haku. Päätin laittaa aina kaksi luokka-astetta samaan nippuun, jottei valikkorakenteesta tulisi liian pitkää (kuva 8).



Kuva 8. Havainnollistava kuva sovelluksen sivuvalikosta toteutettuna

Aloin miettiä mikä olisi paras ja selkein tapa esittää kasvit käyttäjälle. Lähdin kokeilemaan Ionicin css-komponenttia nimeltä list-card. List-card-komponentti hyödyntää ionicin mukana tulevia "kortteja", ne ovat selkeä tapa esitellä joku yksittäinen itemi tai objekti sovelluksessa. List-card-komponentti sisältää pienen avatar-kuvan, otsikon, isomman kuvan ja tekstikentän. Tämä sopisi hyvin kasviteemaan. Avatar-kuvaan voisi laittaa pienen logon riippuen siitä, mitä

kasviryhmää kasvi edustaa, otsikkoon kasvin nimen, jonka alle kuva kasvista ja tuntomerkit (kuva 9).



Kuva 9. Ionic-dokumentaation oma esimerkki list-card komponentista verrattuna omaan toteutukseen

Nämä eri kortit siis tulostuisivat sovelluksessa allekkain. Korttien sisältö, kuten kasvien nimet, kuvat ja tuntomerkit tulisivat json-tiedoston kautta (kuva 10).

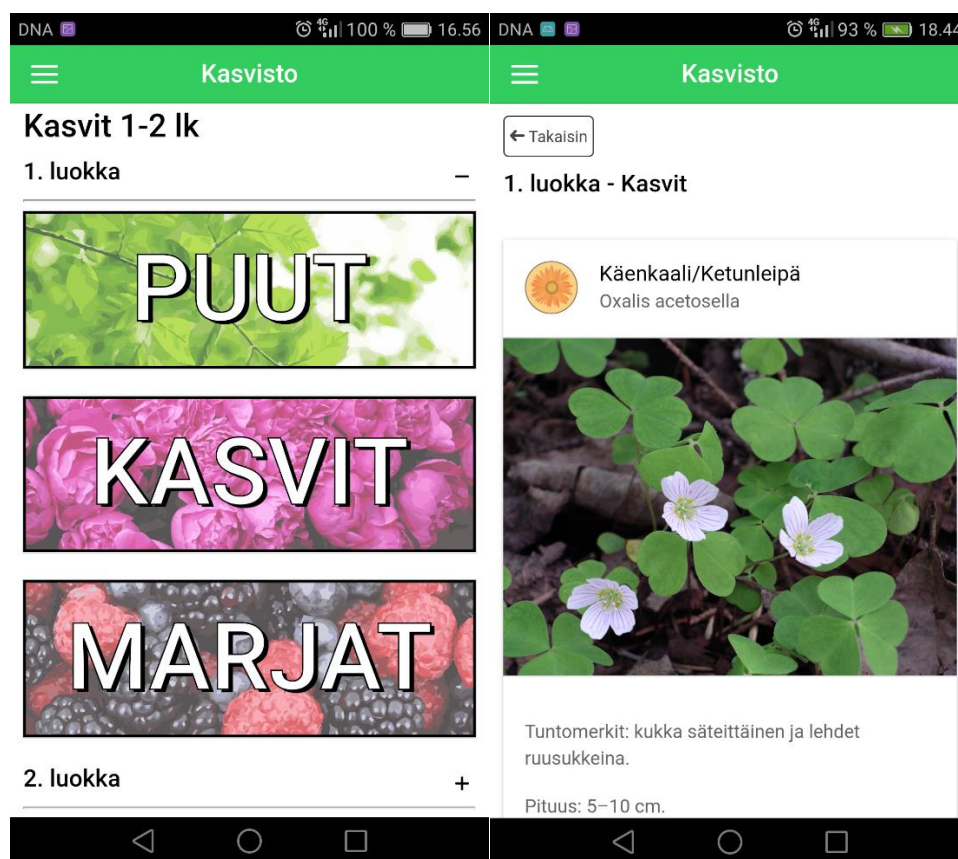
```
{
  "pikkukuva": "img/pikkukuvat/kukat1.png",
  "nimi_fi": "Käenkaali/Ketunleipä",
  "nimi_lat": "Oxalis acetosella",
  "tuntomerkit": "Tuntomerkit: kukka säteittäinen ja lehdet ruusukkeina.",
  "pituus": "Pituus: 5-10 cm.",
  "kukintaaika": "Kukinta-aika: Touko-kesäkuu",
  "muuta": "Muuta: Käenkaali on sopeutunut kasvamaan metsänpohjan varjossa.",
  "kuva": "img/luokkal/kaenkaali.png"
},
```

Kuva 10. Esimerkki yhdestä json-solusta

Kasveja jokaisella luokka-asteella on noin 15 kappaletta ja luokka-asteita kuuksi kappaletta, joten json-tiedostojen kasaaminen tuli viemään yllättävän paljon aikaa. Muokkasin myös kasvien kuvat pienemmiksi Adobe Photoshop -ohjelmalla, jotta sovelluksen tiedostokoko ei kasvaisi liian suureksi. Sovelluksen html-puolella kortit tulostettiin näkyviin li-tägi listassa, jossa jokaista korttia kohdellaan iteminä. Tulostuksessa käytetään hyväksi myös angularin ng-repeat -direktiiviä.

Tässä vaiheessa, kun käyttäjä painoi sivuvalikosta kohtaa ”1. – 2. - luokkalaisille”, tulostui kaikki ykkös- ja kakkosluokan kortit allekkain siististi, yksi kasvi yhtä korttia kohden. Tämä toteutus oli liian sekava ja käyttäjän olisi tuskallista selata pitkää, noin 30 kasvin listaa, jos molemmat luokka-asteet laskee mukaan.

Olisi parempi, jos kasvit lajittelisi muutama alaryhmään, joiden kautta käyttäjä voisi navigoida helpommin eri luokka-asteen kasvien läpi. Kehittelin kolme alaryhmää, jotka olivat Puut, Kasvit ja Marjat (kuva 11). Kysyin myös toimeksiantajalta, olivatko kyseiset alaryhmät kelpollisia ja he hyväksyivät idean.



Kuva 11. 1.-2. luokan näkymä, jonka jälkeen käyttäjä on klikannut ”Kasvit” alavalikkoa

Käyttöliittymästä oli syytä tehdä vielä hieman selkeämpi, joten päätin oletuksena piilottaa alaryhmien napit. Napit liukuisivat animaation avulla näkyviin käyttäjän painaessa joko tekstiä ”1. luokka”, viivaa sen alapuolella tai ”+”-merkkiä (kuva 11). Tämän toteutin tekemällä painikkeiden ympärille ”divin” eli kehyksen, joka oletuksena piilotti napit näkyvistä. Ng-show -komento taas

varmistettiin sen, että napit tulevat näkyviin, kun käyttäjä painaa määrätystä kohdasta.

```
<h3>Kasvit 1-2 lk</h3>
<div ng-click="toggle(klikki='klikki')">
  <span ng-class="{ 'icon ion-minus-round': klikki, 'icon ion-plus-round': !klikki}" ></span>
  <h4>1. luokka</h4>
  <hr>
</div>

<div id="animaatio" class="check-element animate-show-hide" ng-show="avaaValikko">

  <a ng-href="#/puut/luokka1" class="item" id="linkki" cache-view="false">

    <ion-card>
      
      <ion-card-title>
        <h1 class="valiotsikkovari">PUUT</h1>
      </ion-card-title>
    </ion-card>

  </a>
```

Kuva 12. Divin piilottaminen ikonin painamalla koodin muodossa.

Animointi lisättiin CSS-puolella käyttämällä transition-komentoa. Lisäsin myös ominaisuuden, jossa plusmerkki muuttuu miinusmerkiksi, kun nappia on painettu. Havainnollistamaan sitä, että napit ovat näkyvillä. Tämän ominaisuuden toteutin luomalla klikki-käsittelijän, joka tutkii, onko nappia painettu ja sen mukaan vaihtaa näytettävän ikonin (kuva 12).

Hakuominaisuuden toteutus

Sovelluksen kasvienselailuosiosta puuttui vielä yksi toimeksiantajan esittämä suurempi ominaisuus eli kasvien nimihaku. Nimihaun tarkoituksena olisi, että käyttäjä pystyy hakemaan minkä tahansa luokka-asteen kasveja nimen avulla ja tarkastelemaan niitä. Hakuominaisuuden rakensin sivuvalikkoon omana otsikkonaan. Tämän jälkeen piti koostaa yksi iso json-tiedosto, joka käsitti kaikki sovellukseen kuuluvat kasvit, kuvineen ja selityksineen. Tässä vaiheessa oli luotu jokaiselle luokalle ja niiden alaryhmälle omat json-tiedostot, joten pohjatyö oli sinänsä valmiina. Näiden json-tiedostojen tiedot kopioitiin haku-toiminnon json-tiedostoon. Nyt kasassa oli kaikki tarvittava data koottuna hakuominaisuutta varten.

Hakutoiminnon ohjelmointi pystyi alkamaan. Ensimmäinen tavoite oli tulostaa koko json-tiedosto ul-tägiä käyttämällä pitkäksi listaksi. Näiden ul-tägien sisälle luotiin li-tägi, joka käsittelee Angularin ng-repeat nimisen direktiivin avulla

jokaisen json-tiedoston kasvin/solun erikseen (kuva 13). Tyyli oli siis sama kuin sovelluksen muissakin osissa. Kun json-tiedoston data oli onnistuneesti tulostettu sovellukseen oikein, luotiin input-tekstikenttä. Ilmoitin myös ng-model-direktiivin avulla ohjelmalle, että etsii hakutuloksia perustuen kasvin nimeen. Input-kentän laitoin ionicin ion-card -komponentin sisään, jotta se näyttäisi mahdollisimman siistiltä ja sopisi sovelluksen muuhun ulkoasuun.

Jotta tekstikenttään kirjoittamisella olisi mitään vaikutusta hakutuloksiin, piti li-tägin sisään lisätä "filtteri". Filtterit ovat yksi AngularJS:n ominaisuuksista, niiden avulla voi hallita ja suodattaa dataa. Tässä tapauksessa dataa, joka tulee json-tiedostosta. Filtterin lisäämisen jälkeen haku alkoi toimimaan. Tässä vaiheessa hakuominaisuus näytti vakiona kaikki tulokset tekstipalkin alla ja sitä mukaan kuin palkkiin alkoi kirjoittaa tietyn kasvin nimeä, tuloksia alkoi suodattua pois.

```
<ul class="list">
  <li class="item1" ng-repeat="kaikki_kasvi in kaikki_kasvit | orderBy: nimi_fi : false | filter: hakusana " ng-show="hakusana.nimi_fi">
    <div class="list card">
      <div class="item item-avatar">
        
        <h2>{{ kaikki_kasvi.nimi_fi }}</h2>
        <p>{{ kaikki_kasvi.nimi_lat }}</p>
      </div>
      <div class="item item-body">
        
        <p>{{ kaikki_kasvi.tuntomerkit }}</p>
        <p>{{ kaikki_kasvi.pituus }}</p>
        <p>{{ kaikki_kasvi.kukintaaika }}</p>
        <p>{{ kaikki_kasvi.muuta }}</p>
      </div>
    </li>
</ul>
```

Kuva 13. Hakuominaisuus koodin muodossa

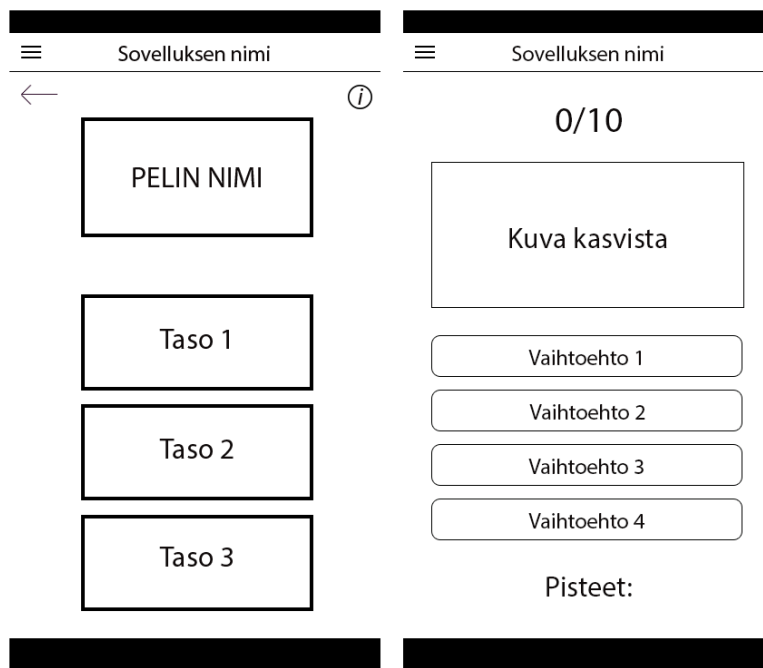
Tässä demonstroituu hyvä esimerkki Angularin parhaista puolista. Haun tulokset päivittyvät dynaamisesti ja muokkautuvat jokaisen kirjaimen perusteella, minkä käyttäjä tekstipalkkiin syöttää. Eli käyttäjän ei tarvitse painaa nappeja rekisteröidäkseen hakua tai kirjoittaa kasvin koko nimeä tarkasti saadakseen tuloksia. Angularin filter -hakuominaisuuden avulla käyttäjä voi hakea vaikka kaikkia kasveja, jotka alkavat A-kirjaimella, jos niin haluaa. Mainitsin aiemmin, että jos haun tekstikenttä oli tyhjä, kaikki kasvit tulostuivat vakiona allekkain ja harvenivat sen mukaisesti, kun käyttäjä alkoi kirjoittaa tekstikenttään. Tavoitteena oli, että haku toimisi juuri toisinpäin, tämä toteutettiin laittamalla li-tägin sisään "ng-show" komento (kuva 13). Tämä komento piilotti kaikki json-tiedostosta tulevat osumat, kunnes käyttäjä kirjoittaa tekstipalkkiin.

4.3 Kasvientunnistuspeli

Kasvientunnistuspeliä aloin lähestyä samoilla tekniikoilla kuin kasvien selailuakin, eli tekemällä rautalankamallit mahdollisesta toteutuksesta. Toteutin rautalankamallit taas Adobe Photoshop -ohjelmalla. Tein peliä varten neljä erilaista rautalankamallia. Tässä alaluvussa käyn järjestelmällisesti läpi, miten käytännössä toteutin kasvientunnistuspeliin rautalankamallit, layoutit ja ohjelmoinnin.

Rautalankamallit

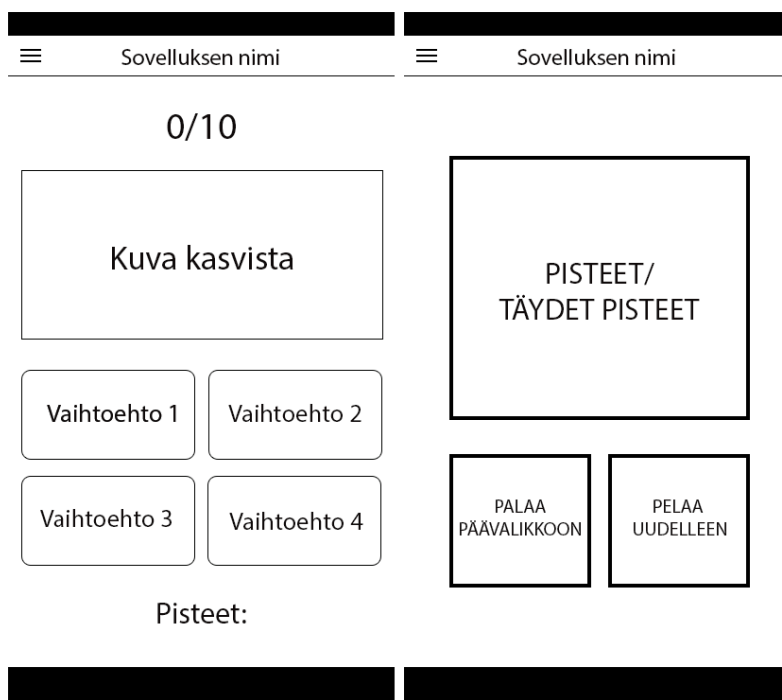
Ensimmäinen tekemäni rautalankamalli kuvaa pelin aloitusvalikkoa. Tässä mallissa on pelin nimi näytön ylälaidassa, kolme nappia joista pelaaja valitsee tason, rasti pelistä poistumiseen ja ohjenappi peliohjeita varten (kuva 14). Toisen tekemäni rautalankamalli kuvaa pelinäyttöä, malli noudattaa aika perinteistä visailupeli-kaavaa. Näytön yläosassa näkyy, montako kysymystä pelissä on vielä jäljellä. Sen alapuolella näkyy kuva kasvista joka pelaajan pitää tunnistaa, jonka alapuolelta löytyy vastausvaihtoehdot. Vastausvaihtoehtojen alapuolella näkyy pisteet, jotka pelaaja on kerännyt (Kuva 14).



Kuva 14. Esimerkkikuva pelivalikon ja pelinäytöstä (vaihtoehto 1) rautalankamalleista

Toteutin pelinäytöstä myös vaihtoehtoisen rautalankamallin, siinä on muutoin samantyylinen asettelu, mutta vastausvaihtoehtopainikkeet on järjestelty toi-

sella tavalla. Viimeinen eli neljäs rautalankamalli kuvaa lopetus/pisteytysnäyttöä (kuva 15).



Kuva 15. Esimerkkikuva pelinäytöstä (vaihtoehto 2) ja palautenäytön rautalankamalleista

Tämä näyttö siis tervehtii pelaajaa, kun hän on vastannut kaikkiin pelin kysymyksiin. Pelaaja näkee näytön yläosasta, montako pistettä hän onnistui keräämään. Pisteet ovat näytön yläosassa, koska pelaajan huomio kiinnittyy sinne ensimmäisenä ja hän saa välittömän palautteen siitä onko kehitystä tapahtunut. Tästä samasta näytöstä pystyy myös palaamaan pelin päävalikkoon tai pelaamaan saman tason uudelleen (kuva 15).

Layoutit

Kun kasvintunnistuspelin rautalankamallit olivat valmiina, siirryin tekemään layouteja pelille. Layoutien eli asettelumallin ideana on havainnollistaa pelin visuaalisia elementtejä pidemmälle, kuin rautalankamallit antavat ymmärtää. Layouteista selviää esimerkiksi minkälaisia fontteja, taustakuvaa, elementtien muotoilua ja painikkeiden muotoilua tunnistuspelin visuaalisessa ilmeessä käytetään. Layoutien toteutus oli myös tärkeää, koska toimeksiantajaa kiinnosti kovasti mille kasvintunnistuspeli tulisi näyttämään. Oli tärkeää siis näyttää heille idea visuaalisesta ilmeestä, ennen kuin pelin ohjelmallinen toteuttaminen aloitettiin.

Layoutit toteutettiin myös Adobe Photoshop -ohjelmalla. Työskentely alkoi luomalla dokumentin, jonka canvasin/taustan resoluutioksi määritettiin 1080x1920 pikseliä. Se on yleinen resoluutio nykypuhelimissa ja oma puhelin tuki myös tätä resoluutiota. On myös tärkeää mainita, että lähdin suunnittelemaan layouteja, sillä periaatteella, että peliä pelataan puhelin pystysuunnassa ns. "Portrait-näkymässä". Peliä voisi lukita tähän näkymään ohjelmoinnin kautta, joka tullaan tekemään myöhemmässä vaiheessa. Peli ei välttämättä toimisi niin hyvin ns. "Landscape" eli vaakatilassa koska useimmat kasvien kuvat ovat pystymuodossa.

Kun taustan/canvasin koko ja muoto oli selvillä, alkoi taustakuvan suunnittelu. Tavoitteena oli luoda yhtenäinen tausta, joka toistuu ja pysyy mukana pelin eri vaiheissa. Taustan olisi myös hyvä sisältää elementtejä, jotka liittyvät jotenkin itse peliin, tässä tapauksessa esimerkiksi kuvia kasveista. Ideana oli tehdä taustasta näyttävä, mutta samaan aikaan sellainen, joka ei häiritse käyttäjää.

Taustan pohjaväriksi tuli vaaleanvihreä, koska vihreä yhdistetään usein kasveihin. Käytin kuvien etsimiseen apunani "Creative Commons" -hakutyökalua, jonka kautta löytää kuvia, joita voi käyttää vapaasti kaupallisiin tuotteisiin ja kuvia, joita saa vapaasti muokata omaan käyttöön. Hakuprosessin aikana löytyi kuvia puunlehdistä, joita päätin käyttää osana taustakuvaa. Muokkasinkin kuvaa lehdistä mieleisekseni ja ripottelin niitä ympäri vihreää taustaa järjestelmällisesti, lopuksi piirsin viivatyökalulla viivoja lehtien välille, luomaan tietynlaisen toistuvan kuvion taustaan. Taustakuva viimeisteltiin hyödyntämällä erilaisia Photoshopin tarjoamia säätötasojia, joilla taustaan syntyi yhtenäinen vaikutelma. (Kuva 16.)

Kun taustakuva oli valmis, oli aika siirtyä työstämään painikkeita. Painikkeissa on tärkeää, että ne näyttävät näytöllä elementeiltä joita voi painaa. Painikkeissa pitää siis olla tietynlainen syvyydentunne, tätä syvyydentunnetta luodaan layouteissa varjostuksilla ja ulkoreunan kevyellä hohdolla. Tavoitteena oli myös tehdä napeista hieman "leikkisän" näköiset raidoittamalla ne, ottaen huomioon pelin kohderyhmänä olevat alakoululaiset. (Kuva 16.)



Kuva 16. Havainnollistava kuva taustasta ja yhdestä painikkeesta

Nyt kun kasvientunnistuspelille oli suunniteltu taustakuva ja painikkeiden ulkoasu, pystyi aloittamaan niiden hahmottelun rautalankamallien esittämiin näkymiin. Aloitin pelin valikkonäytöstä. Laitoin taustan paikoilleen ja kopion näytön alareunaan kolme kappaletta luomiani painikkeita. Painikkeiden sisälle kirjoitin tason nimen. Fonttina toimii ”Slab-serif” -fonttiperheeseen kuuluva Rockwell-fontti. Varustin fontin vielä tuplääriviivoilla.

Nappien yläpuolella on pelin otsikko, sille tarvittiin oma fontti. Otsikkofontti löytyi sivulta nimeltä ”dafont.com”, fontin nimi oli ”Pw Perspective”. Tämä fontti kuului myös kategoriaan, joita sai vapaasti käyttää. Otsikko pelkiltään pelin valikossa näytti yksinäiseltä, joten sen ”alle” tarvittiin jonkinlainen taustaelementti. Toteutin piirtotyökalua apuna käyttäen Adoben Illustrator -ohjelmalla kukan, joka liitettiin otsikon alle pelin valikkonäkymään. Lopuksi lisäsin vielä valikkonäytön oikeaan yläreunaan rastin, jota painamalla käyttäjä voisi sitten varsinaisessa versiossa poistua pelistä ja vasempaan yläreunaan infomerkkin,

jota painamalla avautuisi pop-up ikkunassa olevat ohjeet pelinkulkuun (kuva 17).



Kuva 17. Havainnollistava kuva valikkonäkymä layoutista

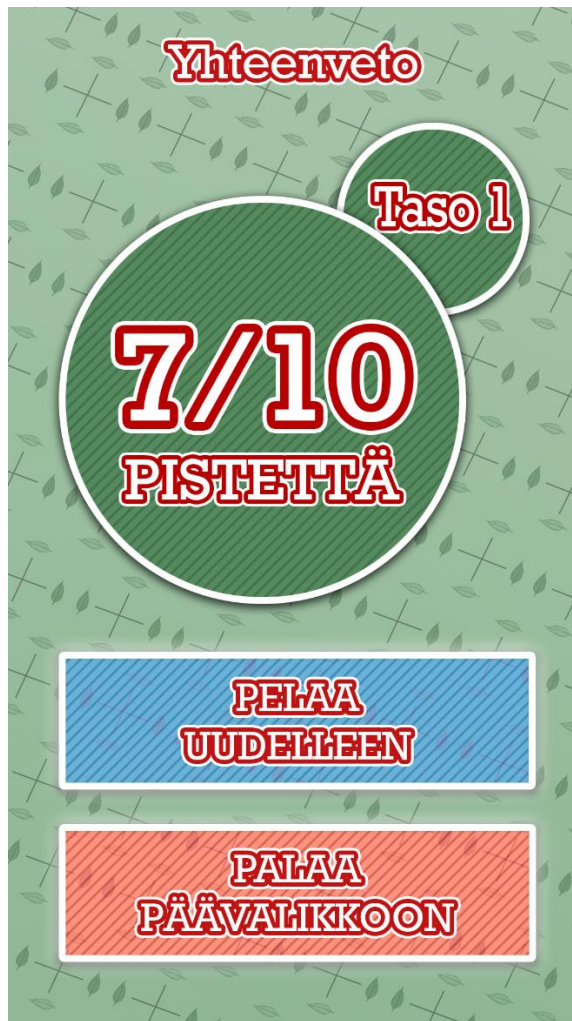
Seuraavaksi aloitin työstämään layoutia siitä, miltä pelinäköymä voisi näyttää. Tausta, napit ja fontit pysyisivät samoina näkymästä riippumatta. Pelinäköymässä isoin kysymys oli se miten arvattavan kasvin kuva näkyisi käyttäjälle. Tulin siihen lopputulokseen, että tunnistettavan kasvin olisi syytä olla jonkinlaisissa kehyksissä, koska en voinut alkaa leikata jokaista kasvin kuvaa taustastaan. Kokeilin aluksi neliömäisiä kehyksiä ja loin myös puuta muistuttavan materiaalin Photoshopissa, jota sitten yritin muuntaa puupintaiseksi kehykseksi. Nämä viritelmät eivät kuitenkaan toimineet ja olivat liian sekavia. Päädyin lopulta ratkaisuun, jossa kasvin kuva esitetään ympyränmuotoisessa kehyksessä. Ympyrän sivuille tuli myös kaksi pienempää ympyrää jotka näyttivät, monennessako kysymyksessä pelaaja on menossa ja senhetkiset pisteet (kuva 18).



Kuva 18. Havainnollistava kuva pelinäytön layoutista

Pelinäyttöön olisi tulossa vielä lisäominaisuuksia, joita tämänkaltaisissa staattisissa layouteissa ei oikein voi esittää. Esimerkiksi ideana olisi, että kysymysten välillä olisi jonkinlainen siirtymäanimaatio. Kuten myös ominaisuus, että kun käyttäjä painaa haluamaansa vastausvaihtoehtoa, näytölle pongahtaa jonkinlainen indikaattori osoittamaan menikö valinta oikein vai väärin.

Enää puuttui layout pelin lopettavalle ”palautenäytölle”, päätin hyödyntää samoja painikkeita, taustaa ja muotoilua myös tähän näyttöön. Tällä tavoin pelistä tulisi yhtenäisempi kokonaisuus. Näytön alalaidasta löytyy painikkeet tason uudelleen pelaamiselle tai vaihtoehto palata takaisin valikkonäyttöön (kuva 19). Palautenäytössä hyödynnettiin myös samaa ympyräteemaa, näyttämään pelaajalle montako pistettä hän onnistui keräämään ja mitä tasoa hän juuri pelasi.



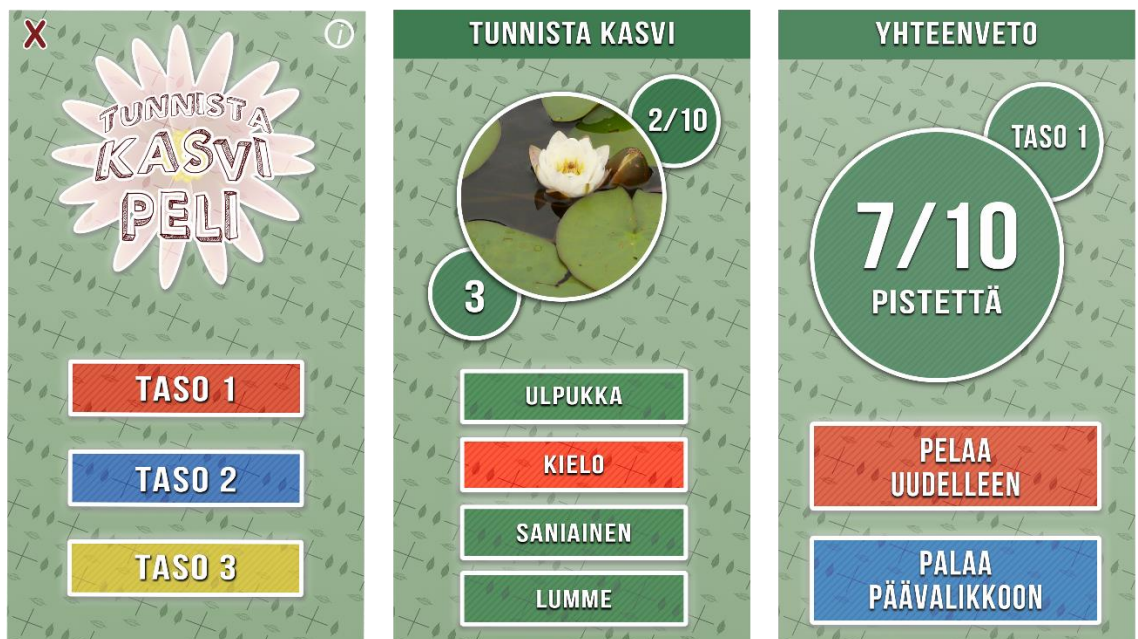
Kuva 19. Havainnollistava kuva palautenäytön layoutista

Huomasin näitä layouteja tehdessä, että kehittäessä peliä puhelimennäytölle täytyy todella ottaa huomioon, mitä näytöllä mahtuu näyttämään järkevästi. Esimerkiksi palautenäytössä olisi ollut hyödyllistä näyttää pelaajalle, mitkä kohdat hänellä menivät väärin. Yksinkertaisesti tila näytöllä oli niin rajattu, että sitä ominaisuutta ei pystytty mahdollistamaan mitenkään. Lopuksi lähetin layoutit toimeksiantajalla ja jäin odottamaan palautetta niistä.

Pian vastaanotin palautetta layouteista. Taustakuva, ympyrämuodot, painikkeet ja yleisilme olivat toimeksiantajan mieleen. Osaa fontteja, niiden tyyllittelyä ja värejä olisi kuitenkin syytä muuttaa selkeämpään suuntaan. Sanoma Pro on laatinut sähköisille oppimateriaaleille oman ohjeistuksen. Ohjeistus määrittelee reunaehdot minkälaiselle Sanoma Prolle tuotettavan sähköisen oppimateriaalin kuuluisi näyttää. Tutustuin ohjeistukseen tarkemmin ja koska eniten kritiikkiä layoutit olivat saaneet lähinnä fontista ja sen tyyllittelystä, päätin keskittyä lukemaan erityisen tarkasti nimenomaan tekstiin liittyviä ohjeita.

Ohjeistuksessa kerrottiin muun muassa, että fontin olisi syytä kuulua ”Sans-serif” -kirjasimiin ja fontin värin olisi syytä olla joko puhtaan valkoinen tai musta.

Layoutien muokkaaminen alkoi uuden fontin etsinnällä. Käytin tähän tarkoitukseen taas ”dafont.com”-sivustoa. Hakukriteereinä oli se, että uusi fontti kuuluisi ”Sans-serif” -kirjasimiin. Löysin fontin nimeltä ”Bebas Neue”, se miellytti silmää ja täytti Sanoma Pron ohjeistuksen asettamat reunaehdot. Vaihdoin fontin layouteissa joka kohtaan paitsi otsikon fontin jätin ennalleen, koska ohjeistuksessa oli määritetty, että otsikossa voi käyttää hieman erikoisempaakin fonttia. Korostin uutta fonttia vielä värjäämällä sen valkoisella ja luomalla siihen varjostuksen. Himmensin myös layouttien painikkeissa olevaa raidoitusta, jotta teksti näkyisi entistäkin paremmin (kuva 20).



Kuva 20. Paranneltu versio layouteista.

Yksi pyynnöistä toimeksiantajan puolelta oli, että pelinäytössä kaikki napit olisivat samanvärisiä ja kun käyttäjä painaa esimerkiksi väärää vastausta niin nappi muuttuu punaiseksi. Tämän pyynnön toteutin myös uusiin layouteihin (kuva 20). Oli hyvä asia, että Sanoma Pro tarjosi nopeaa palautetta, koska layoutit näyttivät nyt paljon paremmilta. Lopputuloksesta tuli selkeämpi, mikä palvelee myös paremmin kohderyhmää eli alakoululaisia.

Pelin ohjelmointi

Seuraavaksi alkoi pohtiminen, millaisilla työkaluilla kasvientunnistuspeli toteutettaisiin. Päätin myös, että aloitan pelin tekemisen omaan projektiinsa, se tuntui tässä vaiheessa helpoimmalta ratkaisulta. Tällä periaatteella ainakaan error-viestin sattuessa syy ei voi olla siinä, että pelissä oleva koodi ei vain toimi kasvien selailuosan koodin kanssa yhteen. Toisaalta nämä kaksi kokonaisuutta pitäisi kuitenkin jossain vaiheessa yhdistää, joten entä, jos pelin ollessa valmis tuleekin vastaan ikävä yllätys siitä, että kahden eri sovelluksen osan koodit ”riitelevät” keskenään.

Lopulta päädyin toteuttamaan sovelluksen Javascriptin/jQueryn, HTML5 ja CSS3 -kirjastojen avulla. jQuery on Javascriptiin löytyvä kirjasto, joka lisää ominaisuuksia tuttuihin Javascript-toimintoihin. Mielessäni kävi myös CreateJS-kirjaston hyödyntäminen, mutta sen idean hylkäsin, koska jQueryn toteutus oli itselle tutumpi vaihtoehto. Olin myös Sanomalla ollut mukana projektissa, jossa käytettiin jQuerya mobiilipelin toteuttamiseen, joten valinta tuntui luontaisemmalta.

Elementtien asettelu

Aloitin pelin ohjelmoinnin yrittämällä siirtää kaikki layout-kuvissa näkyvät elementit, omiin näkymiinsä koodin muodossa. Joten aluksi ”leikkasin” layouteista eri elementit, kuten napit ja taustat irti omiksi kuvatiedostoikseen. Elementit asettelin CSS-määritelmien avulla paikoilleen (kuva 21). Aloitin työskentelyn tunnistuspelin alkuvalikosta. Ensin lisättiin taustakuva CSS-puolella body-elementin sisään. Taustakuvan koko määriteltiin prosenteissa, jotta se skaalautuisi oikein, käyttäjän mobiililaitteen näytönkoosta riippumatta. Taustakuvan lisäämisen jälkeen lisättiin painikkeet, joita painamalla taso valittaisiin. Lopuksi näkymään lisättiin pelin otsikkokuva (kuva 21).

```
.menukuva{
    background-image: url("img/valikko_nappil.png");
    background-repeat: no-repeat;
    background-size: contain;
    width:550px;
    height:150px;
    padding-top:0;
    margin-top:0;
    margin-left: auto;
    margin-right: auto;
}
```



Kuva 21. Esimerkki yhden painikkeen määrittelystä CSS-tiedostossa, oikealla tuloste

Elementtien lisääminen sovellukseen oli suhteellisen suoraviivaista. Kun elementit olivat näkyvillä valikkonäkymässä, niihin piti lisätä toiminnallisuutta, tämä hoidettaisiin jQuery-koodin puolella. Elementteihin tuli myös ongelmia responsiivisuuden kanssa, vaihdettaessa näyttölaitteen kokoa, mutta tähän ongelmaan ja sen ratkaisuun palataan responsiivisuutta käsittelevässä osuudessa. Myös muut näkymät eli pelinäköymä ja palautenäköymä toteutettiin samalla tyylillä, tämän jälkeen aloin lisäämään toiminnallisuuksia ja siirtymiä näkymien välillä.

Siirtymien toiminnallisuus

Aloitin siirtymien luomisen muokkaamalla HTML-koodia, HTML-koodiin loin kaksi diviä, jotka nimesin "peli1" ja "peli2". Nämä divit olisivat kaksi eri näkymää, joiden välillä peli pyörisi. Tämä tapahtuisi siten, että toteuttaisin animaation kahden eri näkymän välille. Ennen animaatioiden tai muidenkaan ominaisuuksien ohjelmointia jQuery-koodin puolella, oli tärkeää määritellä muuttujia, joita käytettäisiin pelin ohjelmoinnin aikana. Näihin tärkeisiin muuttujiin kuului muun muassa kysymysten määrä, tasot, pisteet ja kysymysnumero. Animaation toteutin jQuery:n tarjoamalla animate-metodilla. Periaatteessa voisi siis kuvitella, että pelin valikkonäköymä on "peli1", kun käyttäjä painaa painiketta, esimerkiksi valitsee tason, animointi käynnistyy ja "peli1"-näköymä lähtee liukumaan vasemmalle. Kun näköymä on liukunut vasemmalle kokonaan näkyvistä, "peli2" divi tulee näkyville. Tässä vaiheessa siirtymä toimi vain yhden kerran, joten koodi kaipasi vielä parannusta.

```
function vaihdaKysymys() {
    kysymysNumero++;

    if(taso=="#pelil") {
        taso2="#pelil";
        taso="#pelil2";
    }else{
        taso2="#pelil2";
        taso="#pelil";
    }

    if(kysymysNumero<kysymystenMaara) {
        naytaKysymys();
    }else{
        displayFinalSlide();
    }

    $(taso2).animate({"right": "+=1000px"}, "slow", function() {
        $(taso2).css('right', '-1000px');
        $(taso2).empty();});
    $(taso).animate({"right": "+=1000px"}, "slow", function() {questionLock=false;});
}
```

Kuva 22. Siirtymiä vaihtava if-lause ja animointi.

Tähän auttoi if-lauseella toteutettu koodin osa, jossa luomaani muuttujaa nimeään "taso" käytetään apuna selvittäessä, onko käyttäjä, sillä hetkellä meidän divissä "pelil" vai "pelil2". Taso-muuttujia on kaksi kappaletta, taso-muuttuja ikään kuin tuo aina uuden näkymän tai kysymyksen ja muuttuja nimeään "taso2" poistaa tai siirtää pois nykyisen (kuva 22). Siirtymiä tulee niin monta kuin kysymyksiä yhdestä tasosta löytyy. Palautenäytön jälkeen taso nollataan, tämä takaa sen, että pisteet nollautuvat samalla, jotta käyttäjä voi aloittaa pelin uudestaan puhtaalta pöydältä. Taso-muuttujat sisältävät myös kaikki kolme näkymää, joiden välillä peli pyörii. Näkymät on toteutettu html-tekniikoilla, mutta jQuery-koodin puolella, tämän mahdollistaa jQuery:n append-metodi (kuva 23).

```
function displayMenu() {
    $(taso).animate({"right": "+=1000px"}, "slow", function() {questionLock=false;});

    $(taso).append('' +
        '<div class="menukuva"> <div class="valinta1">Luokat 1-2</div> </div>' +
        '<div class="menukuva2"> <div class="valinta2">Luokat 3-4</div> </div>' +
        '<div class="menukuva3"> <div class="valinta3">Luokat 5-6</div> </div>');
```

Kuva 23. Valikkonäkymän rakennus jQuery-koodin puolella.

Kun näkymät vaihtuivat oikein, oli aika tehdä vielä pieniä hienosäätöjä. Näihin hienosäätöihin kuului esimerkiksi se, että asetin siirtymien välille tuhannen

millisekunnin eli sekunnin viiveen, tämän ominaisuuden toteutin "setTimeout"-funktiolla. Aion myöhemmin lisätä peliin ominaisuuden, jonka avulla pelaaja näkee mobiilipäätteen näytöltä, menikö vastaus oikein vai väärin. Pieni viive seuraavaan kysymykseen siirtyessä varmistaa sen, että käyttäjä ehtii rekistroidymään saamansa palautteen. Tässä vaiheessa animoidut siirtymät toimivat hieman epämääräisesti, välillä ne pätkivät hieman eivätkä rullanneet näytön ohi sulavasti. Tämän kauneusvirheen pystyi korjaamaan lisäämällä css-tiedostoon transform-komennon, jonka arvoksi tuli "translate3d (0,0,0)". Määrittelemällä kaikki 3D-toiminnot nolliksi, ohjelma pystyy suorittamaan siirtymän nopeammin eli sulavammin. Tämä johtuu siitä, että ohjelman ei tarvitse jokaisella kerralla erikseen tutkia, onko kyseessä 2D vai 3D-animaatio. Kun siirtymät toimivat tyydyttävällä tavalla siirryin lisäämään kysymyksiä, vastauksia, pisteenlaskua ja kysymysten satunnaistamista.

Kysymysten tulostaminen

Kasvientunnistuspeli oli nyt siinä vaiheessa, että käyttäjä pystyi siirtymään pelin eri näkymien välillä painikkeita painamalla. Oli aika lisätä kysymykset eli kuva tunnistettavasta kasvista ja siihen liittyvät vastausvaihtoehdot. Kysymyksiä ja vastausvaihtoehtoja varten loin json-tiedoston, johon kysymykset varastoitaisiin omiin soluihinsa. Pelin jokaiselle kolmelle tasolle on oma json-tiedosto, josta kunkin tason kysymykset haetaan. Json-tiedostossa yksi kysymys vastausvaihtoehtoineen muotoiltiin siten, että ensimmäisellä rivillä oli kysymys. Kysymys tässä tapauksessa oli tiedostopolku tunnistettavan kasvin kuvaan. Kysymyslausekkeen alta json-tiedostossa löytyi vastausvaihtoehdot omilta riveiltään, vastausvaihtoehdot asetin siten, että ensimmäinen vastausvaihtoehto on aina oikea vastaus kysymykseen (kuva 24). Myöhemmin jQuery-koodin puolella nämä vastausvaihtoehdot sekoitettaisiin keskenään, jotta itse pelissä oikea vastaus ei aina olisi ylimmäisenä oleva vaihtoehto. Json-tiedoston kautta peliin voi myös myöhemmin lisätä lisää kysymyksiä halutessaan.

```

{"taso2": [
  {
    "question": "../img/kasvit_taso2/ahvenvita.png",
    "option1": "Ahvenvita",
    "option2": "Maitohorsma",
    "option3": "Pujo",
    "option4": "Kataja"
  },
  {
    "question": "../img/kasvit_taso2/harakankello.png",
    "option1": "Harakankello",
    "option2": "Kurjenmiekka",
    "option3": "Ohra",
    "option4": "Puna-apila"
  },

```

Kuva 24. Esimerkki kahdesta pelin kysymyksestä json-tiedostossa, ensimmäinen vaihtoehto on oikea vastaus.

Kun kysymykset oli varastoitu json-tiedostoon, ne piti hakea jQuery-koodin puolelle, jotta niitä voisi käsitellä paremmin. Tämä aloitettiin hyödyntämällä taas kerran jQuery-kirjaston metodeja, tällä kertaa metodia nimeltä "getJSON". Nimensä mukaisesti tämä kyseinen metodi noutaa AJAX-menetelmiä hyödyntäen json-tiedoston (kuva 25). Kun json-tiedosto oli noudettu, se piti muuttaa "arrayksi" eli taulukoksi, tyylillä jossa kysymys ja vastausvaihtoehdot olisivat erillisissä soluissa, jotta niitä voisi käsitellä erillisinä osina myöhemmässä vaiheessa. Muutin json-tiedoston solut taulukoksi, for-loopin sisällä, tällä vältettiin se, ettei jokaista json-tiedoston solua tarvinnut alkaa erikseen muuttamaan, vaan ohjelma kävi läpi koko json-tiedoston solut kerrallaan. Kun kaikki json-tiedoston kohdat oli muutettu taulukkomuotoon, ne piti sekoittaa keskenään, jotta kysymykset eivät jokaisella pelikerralla tulisi samassa järjestyksessä. Taulukon satunnaistin käyttämällä apuna Javascriptin sort- ja random - metodeja (Kuva 25). Nämä samat menetelmät toistettiin jokaisen pelin tason kohdalla, eli yhteensä kolme kertaa.


```
$.getJSON('json/taso3.json', function(data) {
    for(i=0;i<data.taso3.length;i++){
        kysymykset3[i]=new Array;
        kysymykset3[i][0]=data.taso3[i].question;
        kysymykset3[i][1]=data.taso3[i].option1;
        kysymykset3[i][2]=data.taso3[i].option2;
        kysymykset3[i][3]=data.taso3[i].option3;
        kysymykset3[i][4]=data.taso3[i].option4;
    }
    kysymykset3.sort(function() { return 0.5 - Math.random(); });
});
```

Kuva 25. Json-tiedoston tuonti, muuttaminen taulukoksi ja taulukon satunnaistaminen. Esimerkkikuvassa pelin kolmas taso.

Kun json-tiedoston sisältö oli purettu taulukkoihin, oli aika syventyä siihen, miten yksi kysymys ja sen vastausvaihtoehdot käsitellään pelinäköymässä ja miten peli siirtyy seuraavaan kysymykseen. Pelinäköymässä elementit, kuten painikkeet, pisteet ja kysymysten määrä olivat jo näkyvillä layout suunnitelman mukaisesti (kuva 20), vastauspainikkeet olivat vain tyhjiä tässä vaiheessa. Jotta pelistä tulisi vaikeampi ja pelillisempi, myös vastausvaihtoehtojen paikat tulisi satunnaistaa. Eli periaatteessa tämä tarkoitti sitä, että kun käyttäjä käynnistää valitun tason kysymysten järjestyksen lisäksi ohjelma satunnaistaa sen missä järjestyksessä vastausvaihtoehdot tulostuvat, joten oikea vastaus ei aina olisi esimerkiksi ylin tai alin vastausvaihtoehtopainike. Tätä ominaisuutta varten tarvittiin apumuuttujaa, jonka nimeksi tuli "rnd", lyhenne englannin sanasta random eli satunnainen.

Javascriptin "Random"-matematiikkafunktion avulla määriteltiin rnd-apumuuttuja valitsemaan jokin luku yhden ja kolmen väliltä tai myöhemmissä tasoissa yhden ja neljän väliltä. Tämä satunnainen luku pyöristettiin ceil-matematiikkafunktion avulla, jotta luku nolla ei olisi mahdollinen tulos. Satunnainen luku ohjaa rnd-muuttujan yhteen if-lause ryppäaseen, jossa vastausvaihtoehdot on määritetty satunnaiseen järjestykseen. Periaatteessa siis pelin ensimmäisellä tasolla vastausvaihtoehtojen paikat vaihtelevat kolmen eri muodostelman välillä ja loppuissa tasoissa neljän eri muodostelman välillä (kuva 26).


```

function naytaKysymys() {
    var rnd=Math.random()*3;
    rnd=Math.ceil(rnd);
    var q1;
    var q2;
    var q3;

    if(rnd==1){
        q1=kysymykset[kysymysNumero][1];
        q2=kysymykset[kysymysNumero][2];
        q3=kysymykset[kysymysNumero][3];
    }
    if(rnd==2){
        q2=kysymykset[kysymysNumero][1];
        q3=kysymykset[kysymysNumero][2];
        q1=kysymykset[kysymysNumero][3];
    }
    if(rnd==3){
        q3=kysymykset[kysymysNumero][1];
        q1=kysymykset[kysymysNumero][2];
        q2=kysymykset[kysymysNumero][3];
    }
}

```

Kuva 26. Ohjelman osa, joka sekoittaa vastausvaihtoehtojen järjestyksen rnd-muuttujan avulla.

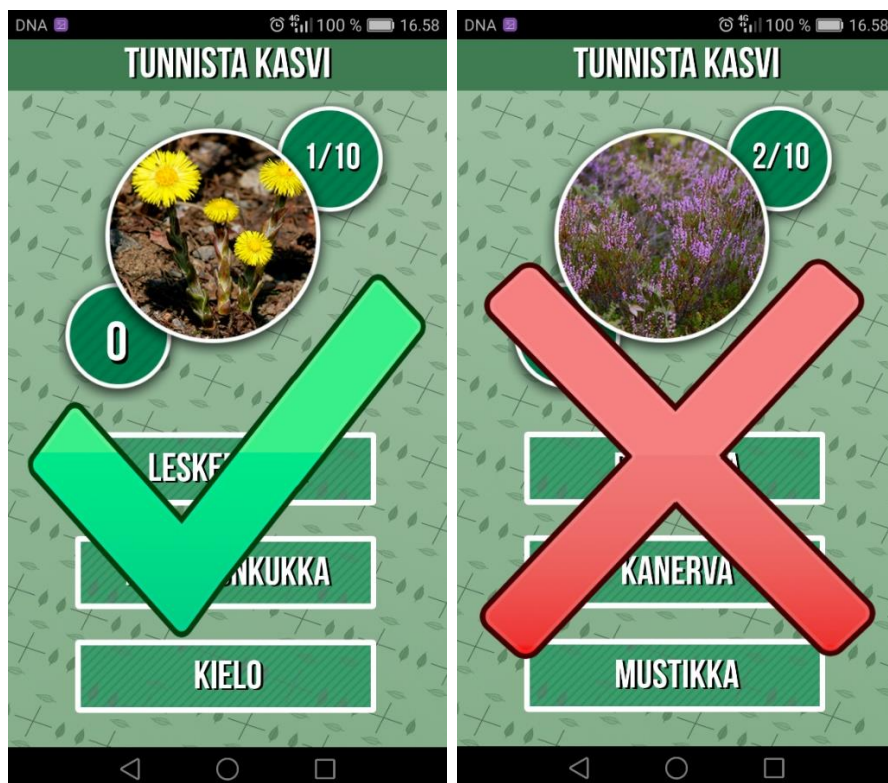
Kun vastausvaihtoehdot oli satunnaistettu, piti varmistaa, että ohjelma reagoisi oikein sen jälkeen, kun käyttäjä on painanut haluamaansa vastausta. Tämä tarkoitti sitä että, kysymys olisi syytä ikään kuin lukita sen jälkeen, kun käyttäjä on valinnut vastauksensa. Lukituksen hoidin apumuuttujalla ja jQueryn klikki-käsittelijällä. Painalluksen piti kertoa ohjelmalla, että on aika siirtyä eteenpäin ja että käyttäjä ei enää pystyisi vaihtamaan vastaustaan. Tässä vaiheessa ohjelman pitäisi myös tarkistaa käyttäjän antama vastaus, tämä tapahtui hyödyntämällä edellisessä kappaleessa mainitsemaani rnd-muuttujaa. Nimittäin jos rnd-muuttujan id-arvo vastasi vastausvaihtoehtopainikkeelle määriteltyä id-arvoa, vastaus menee oikein, jos taas id-arvot eivät vastaa, vastaus on väärin.

Pelin loppu ja pelillisyyys

Nyt kun kysymykset tulostuivat oikein ja siirtymä seuraaviin kysymyksiin toimi moitteettomasti, oli jäljellä vielä luoda pelille layouttien mukainen palautenkymä (kuva 20). Elementtien asettelu hoitui samaan tyyliin kuin muissakin näkymissä, palautenäytön elementteihin kuuluu muun muassa pisteiden esittäminen ja painike, jonka avulla käyttäjä pääsee takaisin päävalikkoon. Pelillinen rakenne oli nyt valmis, aion nyt käydä läpi pelin kulkua, sen eri vaiheita ja pelillisyyttä.

Eli pelin käynnistyessä käyttäjä kohtaa ensimmäisenä valikkonäkymän (kuva 20). Valikosta löytyy kolme tasoa, jotka on nimetty luokka-asteiden mukaisesti. 1–2. luokkalaisille suunnatussa tasossa on kolme vastausvaihtoehtoa, kahdessa muussa tasossa vastausvaihtoehtoja on neljä kappaletta. Ensimmäisessä tasossa myös vastausvaihtoehdot ovat helpotettuja verrattuna myöhempiin tasoihin, esimerkiksi jos tunnistettava kuva on havupuu, niin kaikki muut väärät vastausvaihtoehdot eivät välttämättä ole muita havupuita, vaan oikea vastaus on helpommin löydettävissä. Tämä oli yksi pelillisistä asioista mitä työn aikana joutui pohtimaan, käyttäjien ikä ja tiedon määrä ovat kuitenkin aika suuria alakoulun luokka-asteiden välillä.

Jokaisessa tasossa on kymmenen kysymystä, jotka satunnaistetaan useamman kysymyksen joukkiosta, joten eri pelikerroilla kysymyksiin tulee vaihtelua. Lisäsin myös oikein ja väärin ”ikonit” pelinäkömään, siinä kohtaa missä käyttäjä painaa haluamaansa vastausta (kuva 27).



Kuva 27. Valmis pelinäkömä toiminnassa puhelimen näytöllä, kuvissa havainnollistettu pelin antama palaute, kun käyttäjä on valinnut vastausvaihtoehdon.

Ikonit antavat äänen kera käyttäjälle välittömän palautteen siitä, menikö vastaus väärin vai oikein. Pelillisyyden puolesta mietin myös pitkään pitäisikö käyttäjälle paljastaa vastaamisen yhteydessä mikä olisi ollut oikea vaihtoehto,

mutta päätin jättää tämän ominaisuuden pois koska se olisi voinut paljastaa tulevien kysymyksien vastauksen etukäteen.

Responsiivisuus

Tässä luvussa käyn läpi, miten tunnistuspelin sai sovitettua tukemaan kaikkia yleisimpiä mobiililaitteiden näyttökokoja. Koska en käyttänyt pelin ohjelmoinnissa mitään valmista mobiiliohjelmointiin suunnattua frameworkia tai kirjastoa niin responsiivisuudesta muodostui yksi tunnistuspelin työteliäimmistä osuuksista. Aloitin tekemällä päätöksen siitä, että peliä pelattaisiin vain "portrait" eli puhelimen ollessa pystyasennossa, tämä johtui lähinnä tunnistettavien kasvien esitystavasta, joka ei olisi mielestäni toiminut visuaalisesta näkökulmasta vaakatasossa pelattaessa. Käytännössä tämän ominaisuuden toteutin muokkaamalla Android-manifestiksi kutsuttua XML-tiedostoa, tiedostoon lisätään pari tekstiriviä, jotka varmistavat, ettei sovellus skaalaudu vaakasuuntaan.

Responsiivisuuden ja elementtien skaalautumisen toteutin CSS-tiedoston määrittelyillä. Kasvientunnistuspeli tukee kaikki yleisimpiä puhelinten ja tabletien näyttökokoja, siitä isompia näyttöjä kuten tietokoneen monitoria tunnistuspeli tai kasvisovellus ei tue kunnolla, koska päämääränä oli luoda mobiilisovellus. Jättämällä tietokonemonitorin kaltaiset isommat näytöt välistä, sovelluksessa olevista kuvista saatiin pienempiä, näin ollen myös tiedostokoko supistui.

Responsiivisuus siis muodostui hyödyntämällä CSS-ominaisuutta nimeltään "@media"-sääntöä. Media ominaisuuden avulla voi määritellä CSS-tiedostoon erikseen asetuksia tukemaan eri näyttökokoja, media ominaisuus ottaa huomioon näytön pituuden, leveyden, resoluution ja orientaation. Media ominaisuus linkitetään johonkin tiettyyn pikselimäärään, esimerkiksi maksimileveys tietyssä näkymässä on 400 pikseliä, tämän kaltaisen linkityksen avulla voi luoda siis kaikille laitteille jonka näytön leveys on maksimissaan 400 pikseliä omat määrittelyt (kuva 28). Luomalla näitä määrittelyksiä tasaisin pikselivälein, pystyin varmistamaan, että pelin elementit skaalautuvat näytön koosta riippumatta oikein. Mitä isompi näyttö, niin sitä isommaksi elementtejä piti skaalata.

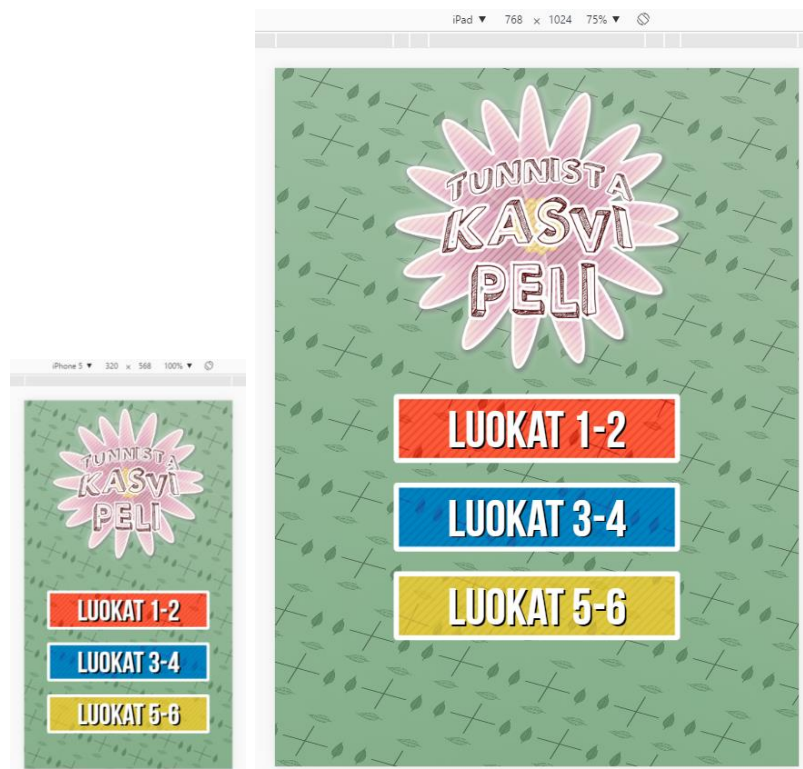
```

@media screen and (max-width:400px) {
  .menukuva{
    background-image: url("img/valikko_nappil.png");
    background-repeat: no-repeat;
    background-size: contain;
    width:270px;
    height:80px;
    padding-top:0;
    margin-top:0;
  }
}

```

Kuva 28. Esimerkki yhdestä näyttökoon määrittelystä, tässä tapauksessa maksimi pikselimäärä vaakasuuntaan 400, kuvassa myös valikon painike-elementin määrittely näyttökoolle sopivaksi.

Määrittelyprofiileja luodessa ja testatessa oli tärkeää nähdä, mille sovellus näytti missäkin näyttökoossa. Koska itseltäni löytyy vain rajattu määrä laitteita erikokoisilla näytöillä, niin käytin apuna Google Chrome -selaimen tarjoamaa "Device Mode" -tilaa (kuva 29).



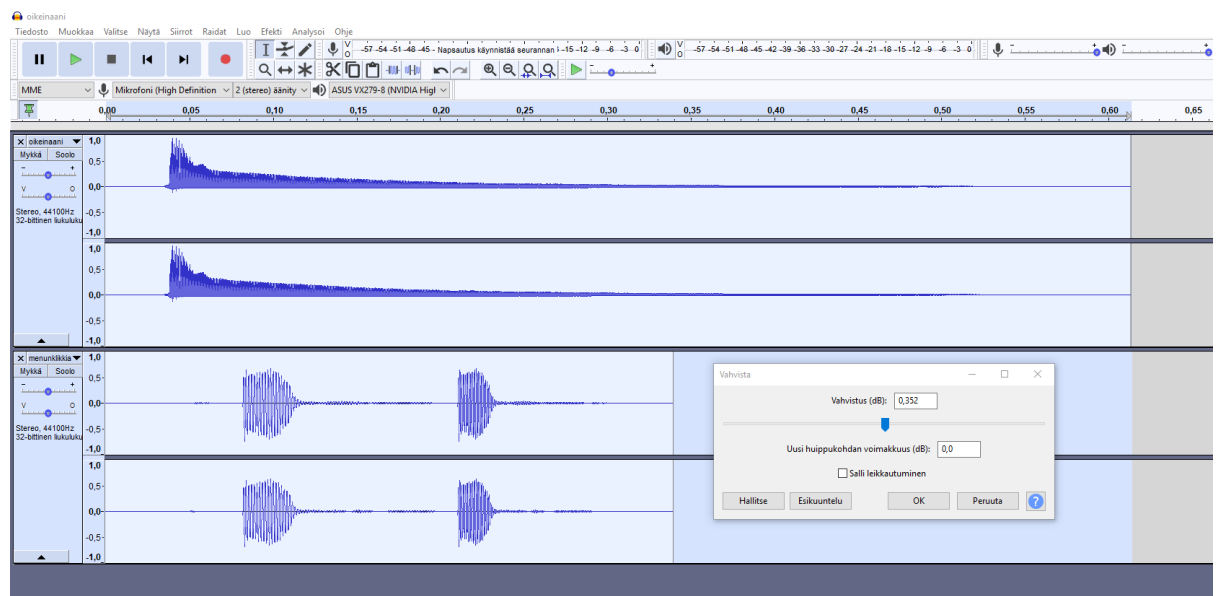
Kuva 29. Google Chromen Device Mode -näkymä. Vasemmalla sovellus skaalattu iPhone 5 näyttökoolle ja oikealla iPad-tabletille. Elementit ja fontit skaalautuvat näyttökoon mukaisesti.

Device Moden avulla voi simuloida, mille sovellus näyttää milläkin näyttökoolla ja laitteella. Device Modeen on valmiiksi lisätty kaikki yleisimmät puhelimet ja tabletit suoraan, mutta myös oman näyttökoon voi lisätä halutessaan. Eli kun

olin tehnyt CSS-tiedoston koodiin haluamani määrittelyn, pystyin Chromen kautta tarkistamaan, mille muutokset näyttäisivät tietyssä näyttökoossa.

Äänet

Äänitehosteita lisäsin valikkojen painikkeisiin ja kohtaan missä käyttäjä painaa vastausvaihtoehtoa kertomaan visuaalisen palautteen lisäksi menikö vastaus oikein vai väärin. Oikein-ääni kuulostaa iloiselta ja heleältä kilahdukselta kun taas väärän vastauksen ääni on mollivoittoisempi. Äänitehosteet etsin ”free-sound.org”-nimiseltä verkkosivustolta. Löysin sivustolta kolme äänitehostetta, joita muokkasin itse myöhemmin käyttötarkoitukseeni paremmin soveltuviksi. Varmistin myös, että valitsemani äänitiedostot olivat tarjolla ”Creative Commons 0” -lisenssin alla, jotta niitä sai vapaasti käyttää omissa projekteissaan tai vaikka kaupalliseenkin käyttöön. Seuraavaksi muokkasin äänitiedostoja Audacity-ääniohjelmalla. Audacity on ilmainen äänenkäsittelyohjelma. Audacitylla leikkasin äänitiedostot sopivan pituisiksi, vahvistin ääniä eri tehosteilla ja muunsin ne mp3-tiedostoiksi (kuva 30).



Kuva 30. Havainnekuva äänenkäsittelystä Audacity-ohjelmassa, kuvassa ääniraidan vahvistus.

Kun äänet oli tallennettu mp3-tiedostoiksi, siirsin ne ohjelmakoodin puolelle käyttäen apuna HTML5:den tarjoamaa ”Audio”-objektia. Äänet voivat tuntua pieneltä osalta kokonaisuutta peleissä, mutta äänet lisäävät peliin immersiota ja auttavat palautteen välittämisessä käyttäjälle.

5 PÄÄTÄNTÖ

Tässä luvussa reflektoidaan opinnäytetyöprosessin onnistumista, toimeksiantajan antamaa loppupalautetta ja kehitysideoita joilla lopputulosta olisi voinut mahdollisesti parantaa. Luvussa käsitellään myös tiettyjä sovelluksen ominaisuuksia, joita olisi voinut toteuttaa paremmin.

Opinnäytetyö onnistui kokonaisuutena hyvin. Toimeksiantajan asettamat vaatimukset täyttyivät suurten linjojen osalta hyvin. Näitä olivat muun muassa mobiililaitteilla toimiminen, kasvien selailu, nimihaku, tunnistuspeli ja luokkasteittain jaottelu. Työskentely toimeksiantajan kanssa sujui moitteettomasti ja jatkoimme yhteydenpitoa koko prosessin läpi. Toimeksiantajalta sain hyviä neuvoja sovelluksen kehittämiseen ja parantamiseen. Lähetin myös prosessin aikana päivityksiä sovelluksen sen hetkisestä tilasta toimeksiantajalle. Sano-ma Pro myös kertoi loppupalautteessaan olevansa tyytyväisiä suunnittelutyöhön, sovelluksen selkeyteen sekä aktiiviseen palautteenhakuun. Sovelluksen lopputulosta toimeksiantaja kommentoi myös varsin hyväksi.

Projektin aikana kehitysideoita tuli vastaan useita, osaa niistä ei voinut toteuttaa perustellusti, esimerkiksi siitä syystä, että mobiililaitteen näytön koko on niin rajattu. Isoimpina kehityskohtina mieleen jäi kasvientunnistuspelin ominaisuuksien rajallisuus. Pelin lopputulos oli toimeksiantajan vaatimusten mukainen, mutta peliin olisi voinut lisätä lisäominaisuuksia, kuten pisteiden tallennuksen tai aikarajan tason suorittamiseen.

Projektissa tuli myös vastaan asioita, jotka oli suunniteltu alun perin toteutettaviksi toisella tavalla, mutta esiintyvät lopputuloksessa hieman eri tavalla. Huomattavimpana näistä oli sovelluksen jakaminen kahteen osaan. Nimittäin alkuperäisissä suunnitelmissa kasvienselailuosan ja kasvientunnistuspelin oli tarkoitus olla yhdessä applikaatiossa. Tästä ideasta jouduin kuitenkin luopumaan, koska kaksi osa-aluetta hyödynsi toteutuksessaan erilaisia tekniikoita, joten toteutukset eivät menneet hyvin yhteen, kun lopulta yritin niitä yhdistää. Näin ollen lopputuloksena syntyi kaksi erillistä sovellusta: kasvienselailuosa ja kasvientunnistuspeli. Tämä ei onneksi vaivannut toimeksiantajaa, sillä nämä kaksi osa-aluetta erosivat toisistaan kuitenkin niin mittavasti ominaisuuksiltaan. Myös kasvientunnistuspeliin ei loppujen lopuksi tullut kaikkia samoja

elementtejä, joita etukäteen luoduilla layouteilla oli suunniteltu. Näihin lukeutui muun muassa palautenäytön "pelaa uudelleen" -painike ja palautenäytön ikkuna jossa käyttäjä näkee minkä tason juuri pelasi. Projektin ulkoasuun ja yleiseen toteutukseen olin myös itse tyytyväinen, vaikka parantamisen varaa tietyin paikoin jäikin.

Kokonaisuutena opinnäytetyöprosessi oli erittäin työntäyteinen ja opettavainen. Osa käytetyistä tekniikoista oli entuudestaan tuttuja, mutta opinnäytetyössä riitti silti haasteita riittämiin. Oli opettavaista tehdä toimeksiannon mukainen sovellus oikealle alan yritykselle ja käydä keskusteluja toimeksiantajan kanssa työskentelyn lomassa. Opin paljon projektien kehittämisestä mobiilialustoille, tietojen varastoinnista json-tiedostoihin ja myös pelisuunnittelusta opetuskäyttöön. Opinnäytetyön raportointi myös harjaannutti kirjallista osaa-mistani, kuten myös asiatekstien luomisen taitoa.

LÄHTEET

Anat, B. 2014 "Ludwig Wittgenstein", The Stanford Encyclopedia of Philosophy. WWW-dokumentti. Saatavissa:

<https://plato.stanford.edu/entries/wittgenstein/> [viitattu 16.12.2017].

AngularJS. 2015. AngularJS Documentation. WWW-dokumentti. Saatavissa:

<https://docs.angularjs.org/guide> [viitattu 27.11.2017].

Cheek, C. 2015. Integrating Health Behavior Theory and Design Elements in Serious Games. WWW-dokumentti. Saatavissa:

<https://mental.jmir.org/2015/2/e11/> [viitattu 15.12.2017].

Cordova Documentation. 2015a. Android Platform Guide. WWW-dokumentti. Saatavissa:

<https://cordova.apache.org/docs/en/latest/guide/platforms/android/> [viitattu 23.11.2017].

Cordova Documentation. 2015b. Create your first Cordova app. WWW-dokumentti. Saatavissa:

<https://cordova.apache.org/docs/en/latest/guide/cli/index.html> [viitattu 23.11.2017].

Cordova Documentation. 2015c. Introduction. WWW-dokumentti. Saatavissa:

<https://cordova.apache.org/docs/en/latest/guide/overview/index.html> [viitattu 23.11.2017].

Gowell, S., McWherter, J. 2012. Professional Mobile Application Development. John Wiley & Sons, Incorporated. E-kirja. Saatavissa:

<http://ebookcentral.proquest.com.ezproxy.xamk.fi:2048/lib/xamk-ebooks/detail.action?docID=843643> [viitattu 20.11.2017].

Heard, P. 2017. How to choose the best mobile app development framework. WWW-dokumentti. Saatavissa:

<https://www.logicroom.co/how-to-choose-the-best-mobile-app-development-framework/> [viitattu 20.11.2017].

Ionic. 2016. Ionic Documentation. WWW-dokumentti. Saatavissa:

<https://ionicframework.com/docs/v1/guide/preface.html> [viitattu 7.12.2017].

Kangas, M., Koskinen, A., Krokfors, L. 2014. Oppiminen pelissä: pelit, pelillisyyys ja leikillisyyys opetuksessa. E-kirja. Saatavissa:

<https://kaakkuri.finna.fi/Record/kaakkuri.217429> [viitattu 5.1.2018].

LeRoux, B. 2012. PhoneGap, Cordova, and what's in a name? WWW-dokumentti. Saatavissa:

<https://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name/> [viitattu 23.11.2017].

Lyytinen, T. 2013. HTML5 web-sovellukset. WWW-dokumentti. Saatavissa:

<https://www.vincit.fi/blog/html5-web-sovellukset/> [viitattu 27.11.2017].

Malykhina, E. 2014. Fact or Fiction? : Video Games Are the Future of Education. WWW-dokumentti. Saatavissa:

<https://www.scientificamerican.com/article/fact-or-fiction-video-games-are-the-future-of-education/> [viitattu 17.1.2018].

Marisi, A. 2015. Angular Expressions. WWW-dokumentti. Saatavissa: <http://angularfirst.com/angular-expressions-2/> [viitattu 27.11.2017].

Mononen, M. 2003. Pelisuunnittelu, Tietokonepelisuunnittelijan näkökulmasta. WWW-dokumentti. Saatavissa: <http://www.pingstate.nu/omnilayer/yksi/media/552/pelisuunnittelu1.html> [viitattu 13.12.2017].

Mylläri, J., Vesterinen, O. 2014. Oppiminen pelissä: Peleistä pelillisyyteen. E-kirja. Saatavissa: <https://kaakkuri.finna.fi/Record/kaakkuri.217429> [viitattu 6.1.2018].

Opetushallitus. 2016. Edu.fi - Oppimispelit opetuksessa. WWW-dokumentti. Saatavissa: http://www.edu.fi/ammattikoulutus/digitaalisuus_ohjauksessa_%20ja_opetuksessa/oppimispelit/oppimispelit_opetuksessa [viitattu 14.1.2018].

Panda, S. 2014. A Practical Guide to AngularJS Directives. WWW-dokumentti. Saatavissa: <https://www.sitepoint.com/practical-guide-angularjs-directives/> [viitattu 27.11.2017].

Puentedura, R. 2009. An Introduction to Educational Gaming, What is a game? PDF-dokumentti. Saatavissa: http://hippasus.com/resources/gameandlearn/slides/1_WhatIsAGame.pdf [viitattu 17.12.2017].

Rabkina, V. 2015. 5 Ways to add gamification to your website. WWW-dokumentti. Saatavissa: <https://rubyroidlabs.com/blog/2015/10/gamification/> [viitattu 21.1.2018].

Renaux, J. 2017. Ionic 2 vs. Ionic 1: Performance Gains, New Tools, and a Big Step Forward. WWW-dokumentti. Saatavissa: <https://www.toptal.com/ionic/ionic-1-vs-ionic-2-key-differences> [viitattu 10.12.2017].

Ristola, T. 2016. Kuinka toteutustapa vaikuttaa mobiilisovelluksen käytön sujuvuuteen. WWW-dokumentti. Saatavissa: <https://gofore.com/toteutustapa-vaikuttaa-mobiilisovelluksen-kayton-sujuvuuteen/> [viitattu 20.11.2017].