Ville Hurme

Lääkekasvatuspeli

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Media Engineering-koulutusohjelma

Insinöörityö

8.5.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Ville Hurme Lääkekasvatuspeli 23 sivua 8.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Media Engineering
Ohjaaja	Yliopettaja Kari Salo

Insinöörityön tarkoituksena oli suunnitella ja toteuttaa lääkekasvatuspeli osana Metropolia Game Studion projektia. Projekti toteutettiin lääkealan turvallisuus- ja kehittämiskeskus Fimealle.

Projektissa selvitettiin ensin, mitä sovelluskehystä pelin tekemiseen käytettäisiin, ja vertailtiin kahta eri vaihtoehtoa. Koska pelin täytyi toimia selainympäristössä, ohjelmointikielenä oli JavaScript. Sovelluskehyksenä päädyttiin käyttämään Phaseria, joka toimii mainiosti pienten 2D-pelien tekemiseen web-ympäristöön JavaScriptillä.

Peli suunniteltiin helppokäyttöiseksi sen kohderyhmän takia, ja se sisälsi harjoitustason (tutorial) pelin jokaiselle eri vaikeusasteelle. Apuna pelissä oli myös pelaamista ohjaava lääkäri-hahmo, joka neuvoi lääkkeiden, potilaiden ja pelin eri toimintojen kanssa.

Insinöörityö täytti asiakkaan vaatimukset ja samalla opetti työn toteuttajia eri sovelluskehyksien valinnassa ja selainympäristöön ohjelmoitavien pelien tekemisessä. Työ on pelattavissa Fimean lääkekasvatukseen tarkoitetulla verkkosivuilla http://www.laakekasvatus.fi/.

Avainsanat	cloud9, Phaser, lääkekasvatuspeli, Fimea

Author(s) Title Number of Pages Date	Ville Hurme Fimeas medicines education game 23 pages 8. May 2017
Degree	Bachelor of engineering
Degree Programme	Media Engineering
Specialisation option	Mobile Programming & .NET Application Development
Instructor(s)	Kari Salo, Principal Lecturer

The purpose of this study was to plan and create a medicines education game as part of a Metropolia Game Studio project. The project was done for Finnish Medicines Agency Fimea.

First, we had to decide which of the two possible frameworks was the more suitable one for this project. Because the game had to work in a web environment, it was coded in JavaScript. Phaser was chosen as the framework, because it is good for coding small 2D games for web environment with JavaScript.

The game was designed to be easy to use because of its target group and it consisted of a tutorial for each possible difficulty in the game. As additional help there was also a doctor character who helped with drugs, patients and different functionalities of the game.

The thesis project fulfilled the customers needs. Also, it taught the creators of this project a lot about choosing different frameworks as well as coding games for web environments. The game is still available at Fimeas website dedicated for medicines education. http://www.laakekasvatus.fi/

Keywords	cloud9, Phaser, medicines education game, Fimea

Sisällys

1 Johdanto

Insinöörityön tarkoituksena on luoda lääkekasvatuspeli, jonka avulla pelin tilannut asiakas pystyisi jakamaan tietoutta erilaisista lääkevalmisteista ja niiden oikeaoppisesta käytöstä. Pelin kohderyhmäksi on määritelty 3.–6.-luokkalaiset nuoret. Pelin on tilannut Metropolia Game Studiosilta lääkealan turvallisuus- ja kehittämiskeskus Fimea, jonka pyrkimyksenä on pelin avulla lisätä nuorten lääkekasvatusta mielenkiintoisella ja nuorten houkuttelevaksi kokemalla tavalla. Lääkekasvatuksella pyritään ohjaamaan oikeiden itsehoitovalintojen tekemiseen ja ehkäisemään lääkkeiden väärinkäyttöä. Lääkekasvatusta on perinteisesti annettu kouluissa päihdekasvatuksen yhteydessä, mutta sen merkityksen tunnistamisen myötä lääkekasvatuksen määrää on viime vuosina lisätty ja sitä on pyritty erottamaan päihdekasvatuksesta omaksi opetusalueekseen. [1.]

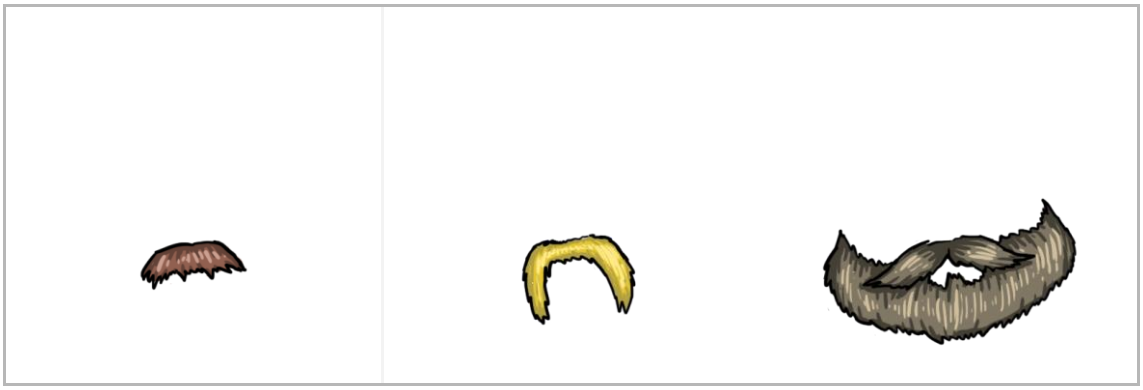
Fimean tilauksen mukaisen pelin kehitykseen on saatavilla useita eri työkaluja. Pelin rakenne on muotoutunut alustavissa suunnitteluvaiheissa melko yksinkertaiseksi, ja sen tekemiseen on tärkeää valita juuri tämäntyyppiselle, vähemmän kompleksiselle pelille sopivat työkalut. Jotkin saatavilla olevista vaihtoehdoista olisivat olleet tämän tyyppiselle pelille turhan monimutkaisia, vaikka laajemmissa kokonaisuuksissa niiden muokkausmahdollisuuksista onkin merkittävää etua. Työkaluratkaisuissa on kuitenkin oltava tarkkana myös siinä, etteivät valitut työkalut olisi projektille alimitoitettuja ja vaatisi turhia resursseja toteutukseen.

2 Lääkekasvatuspelin suunnittelu

Insinöörityön asiakas halusi lääkekasvatuspelin olevan käyttöliittymältään tarpeeksi helppo kohderyhmälle, mutta pyrkimyksenä oli myös toteuttaa erilaiset sairaustapaukset ja lääkkeiden toiminta pelissä mahdollisimman realistisesti. Tämä oli hyvin tärkeää, sillä pelin tarkoituksena oli opettaa nuoria toimimaan oikein sairaustapauksissa ja valitsemaan oikeat itsehoitolääkkeet. Koska pelin käyttöliittymästä haluttiin yksinkertainen, ehdotettiin asiakkaalle point&click-tyyppistä peliä, jossa pelaaja käyttää pelkästään hiirtä eri toimintojen tekemiseen. Peli suunniteltiin selainpohjaiseksi ja yhteensopivaksi kaikkien modernien selaimien kanssa, sillä sen oli tarkoituksena tulla esille Fimean lääkekasvatukseen tarkoitetuille verkkosivuille.

Projektiryhmässä työskenteli viisi henkilöä: projektin johtaja, pelisuunnittelija, graafikko ja kaksi ohjelmoijaa. Itse toimin projektissa toisena ohjelmoijana, ja koska minulla on kokemusta erityisesti JavaScript-kielellä ohjelmoimisesta, otin ryhmässä vastuulleni ominaisuuksien rakenteen suunnittelun ja toteutuksen, josta tuli insinöörityöni. Projektin työstämiseen oli varattu rajallinen määrä tunteja kuukaudessa, joten sovimme työryhmässä, milloin olisimme paikalla samaan aikaan keskustelemassa pelin ohjelmoinnin ongelmista ja ratkaisuista.

Pelin grafiikan täytyi olla nuorelle kohderyhmälle kiinnostavan näköistä. Graafikko suunnitteli ja toteutti mallikappaleet grafiikasta, minkä jälkeen se hyväksytettiin asiakkaalla. Grafiikasta päätettiin tehdä sarjakuvamaiset, mutta sen tuli myös kuvata ihmistä tarpeeksi realistisesti, jotta se visualisoisi tauteja ja niiden oireita tarpeeksi tarkasti. Asiakas piti pelille suunniteltua graafista ilmettä hyvänä, ja tehdyt mallikappaleet otettiin heti käyttöön projektissa (kuva 1).



Kuva 1. Esimerkki varhaisesta grafiikasta, joka sittemmin jätettiin pois pelistä, koska se peitti oireiden visualisoinnit.

3 Peliohjelmoinnin tekniikat ja teknologiat

3.1 Työkalujen valinta

Pelin tekemiseksi vertailtiin eri sovelluskehyksiä, joilla peli olisi mahdollista toteuttaa. Vaihtoehtoja rajoitti se, että peli oli tarkoitus toteuttaa selainympäristössä. Sopivan sovelluskehyksen valinta projektille oli tärkeää pelin toteuttamisen kannalta. Jo suunnitteluvaiheessa oli otettu huomioon työryhmään kuuluvien henkilöiden hallitsemat ohjelmointikielet ja kokemukset eri tavoista toteuttaa vastaavia projekteja. Tämä vaikutti paljon sovelluskehyksen valintaan.

Kun sovelluskehyksiä tutkittiin, oli selvää, että selaimessa toimiva peli olisi helpointa toteuttaa käyttäen JavaScriptiä, koska siitä oli työryhmässä eniten osaamista. Muita mahdollisuuksia olisivat olleet muun muassa Java tai Flash, mutta koska ohjelmointi JavaScriptillä oli työryhmän jäsenille tuttua, projekti pyrittiin toteuttamaan sitä ohjelmointikieltä käyttäen. Pelin toteutus olisi ollut mahdollista myös ilman valmista sovelluskehystä, mutta tämä vaihtoehto olisi vaatinut oman pelimoottorin kirjoittamisen ja lisännyt työn määrää projektissa merkittävästi.

Vertailtavat sovelluskehykset olivat EaselJS, melonJS, Phaser ja Unity (taulukko 1). Unity poikkesi muista vertailussa olleista sovelluskehyksistä merkittävästi, sillä kaikki muut sovelluskehykset olivat JavaScript-kirjastoja. Unityn kaltaisia pelinkehitykseen tarkoitettuja sovelluksia, jotka soveltuvat myös selainpelien tekemiseen, ei juuri ole olemassa Unityn lisäksi. [2; 3; 4; 5.]

Taulukko 1. Sovelluskehysten vertailu [2; 3; 4; 5].

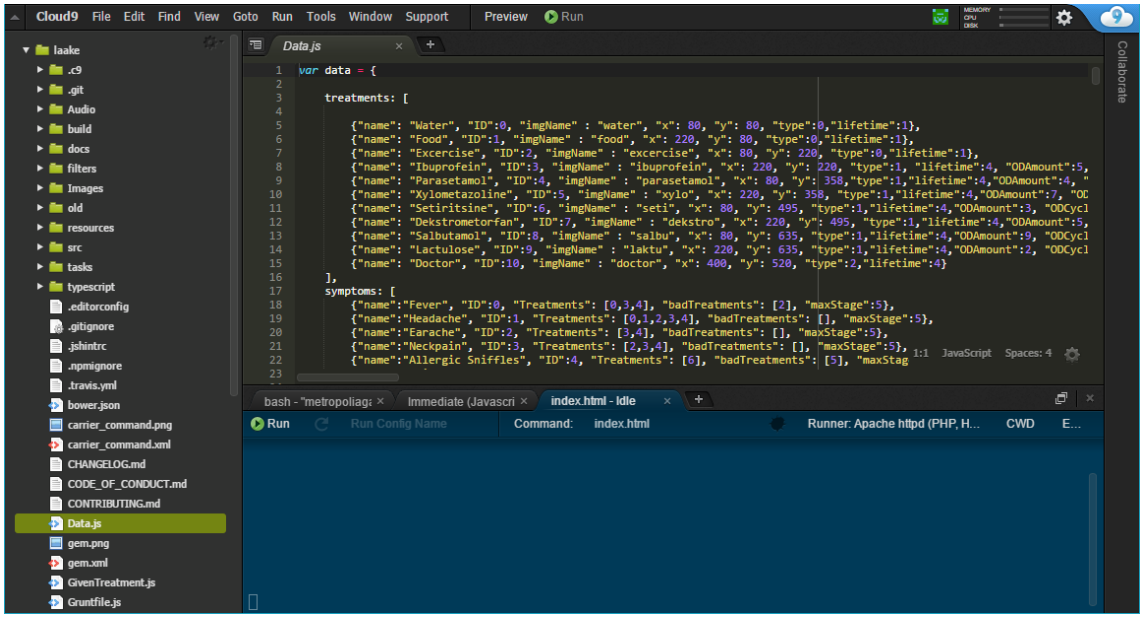
Sovelluskehys	**+**	**-**
EaselJS, JavaScript-kirjasto	• ammattimainen • hyvät esimerkit • hyvä dokumentaatio	• olisi vaatinut lisäksi TweenJS-, SoundJS- ja PreloadJS-kirjastot • tarkoitettu enimmäkseen datan graafiseen esittämiseen
melonJS, JavaScript-kirjasto	• kevyt • yksinkertainen	• suunniteltu tiilipohjaisten pelien luontiin • huonot Esimerkit • suppea dokumentointi
Phaser, JavaScript-kirjasto	• hyvät esimerkit • hyvä dokumentointi • toiminnallisuudet	• suuri määrä koodia hidastaa suorituskykyä
Unity, Pelinkehitys-sovellus	• ammattimainen • tuttu työkalu	• paljon toimintoja, joille ei käyttöä • suuri pelin koko web-ympäristöön

EaselJS mahdollistaa hienojen graafisten toteutusten esittämisen, mutta pelityökaluna sen käyttö vaikutti liian työläältä ja tähän projektiin liian monimutkaiselta, joten se jätettiin ensimmäiseksi pois vertailusta. melonJS vaikutti hyvältä vaihtoehdolta, mutta Phaseriin verrattuna sen dokumentaatio ja esimerkit ovat huonoja. Unity olisi toiminut paremmin, mikäli peli olisi haluttu toteuttaa 3D-grafiikalla, sillä myös Unity olisi tukenut JavaScriptin käyttöä työryhmän toiveiden mukaisesti. Peli kuitenkin suunniteltiin alusta alkaen käyttämään 2D-grafiikkaa asiakkaan toivetta noudattaen. Vaikka Unity mahdollistaa myös 2D-pelien luomisen, oli Phaserin valinnassa selkeä etu, kun arvioitiin pelin lopullista pakattua kokoa. Pakattu Phaser-kirjasto oli kooltaan vain alle yhden megatavun, kun taas pakattu Unity-projekti olisi kasvattanut pelin kokoa Phaseriin verrattuna useiden megatavujen verran. Tämä seikka huomioon ottaen peliä päädyttiin tekemään Phaserilla, jolla pelin latausajat pystyttiin pitämään lyhyempinä, kun jo suunnitteluvaiheessa tiedettiin, että eri tekstuureja ja ääniä tulisi pelissä olemaan useita. [2; 3; 4; 5.]

Koska projektiin valittu sovelluskehys oli tehty JavaScriptillä, harkittiin kehitysympäristöksi jotakin tietolähdettä, jossa lähdekoodi tulisi sijaitsemaan. Vaihtoehtoina olivat Git-pohjaiset tietolähdepalvelut Github ja Bitbucket, joita käyttäessä lähdekoodi pysyy koko ajan palvelussa ja ohjelmoijat vain kopioivat sen omalle koneelleen muokkausten ajaksi. Muokkausten jälkeen muutetut tiedostot lähetetään takaisin tietolähteeseen, jolloin eri käyttäjien tekemät muokkaukset on helppo yhdistää. Harkinnan jälkeen päädyttiin kuitenkin käyttämään Cloud9-nimistä verkkopalvelua, jossa erillistä tietolähdettä ei tarvita. Cloud9 valintana helpotti projektin toteuttamista, sillä se ei vaatinut konfigurointia ja sen käyttö oli hyvin yksinkertaista, mikä sopi projektin luonteeseen. [9; 10.]

3.2 Cloud9-verkkopalvelu

Cloud9 on verkkopalvelu, joka toimii kuin kehitysympäristö (kuva 2). Se mahdollistaa usean käyttäjän samanaikaisen tiedostojen muokkauksen palvelussa. Kaikki tiedostot voidaan säilyttää kehitysympäristössä, jossa kirjoitettua koodia voi suorittaa Apache-palvelua käyttävällä web-palvelimella. Palvelu näyttää reaaliaikaisesti eri käyttäjien osoittimet ja heidän kirjoittamansa osiot koodista. Tällä tavalla koodin kirjoittaminen on sujuvaa, eikä erillistä tietolähdettä välttämättä tarvita. Tietolähteen tarvetta pienentää myös se, että palvelu pitää itse yllä historiaa tiedostojen muokkauksesta, joten tarvittaessa vanhoja tiedostoversioita voidaan selata ja tiedostoja voidaan palauttaa aiempiin versioihin. Palvelussa kuitenkin on myös mahdollisuus linkittää eri tietolähteitä, kuten esimerkiksi Gitiä käyttävät palvelut. [8.]



Kuva 2. Cloud9-verkkopalvelun käyttöliittymä [8].

3.3 Phaser-sovelluskehys

Phaser on JavaScriptillä kirjoitettu valmis pelimoottori, joka on tarkoitettu 2D-pelien kehitykseen. Se on rakennettu käyttäen pohjana muokattua pixi.js-sovelluskehystä, joka taas on luotu canvas-renderöintiin käyttäen WebGL- tai 2D-kontekstia. WebGL- ja 2D-kontekstit ovat renderöintimoottoreita (rendering engine), jotka mahdollistavat joko 2D- tai 3D-grafiikan piirtämisen canvasille eli HTML:ssä sijaitsevalle elementille, ja ne löytyvät lähes kaikista moderneista selaimista. [7; 12; 13.]

Valitun pelimoottorin vahvuutena olivat ominaisuudet, jotka mahdollistavat erityyppisten 2D-pelien tekemisen, esimerkiksi point&click- ja sidescroller-pelien. Point&click tarkoittaa pelejä, joissa pääosa toiminnoista suoritetaan hiirellä painamalla, ja sidescroller puolestaan on sivustapäin kuvattu peli, jossa pelihahmo usein liikkuu pelikentässä horisontaalisesti. Phaserin erilaisista käyttösovelluksista on esimerkkejä useissa erilaisissa selainpeleissä, muun muassa erään veikkaus.fi-sivuston verkkoympäristöön tehdyssä arpapelissä, joka on toteutettu tätä sovelluskehystä käyttäen. [5.]

Alustus

Phaserin alustus (Initialization) tehdään onload-funktiossa, jota kutsutaan, kun HTML-sivu on ladattu kokonaan. Tässä vaiheessa luodaan uusi peliobjekti, jolle annetaan parametriksi pelin dimensiot ja pelin renderöintimoottori. Phaser luo HTML- sivulle canvas-elementin ja alustaa sovelluskehyksen käytettäväksi peliobjektiin. Koodiesimerkissä 1 luodaan peliobjekti ja lisätään peliobjektiin kaikki pelissä tarvittavat pelitilat, joista jokaisella on omat määrityksensä. Tämän jälkeen aloitetaan ensimmäinen pelitila “Boot”.

```
window.onload = function() {
            //initialize game
            game = new Phaser.Game(1280, 720, Phaser.AUTO, '');

            game.state.add('Boot', state.Boot);
            game.state.add('Scenarios', state.Scenarios);
            game.state.add('Game', state.Game);
            game.state.add('End', state.End);

            game.state.start('Boot');
};
```

Koodiesimerkki 1. Phaserin alustus.

Pelitilat

Peliobjektille voidaan määrittää pelitiloja (game states), jotka sisältävät pelin kaikki erilaiset tasot, kuten muun muassa latausikkunan, kentät ja valikot. Jokainen luotu pelitila ottaa parametrikseen pelitilan nimen ja määrityksen pelitilan sisällölle. Pelitilan sisällöt määritellään “state”-nimiseen objektiin (koodiesimerkki 2). Objektiin voidaan määrittää kaikki mahdolliset funktiot, joita tilassa tarvitaan, kuten esimerkiksi nappien painallukset.

```
state.Boot.prototype = {
	preload: function(){
		...
	},
	create: function() {
		...
	}
};
```

Koodiesimerkki 2. Pelitilan luonti.

Pelitilojen välillä voidaan siirtyä saumattomasti kutsumalla koodiesimerkki 3 mukaista funktiota, jossa kerrotaan siirtymään johonkin peliin määritellyistä pelitiloista. Phaser pitää huolen, että pelitilat, joista poistutaan, sulkeutuvat oikein eivätkä jää taustalle viemään muistia.

```
game.state.start('Scenarios');
```

Koodiesimerkki 3. Pelitilan aloitus.

Ominaisuudet

Phaserissa on hyvät työkalut materiaalien, kuten kuvien, äänien, fonttien ja fysiikan mallinnukseen sekä tween-siirtymien käyttämiseen sovelluskehyksessä. Myös objektien animointi onnistuu helposti käyttäen Phaserin omaa animointiin tarkoitettua funktiota. Pelin animaatio voidaan sijoittaa spritesheettiin eli kuvatiedostoon, jotka sisältävät ruudukon, jossa eri tekstuurit sijaitsevat. Yksi spritesheet voi sisältää animaation eri ruudut tai ryhmän vaihtuvia tekstuureja. Spritesheet luodaan yksinkertaisella kutsulla (koodiesimerkki 4).

```
game.load.spritesheet('myspritesheet',
'Images/spritesheets/myspritesheet.png',250,250,4);
```

Koodiesimerkki 4. Spritesheetin luonti.

Koodissa 4 luodaan spritesheet, jonka yhden ruudun koko on 250px * 250px. Funktiolle annetaan parametreina seuraavat muuttujat:

- uniikki nimi, jolla spritesheetiä voidaan kutsua
- polku, joka määrittää kuvatiedoston sijainnin
- vaaka- ja pystypikselit, jotka määrittävät yhden ruudun koon spritesheetissä
- kokonaisluku, joka määrittää ruutujen määrän yhdellä rivillä; tällöin Phaser tietää, kuinka moneen osaan kuva pitää jakaa (kuva 3).

Kuva 3. Esimerkki pelin potilaiden hiusten tekstuureista spritesheetissä.

Phaser tukee tavanomaisten bitmap-fonttien käyttämistä peleissä. Bitmapit ladataan Phaserin omalla funktiolla, joka ottaa parametrikseen nimen, jolla fonttia kutsutaan, kuvatiedoston ja XML-tiedoston, jossa määritellään kunkin kirjaimen alue kuvatiedostosta (koodiesimerkki 5).

```
game.load.bitmapFont('desyrel',
'assets/fonts/bitmapFonts/desyrel.png',
'assets/fonts/bitmapFonts/desyrel.xml');
```

Koodiesimerkki 5. Bitmap-fonttien lataaminen Phaserissa.

Sovelluskehys mahdollistaa myös web-fonttien käyttämisen, kunhan fontit ladataan ensin HTML:ään. Koodiesimerkissä 6 on esimerkki web-fontin “Gloria Hallelujah” lataamisesta Google Fonts -palvelusta.

```
<link href='https://fonts.googleapis.com/css?family=Gloria+Hallelujah'
rel='stylesheet' type='text/css'>
```

Koodiesimerkki 6. selainfonttien lataaminen HTML-elementtinä.

Phaserissä voidaan käyttää ääniä lataamalla äänitiedostot muuttujaan kutsulla (koodiesimerkki 7). Tuetut ääniformaatit ovat riippuvaisia selaimesta, mutta yleisesti suositellaan käyttämään mp3-, ogg- tai m4a-formaattia, jolloin useimmat moderneista selaimista pystyvät toistamaan äänet (taulukko 2).

```
game.load.audio('click', 'Audio/click.m4a');
```

Koodiesimerkki 7. Äänitiedoston lataus.

Taulukko 2. Audioformaattien tuki selaimissa [7].

Formaatti	Tuetut selaimet
mp3	IE, Edge, Firefox, Chrome, Safari, Opera, Android, iOS Safari
ogg	Firefox, Chrome, Opera, Android
m4a	Edge, Firefox, Chrome, Safari, Opera, Android, iOS Safari

Fysiikan mallinnus Phaserissa on toteutettu siten, että objektit tai objektiryhmät, joiden halutaan käyttäytyvän mallinnuksen mukaan, annetaan parametreinä funktiolle. Tämän jälkeen ne käyttäytyvät fyysisinä objekteina toisiaan kohtaan. Koodiesimerkissä 8 näytetään esimerkki, kuinka pelaajan hahmon (player) ja pelin lattian (floor) välille luodaan törmäytin (collider), joka varmistaa, etteivät objektit voi kulkea toistensa lävitse, vaan törmäävät toisiinsa, mikäli ne koskettavat toisiaan.

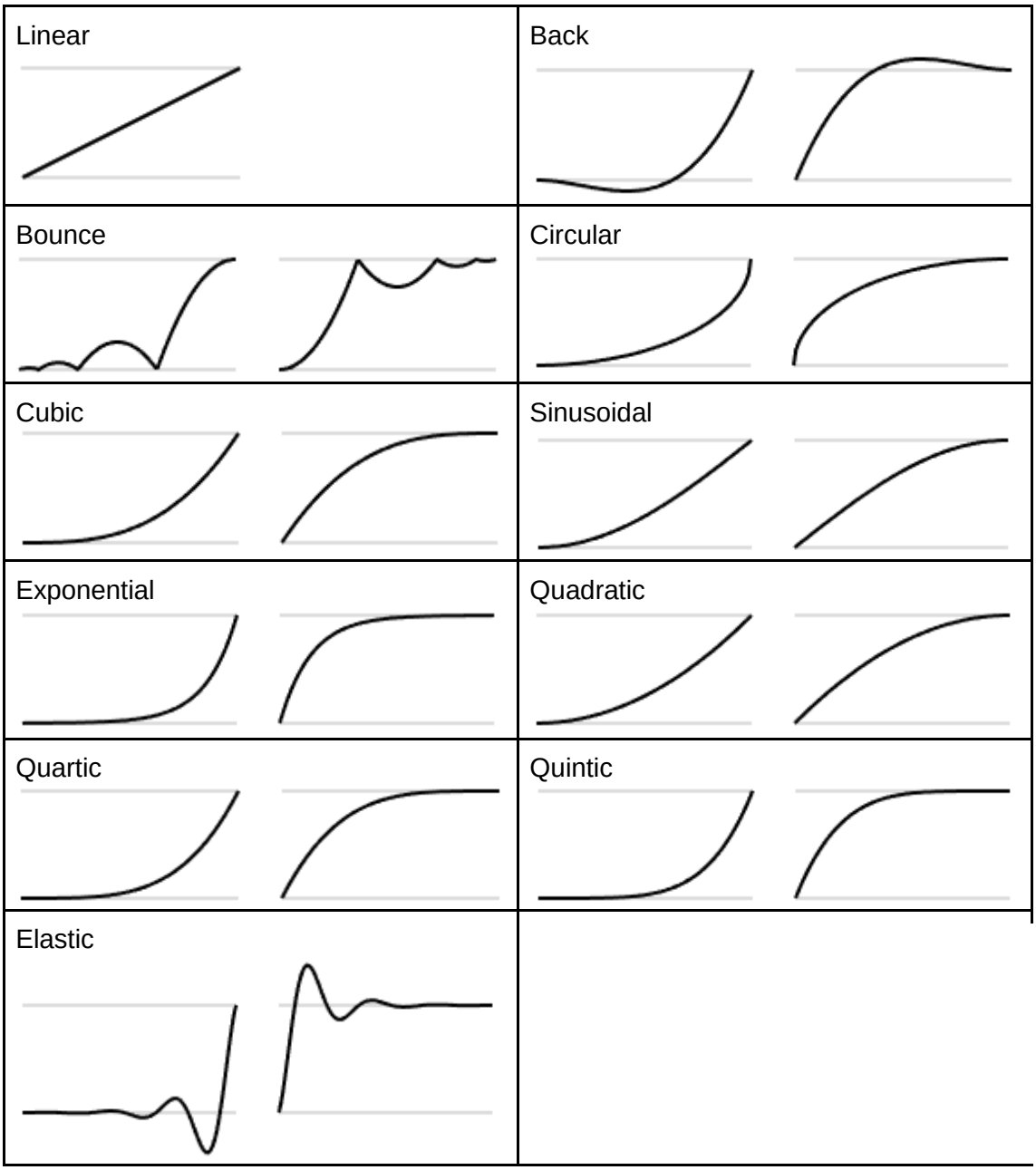
```
var collider = game.physics.arcade.collide(player, floor);
```

Koodiesimerkki 8. Fysiikan mallinnus kahden objektin välille.

Tween-siirtymä tarkoittaa toimintoa, joka mahdollistaa objektin yksittäisten tai useiden parametrien muuttamisen tietyn ajanjakson aikana. Tätä toimintoa voidaan käyttää esimerkiksi objektin X- ja Y-koordinaattien muuttamiseen. Koordinaattien arvot liikkuttavat objektia pelialueella. Siirtymälle annetaan parametriksi objekti, joka sisältää muutettavat parametrit, aika millisekunteina sekä siirtymän tyylin (koodiesimerkki 9). Siirtymän tyylillä tarkoitetaan sitä, miten animaatio toteuttaa itsensä. Lisäksi tyyleillä on vaihtoehtoiset metodit “easeIn” tai “easeOut”, joilla määrätään tyylin suunta (kuva 4).

```
tweenA = game.add.tween(spriteA).to( { x: 600 }, 2000,
"Quart.easeOut");
```

Koodiesimerkki 9. Tween-siirtymän luominen.

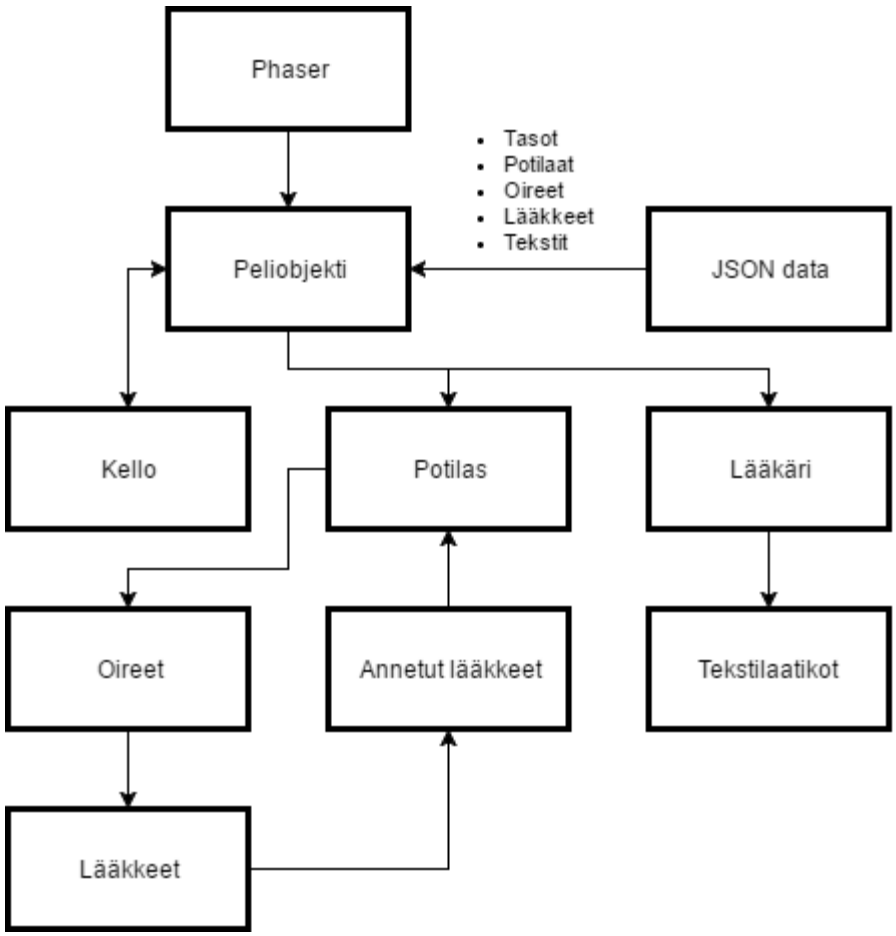


Kuva 4. Siirtymien eri tyylit, joista vasemmalla “easeIn” ja oikealla “easeOut” [11].

4 Lääkekasvatuspelin ohjelmointi

4.1 Ohjelmoinnin suunnittelu

Projektissa tehty lääkekasvatuspeli sisältää useita toimintoja, joista monet olivat oma objektinsa, jotta niitä voitiin tarvittaessa kutsua ja kontrolloida peliobjektista (kuva 5). Osa toiminnallisuuksista kirjoitettiin erillisiin tiedostoihin, minkä tarkoituksena oli pitää toiminnallisuudet omina kokonaisuuksinaan ja helpottaa niiden muokkausta. Eri objektit sisältävät omat funktionsa, joilla kutakin objektia voidaan ohjata. Lähestymistavalla pyrittiin siihen, ettei toiminnallisuuksien koodi olisi hajautettuna useaan eri paikkaan, vaan yhdestä tiedostosta käsin kunkin ominaisuuden muokkaaminen olisi nopeaa ja helppoa.



Kuva 5. Lääkekasvatuspelin toiminnot kuvattuna kaaviossa.

4.2 JSON-muuttujat

Pelin JSON-data on muuttujissa, jotka sisältävät tiedot kentistä, potilaista, taudeista, oireista, allergioista, piilevistä sairauksista sekä eri kielistä JSON-muodossa. JSON tekee datan muokkaamisesta tai lisäämisestä helppoa, sillä se koostuu matriiseista ja objekteista. Objektit sisältävät avain-arvoparin, joihin tarvittavat tiedot voidaan tallentaa. Pelin vaikeusasteiden balansointi oli projektin onnistumisen kannalta tärkeää, ja sitä helpotti huomattavasti helposti muokattavissa oleva JSON-data. Eri hoitomuotoja, oireita ja potilaita pystytään lisäämään JSON-dataan vähällä työllä (koodiesimerkki 10).

```
data = {
      treatments: [
            { "name": "Water", "ID":0, "imgName": "water", "x": 80,
            "y": 80, "type":0, "lifetime":1 }
      ],
      symptoms: [
            { "name": "Fever", "ID":0, "Treatments": [0,3,4],
            "badTreatments": [2], "maxStage":5 }
      ],
      patients: [
            { "name": "Matti", "ID":0, "x":555, "y":180 }
      ]
}
```

Koodiesimerkki 10. Pelin asetusten JSON-data.

Peli sisälsi kaiken tekstisisältönsä sekä suomen- että ruotsinkielisenä. Asiakas toimitti peliin upotettavien tekstien molemmankieliset versiot, jotka sijoitettiin JSON-dataan omiksi muuttujikseen. Koodiesimerkissä 11 on esimerkki suomenkielisestä JSON-datasta. Riippuen käyttäjän kielivalinnasta peli lataa oikean kielen muuttujan perusteella. Pelin tekstejä muutettiin useita kertoja, ja pyrkimyksenä oli saada ne toimimaan dynaamisesti eri tilanteissa. Lauseiden muokkaaminen oli helppoa JSON-datan ansiosta.

```
strings_fi = {
            "general": {
                        "person":"Henkilö"
      }
      "short_desc": {
            "timer": "Tämä kello näyttää, mikä aika vuorokaudesta on.
            Mikäli lääkehoitoa tarvitaan, muista antaa lääkkeet oikeaan
            aikaan. Pidä huoli, ettei kerralla otettava annos, eikä
            päivittäinen annos nouse liian suureksi"
      }
}
```

Koodiesimerkki 11. Suomenkielisiä tekstejä JSON-datassa.

4.3 Kello

Pelin ensimmäisessä versiossa siihen sisältyneen kellon oli tarkoitus olla reaaliaikainen sekuntikello, mutta pelin testauksen yhteydessä ominaisuus päädyttiin muuttamaan lopulliseen muotoonsa, jonka toiminta ja toteutus kuvataan tässä luvussa.

Peli koostuu pelaajalle annetuista vuoroista, joiden aikana hän pystyy toteuttamaan pelin tarjoamia toimintoja, sekä vaikutusajasta, jonka aikana iteroidaan läpi pelaajan tekemät toiminnot ja muokataan potilaiden tilaa ja annettujen lääkkeiden vaikutusta. Pelaaja siirtyy vuorostaan vaikutusjaksoon painamalla pelissä nuolipainiketta. Jokainen vaikutusaikajakso siirtää pelin kelloa kolme tuntia eteenpäin.

Koodiesimerkissä 12 on funktio, joka suoritetaan aina, kun kelloa siirretään eteenpäin. Ensin toistetaan ääni, joka ilmaisee pelaajalle ajan kulumista. Tämän jälkeen käännetään kellon viisaria 90 astetta eteenpäin. Lopuksi kutsutaan updatePatients-funktiota, joka päivittää potilaiden tilan vaikutusajan jälkeen riippuen pelaajan aikaisemmin tekemistä toiminnoista.

```
updateTime: function(){
            click.play();
            clockHand.angle+=90;
            this.updatePatients();
}
```

Koodiesimerkki 12. Kellon päivitysfunktio.

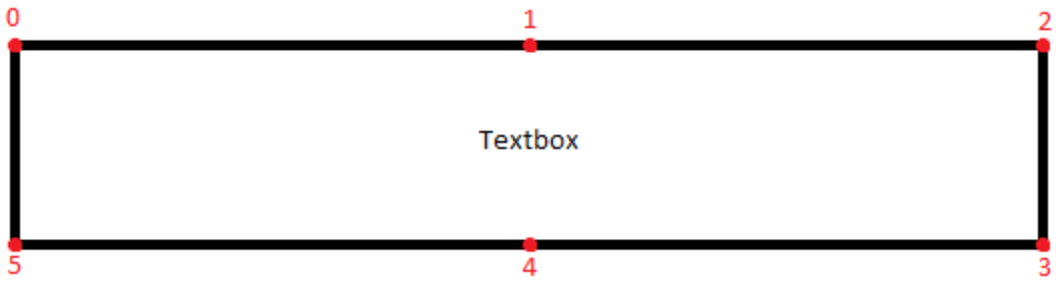
4.4 Tekstilaatikot

Pelin alkutekstit, puhekuplat ja potilasraportit luodaan käyttäen koodiesimerkin 13 mukaista overlayElement-funktiota. Funktio ottaa vastaan parametrit tekstilaatikon sijainnista, koosta, tekstistä, ankkurin kohdasta ja tyypistä.

```
function overlayElement(x,y,width,height,text,anchor,type)
```

Koodiesimerkki 13. Tekstilaatikon funktio ja sen hyväksymät parametrit.

Ankkuripisteellä ilmaistaan, mikä kohta laatikosta sijoitetaan X- ja Y-koordinaatteihin. Pisteellä on kuusi eri vaihtoehtoa alkaen vasemmasta yläkulmasta ja kiertäen myötäpäivään (kuva 6). X- ja Y-koordinaatit ovat suhteellinen luku 0:n ja 1:n välillä, jolla ilmaistaan tekstilaatikon sijaintia riippumatta pelialueen koosta.



Kuva 6. Ankkuripisteiden sijainnit.

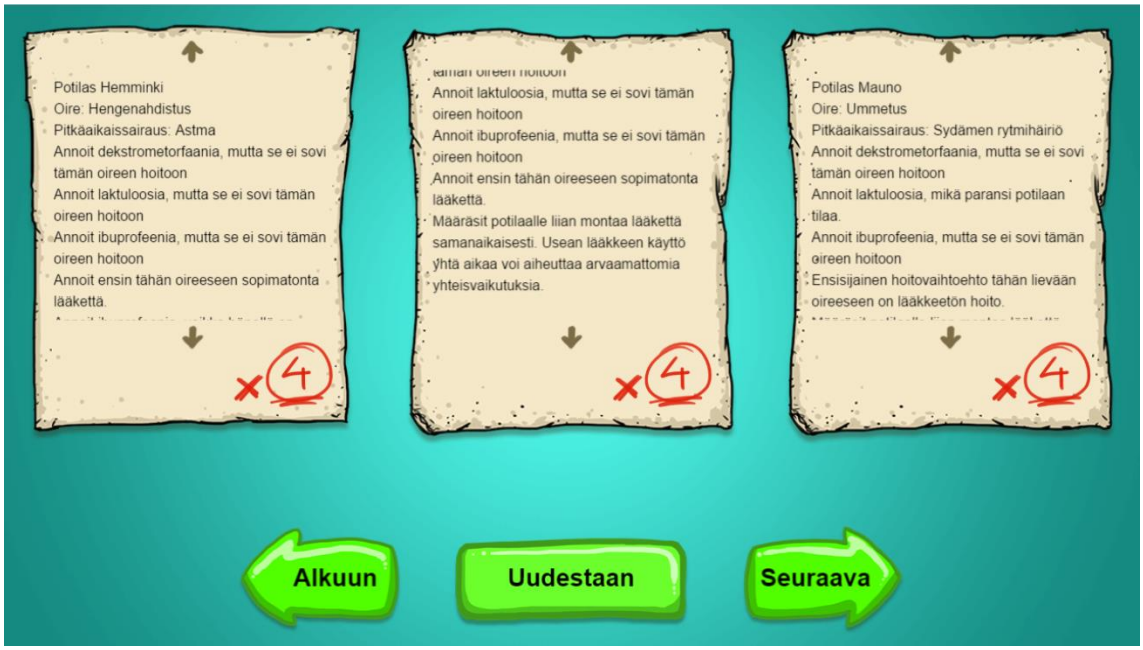
4.5 Potilaat

Potilas-objektit tallennetaan Patients-nimiseen matriisiin, josta tiettyjä potilaita voidaan etsiä annetun ID:n perusteella. Pelissä potilaita voi olla yhdestä kolmeen riippuen siitä, mitä tasoa pelissä pelataan. Objektit sisältävät tiedot potilaiden tekstuureista, terveydestä, oireista ja piilevistä sairauksista. Jokaisella potilaalla on JSON-datassa määritellyt oireet ja piilevät sairaudet, joihin vaikuttaa valittu pelitaso. Potilaiden luonnissa käytetään satunnaisia tekstuureja ja henkilöiden nimiä, jotta jokaisella pelikerralla peli tuntuisi pelaajalle erilaiselta. Koodiesimerkissä 14 nähdään, miten potilaan nimi valitaan satunnaisesti nimi-matriisista.

```
var nameIdx = Math.floor(Math.random() * (nameArray.length));
   this.name = nameArray[nameIdx].toString();
   nameArray.splice(nameIdx,1);
```

Koodiesimerkki 14. Nimien valitseminen.

Peli kerää talteen pelaajan kullekin potilaalle tekemät toiminnot ja kirjoittaa ne matriisiin, joka sijoitetaan potilas-objektiin. Tason loputtua kunkin potilaan matriisit iteroidaan läpi ja näytetään omina raportteinaan niin sanotussa pelin loppuruudussa. Tekstit koostuvat lauseen osista, jotta niistä voidaan dynaamisesti muodostaa järkeviä lauseita käytettyjen lääkkeiden ja niiden vaikutusten perusteella (kuva 7).



Kuva 7. Loppuruutu, jossa näytetään pelin jokaisen potilaan koostetut raportit.

Koodiesimerkissä 15 nähdään pieni osa raportoinnin koodia. Mikäli potilaan tila on tarpeeksi hyvä ja hoitomuotona on lääkkeetön hoito, siitä tulostetaan palaute riippuen siitä, oliko lääkkeetön hoitomuoto väärä vai oikea.

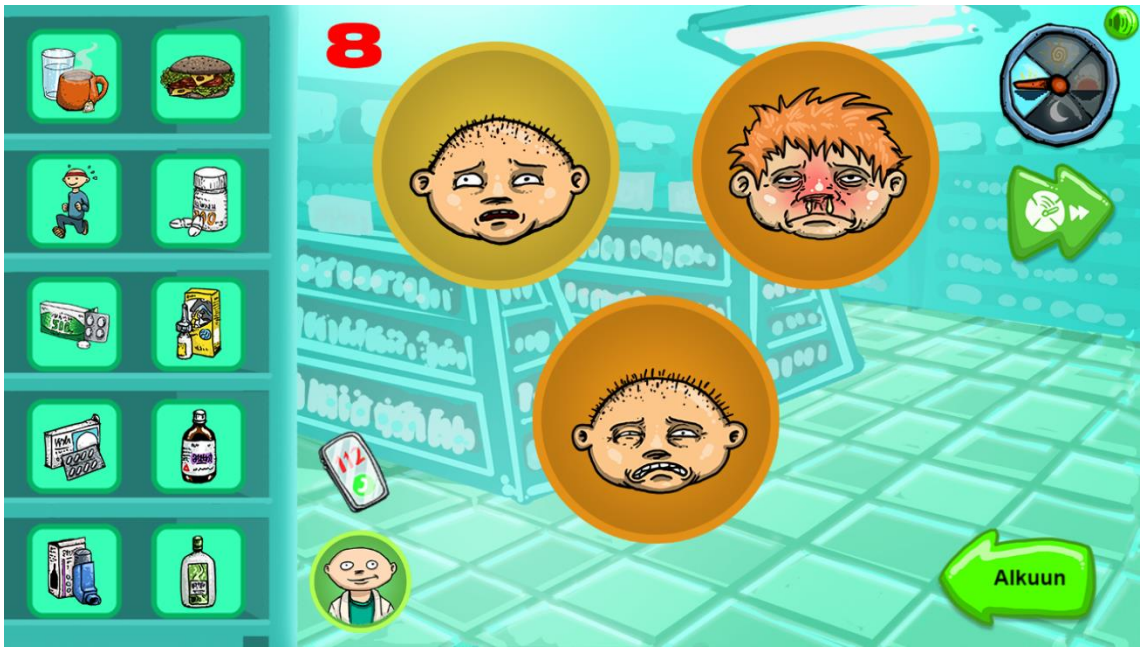
```
if(this.symptom.stage <= 3) {
            if(correctTreatment&&treatment.type == 0 ) {
                        this.score += 1;
            } else if(treatment.type == 0 && !correctTreatment){

            this.log.push(strings['feedback']['wrongNonMed']);
            } else {
                        this.log.push(strings['feedback']['nonMed']);
            }
}
```

Koodiesimerkki 15. Hoidon vaikutus potilaaseen.

4.6 Oireet

Oireille on määritelty pelin koodissa eri tasoja riippuen siitä, kuinka huonoon tilaan potilas on päässyt. Jokainen oire-objekti sisältää tiedon siitä, millä tasolla oire on. Oireen taso vaikuttaa siihen, kuinka peli reagoi potilaalle tehtyihin toimenpiteisiin ja miten se näytetään visuaalisesti potilaan spritesheetissä (kuva 8). Taso määrittelee pelin tarjoamien hoitomuotojen tehokkuuden. Esimerkiksi vaadittava lääkkeiden määrä tai tarve mennä lääkäriin määräytyvät oireen tason mukaan. JSON-datassa määritellään, mitkä lääkkeet antavat positiivisia tai negatiivisia vaikutuksia oireiden hoidossa (koodiesimerkki 10).



Kuva 8. Potilaiden oireet näytetään visuaalisesti erinäköisillä spritesheeteillä. Kuvassa havainnollistuvat myös lääkevaihtoehdot ja kello.

4.7 Lääkkeet

Lääke-objekteihin tallentuvat JSON-datasta tiedot pelissä käytettävissä olevista lääkkeistä. Jokainen lääke luo oman objektinsa, josta tarkistetaan kunkin lääkkeen tiedot, kun pelaaja valitsee käyttävänsä lääkettä pelissä. Lääke-objekti sisältää lääkkeen tekstuurit ja tiedon siitä, kuinka monta kappaletta kyseistä lääkettä voi ottaa samanaikaisesti, ennen kuin potilas saa yliannostuksen.

Jokainen potilaalle annettu uniikki lääke luo oman givenTreatment-objektin, joka tallennetaan Patient-objektiin. Tällä tavalla pidetään kirjaa pelissä potilaalle annettujen lääkkeiden määrästä ja tyypistä sekä vaikutuksen kestoajoista. Lääkkeelle luodussa givenTreatment-objektissa seurataan, onko kyseistä lääkettä annettu pelissä liikaa. Jokaisella pelin kellon siirrolla peliobjektissa iteroidaan potilaille annetut lääkkeet läpi ja vähennetään niiden vaikutusaikaa yhdellä vuorolla. Kun vaikutusaika on loppu, givenTreatment-objekti poistetaan Patients-objektista.

4.8 Lääkäri

Lääkäri on peliä helpottamaan ja sen opettelua nopeuttamaan suunniteltu toiminto, joka antaa pelaajalle neuvoja eri lääkkeiden käytöstä ja kertoo tarkemmin pelin muista pelaajalle tarjotuista toiminnoista (kuva 9). Lääkäri-objekti sisältää funktiot pelin puhekuplien näyttämiselle. Kun lääkäri-painiketta klikataan pelissä, se asetetaan pelialueen oikean reunan ulkopuolelle ja siirretään näkyviin käyttäen Tween-toimintoa (koodiesimerkki 16). Kun animaatio on valmis, luodaan peliin puhekupla, minkä jälkeen se näytetään pelissä halutulla tekstillä.

Kuva 9. Lääkärin puhekuplassa neuvotaan pelaajaa. Puhekupla on luotu overlayElement-funktiolla ja siinä näkyvät tekstit on haettu JSON-datasta.

```
this.animIn.to({ x: game.world.width-175 }, 0,
Phaser.Easing.Circular.Out);
this.animIn.onComplete.add(inAnim, this);
this.animIn.start();


function inAnim () {
      bubble = new
      overlayElement(0.57,0.26,640,190,strings['general']['help'], 1,
      2);
}
```

Koodiesimerkki 16. Lääkärin animointi esille.

4.9 Testaus

Pelin ensimmäinen testausvaihe toteutettiin työryhmässä. Työryhmässä suoritetulla testauksella pyrittiin löytämään pelistä merkittävimmät ongelmat ja virheet sekä varmistamaan sen perusominaisuuksien toimivuus.

Kun pelin runko oli rakennettu ja toimiva, annettiin peli asiakkaan testattavaksi projektin tekemisen ohella niin, että asiakas toteutti testejä aina viikonloppuisin. Jokaisena maanantaina testeissä ilmenneet viat ja muutokset käytiin läpi ja kirjattiin työlistalle. Jatkuva viikoittainen testaus oli hyvä tapa pitää korjausten määrä mahdollisimman pienenä, sillä ilmenneisiin vikoihin puututtiin heti ja ne pystyttiin korjaamaan viiveettä. Kun asiakkaan mielestä peli alkoi olla lähellä julkaisukelpoista, järjestettiin kaksi suurempaa tes-

tauspäivää, joiden aikana peliä esiteltiin kohderyhmän ikäisille koululaisille ja heidän annettiin kokeilla peliä. Pelin kokeilemisen jälkeen heiltä kysyttiin pelin hyviä ja huonoja puolia. Palautteen perusteella saatiin tehtyä peliin hyödyllisiä muutoksia. Kohderyhmän kokemuksien ansiosta esimerkiksi pelin kello muutettiin ensimmäisen version reaaliaikaisesta sekunttikellosta lopulliseen versioon päätyneeksi vuoropohjaiseksi kelloksi, joka siirtyi eteenpäin aina kolmen tunnin jaksoissa. Myös tekstejä muokattiin testipäivien jälkeen kohderyhmälle sopivammaksi, jotta he ymmärtäisivät annetut opastukset paremmin.

5 Yhteenveto

Insinöörityön tavoitteena oli suunnitella ja toteuttaa lääkekasvatuspeli lääkealan turvallisuus- ja kehittämiskeskus Fimealle Metropolian Game Studiosin työryhmässä. Asiakkaan tavoitteena oli pelin avulla lisätä nuorten lääkekasvatusta mielenkiintoisella ja nuorten houkuttavaksi kokemalla tavalla. Projektin haasteita olivat rajattu työaika sekä pyrkimys pitää projektissa käytetyt työkalut ja sen toteutus mahdollisimman yksinkertaisena.

Pelin tekemiseen valittu sovelluskehys Phaser toimi projektissa kuten oli suunniteltu, ja sillä pystyttiin toteuttamaan kaikki pelin suunnitteluvaihessa pelille esitetyt vaatimukset. Phaserin käyttö projektissa osoittautui myös hyväksi valinnaksi sen sijaan, että pelimoottori olisi rakennettu itse, koska sen avulla säästettiin paljon aikaa. Tämän kokemuksen perusteella Phaser sopii mainiosti useimpien selaimessa toimivien pienten 2D-pelien tekemiseen.

Koska pelin ohjelmoinnissa yritettiin keskittyä pääosin toimiviin ja helppoihin ratkaisuihin, uudelleen käytettävän koodin kirjoittamiseen ei aina käytetty aikaa, eikä sille myöskään projektin suunnitteluvaiheessa vaikuttanut olevan sen kertaluontoisuuden takia todellisia perusteita. Vaikka koodaustavan valinta oli tietoinen, se johti siihen, että kehitysvaiheessa pelin ominaisuuksia muokattaessa jouduttiin kirjoittamaan uudestaan esimerkiksi useita rivejä if-else-lausekkeita. Koodin jäykkyys ja hankala muokattavuus myös aiheutti sen, että projektin loppuvaiheilla pelistä löytyi paljon ohjelmointivirheitä, jotka olivat aiheutuneet ominaisuuksien muokkauksesta. Näistä syistä voi jälkikäteen todeta, että uudelleen käytettävän koodin kirjoittaminen alusta saakka olisi voinut olla parempi ratkaisu paitsi tyylillisesti, myös käytännöllisyyden kannalta.

Fimea oli tyytyväinen projektin lopputulokseen. Lääkekasvatuspeli on edelleen esillä Fimean lääkekasvatukseen tarkoitetulla sivulla, ja sitä myös käytetään kouluissa osana 3.–6.-luokkalaisten lääkekasvatusta. Projekti siis ylsi hyvin sille asetettuihin tavoitteisiin, ja peli palvelee tarkoitettua käyttötarkoitusta.

Lähteet

1 Lasten lääkekasvatus. 2013. Verkkodokumentti. Fimea. <https://www.fimea.fi/documents/160140/753095/22752_KAI_1_2013_verkkoon.PDF/70664662-e863-4514-b293-8b4f35e6df92> Luettu 12.11.2016.

2 Unity3D. Verkkodokumentti. Unity Technologies. <https://unity3d.com/> Luettu 25.9.2016.

3 Phaser framework. Verkkodokumentti. Photon Storm Ltd. <https://phaser.io/> Luettu 25.9.2016.

4 melonJS. Verkkodokumentti. melonJS team. <http://melonjs.org/> Luettu 19.4.2017.

5 EaselJS. Verkkodokumentti. createJS. <http://www.createjs.com/easeljs> Luettu 19.4.2017.

6 Davey, Richard. 2016. Take on the role of a monster hunter, and free the imprisoned fairies in this Finnish lottery game. Verkkodokumentti. Photon Storm Ltd. <https://phaser.io/news/2016/09/veikkaus-hirviot> Luettu 15.10.2016.

7 Deveria, Alexis & Schoors, Lennart. Can I use. Verkkodokumentti. <http://caniuse.com/> Luettu 19.4.2017.

8 Cloud9 - Your development environment, in the cloud. Verkkodokumentti. Cloud9 IDE Inc. <https://c9.io/> Luettu 29.10.2016.

9 BitBucket - The Git solution for professional teams. Verkkodokumentti. BitBucket. <https://bitbucket.org/> Luettu 19.4.2017.

10 GitHub - The world's leading software development platform. Verkkodokumentti. GitHub. <https://github.com/> Luettu 19.4.2017.

11 Easing functions. Verkkodokumentti. Sitnik, Andrey. <http://easings.net/> Luettu 19.4.2017.

12 Hawkes, Rob. 2011. Foundation HTML5 Canvas. Paul Manning. Luettu 20.4.2017.

13 Parisi, Tony. 2012. WebGL: Up and Running. O'Reilly Media, Inc. Luettu 20.4.2017.