

Kami Nasri

Mobiilipelin kehitys Libgdx-sovelluskehyksellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

18.4.2016

Tekijä Otsikko Sivumäärä Aika	Kami Nasri Mobiilipelin kehitys Libgdx-sovelluskehityksellä 36 sivua + 2 liitettä 18.04.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Fil.maist. Teemu Saukonoja
<p>Insinööriyön tavoitteena oli kehittää mobiilipelille prototyyppi, jolla pystyisi testaamaan pelimekaniikan toimivuutta ja katsomaan, mitä lisäyksiä peli tarvitsisi, jotta peli olisi mahdollisimman koukuttava.</p> <p>Insinööriyöksi tehtiin Redpill-niminen peli, jossa tavoitteena on rikkoa punaisia pillereitä, ja estää niitä etenemästä keskellä olevaan neliöön. Peli päättyy keskellä olevan neliön kasvaessa peliruudun yli. Peliin tehtiin myös päävalikko, josta pelaaja pääsee aloittamaan uuden pelin.</p> <p>Peli ohjelmoitiin Java-ohjelmointikielellä käyttäen Libgdx-sovelluskehystä, joka mahdollistaa pelin kääntämisen monille eri alustoille. Tämä mahdollisti pelin kääntämisen työpöydälle testaamista varten sen sijaan, että peli pitäisi kääntää joka kerta puhelimelle.</p> <p>Insinööriyöprojektista syntyi suunnitellun pelin prototyyppi, jolla pystyttiin testaamaan pelimekaniikan toimivuutta. Peliä testattaessa huomattiin, että peli on liian helppo ja yksinkertainen, jonka myötä lisäsimme pelin vaikeusastetta.</p> <p>Johtopäätöksinä huomattiin, että Libgdx-sovelluskehityksellä prototyypin kehitys osoittautui työlääksi muihin vaihtoehtoihin verrattuna kuten Unity-pelimooottoriin. Jatkosuunnitelmana on kehittää peli uudestaan Unity-pelimooottoria käyttäen.</p> <p>Redpill-peli ei julkaistu markkinoille prototyypin valmistuttua. Tavoitteena oli julkaista peli Google Play -palveluun ja App Storeen pelin valmistuttua.</p>	
Avainsanat	LibGDX, pelikehitys, mobiili, mobiilipeli, Android

Author Title Number of Pages Date	Kami Nasri Mobile game development using Libgdx framework 36 pages + 2 appendices 18.04.2016
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software engineering
Instructor	Teemu Saukonoja, M.Sc.
<p>The goal of the study was to develop a prototype mobile game using (the) Libgdx framework. The aim was to become familiar with the framework and compare it to other frameworks.</p> <p>The study seeks to explain the pros and cons of using the Libgdx framework and to show what the different processes and steps are to make a cross-platform mobile game.</p> <p>The mobile game is a puzzle like game in which the aim is to prevent specific objects from hitting the center of the screen by destroying them and to survive as long as possible. When a red object hits the center object, it grows little by little, and when the center object crosses the screen, the game ends.</p> <p>The project was carried out with the Libgdx framework which is a cross-platform framework. This made it possible to debug the game in desktop environment rather than compiling it to mobile every time. The game was made using the Java programming language.</p> <p>The end result was a prototype of the designed game which allowed testing the game mechanics. While testing it was found out that the game was too easy and a decision was made to raise/increase the difficulty.</p> <p>In conclusion, the Libgdx framework requires high programming knowledge and it takes much more time and effort to create a game with it considering other much easier options like Unity.</p> <p>The game was not published after the prototype was made. The aim is to publish the game on Google Play and App Store when it is ready.</p>	
Keywords	Libgdx, mobile game, game development, 2D game

Sisällys

Lyhenteet

1	Johdanto	1
2	Libgdx-sovelluskehys	2
2.1	Libgdx-sovelluskehysten historia	2
2.2	Libgdx-sovelluskehysten elämäntaari	3
2.3	Projektissa käytettyjä kirjastoja	5
2.4	Hyödyt ja heikkoudet	6
2.5	Muita sovelluskehysiä ja pelimoottoreja	7
3	Pelin suunnittelu	10
3.1	Peli-idea ja sen suunnittelu	11
3.2	Pelin toiminta	12
3.3	Pelin säännöt ja pelimekaniikka	14
4	Pelin toteutus	16
4.1	Projektissa käytetyt työkalut	16
4.2	Libgdx-projektin luominen	17
4.3	AndroidLauncher- ja DesktopLauncher-luokkien toteutus	19
4.4	AssetManager-luokan toteutus	20
4.5	Pelinäkymän toteutus	21
4.6	Pelaajan viivan luominen	25
5	Johtopäätökset ja jatkosuunnitelmat	28
5.1	Johtopäätökset	28
5.2	Jatkosuunnitelmat	30
6	Yhteenveto	31
	Lähteet	34
	Liite 1. GameScreen.java	
	Liite 2. PillManager.java	

Lyhenteet

AFX	Android effects. Mario Zechnerin oma Android-alustoille tehty projekti, josta kehittyi Libgdx-sovelluskehys.
I/O	Input/output. Siirräntä eli tiedonsiirto laitteiston komponenttien välillä.
OpenGL	Open Graphics Library. Rajapinta tietokonegrafiikan tuottamiseen. OpenGL on laitteis- toriippumaton rajapinta.
OpenAL	Open Audio Library. Rajapinta, joka on tarkoitettu ääniohjelmointiin.
FreeType	Ohjelmontikirjasto, joka mallintaa tekstin bitmap-muotoon ja mahdollistaa fonttiin liittyviä operaatioita.
Mpg123	Audiosoitin, joka tukee MPEG-, MP2- ja MP3-formaatteja.
Xiph.org	Organisaatio, joka tarjoaa vapaita multimediatiedostoformaatteja ja jonka pääpaino on Ogg-perheen formaateissa.
Soundtouch	Audion prosessointikirjasto, jolla voidaan säätää esimerkiksi äänen tempo.
Box2D	Kaksiulotteisille peleille tarkoitettu fysiikkamoottori.
LWJGL	Lightweight Java Game Library. Alustariippumaton kirjasto, jota käytetään useimmiten pelien kehitykseen.
KissFFT	Kiss Fast Fourier Transform. Kevyt kirjasto, joka tarjoaa algoritmin nopeaan Fourier'n muunnokseen.
IDE	Integrated Development Environment. Ohjelmointiympäristö, jolla voidaan suunnitella ja toteuttaa ohjelmistoja. Ohjelmistoympäristö tarjoaa tekstieditorin ja ohjelmointikielen kääntäjän.
SDK	Software Development Kit, joka tarjoaa eri työkaluja ohjelmiston kehitykseen.

1 Johdanto

Insinööriyön tavoitteena on suunnitella ja kehittää Libgdx-sovelluskehyksellä mobiilipelin prototyyppi, jota voi jälkeenpäin jatkokehittää ja pelin valmistuttua julkaista Google Play -palvelussa. Raportissa perehdytään ensimmäisenä Libgdx-sovelluskehityksen historiaan, ominaisuuksiin, hyötyihin ja haittoihin ja esitellään, mitä muita sovelluskehityksiä on tarjolla, jotka myös tarjoavat monialustaista pelinkehitystä. Työssä esitellään ja käydään läpi mobiilipelin kehitys Libgdx-sovelluskehyksellä.

Insinööriyön aihe valittiin kirjoittajan omasta kiinnostuksesta ja halusta saada mobiilipelistä prototyyppi valmiiksi jatkokehittäväksi. Myös projektista oppiminen ja Libgdx-sovelluskehikseen tutustuminen olivat suurena motiivina insinööriyön aiheen valinnassa.

Libgdx-sovelluskehys käyttää avukseen monia muita kolmannen osapuolen kirjastoja ja tarjoaa niiden ominaisuuksia kehittäjille. Kolmannen osapuolen kirjastot, joita Libgdx-sovelluskehys käyttää, ovat OpenGL, FreeType, box2d, LWJGL, OpenAL, KissFFT, mpg123, xiph.org ja soundtouch.

Itse mobiilipeli, joka insinööriyönä tehdään, kuuluu Arcade-lajityyppiin, jossa päätarkoituksena on estää punaisia pillereitä osumasta ruudun keskellä olevaan neliöön tuhoamalla punaiset pillerit. Keskellä oleva neliö laajenee aina punaisen pillerin osuttua neliöön. Peli sisältää myös keltaisia pillereitä, jotka kutistavat keskellä olevaa neliötä osuessaan neliöön. Jos pelaaja tuhoaa keltaisen pillerin, sektio menee inaktiiviseksi muuttamaksi sekunniksi, jolloin sektorilla punaiset pillerit pääsevät liikkumaan neliötä kohti ilman, että pelaaja pystyy tuhoamaan niitä.

Tulevaisuuden suunnitelmiin kuuluu Google Play Developer -ohjelmointirajapinnan lisääminen projektiin ja pelin julkaiseminen Google Play -palveluun ja myöhemmin mahdollisesti pelin julkaisu myös App Store -sovelluskauppaan.

2 Libgdx-sovelluskehys

Libgdx-sovelluskehys on avoimen lähdekoodin sovelluskehys, joka tarjoaa pelinkehittäjälle eri kirjastoja, joiden avulla pelikehitys helpottuu. Libgdx-sovelluskehys tarjoaa yhteisen API:n eli ohjelmointirajapinnan, joka toimii kaikilla tuetuilla alustoilla. Tuetut alustat ovat Windows, Linux, Mac OS X, Android, iOS ja HTML5. [1.]

Libgdx-sovelluskehys mahdollistaa pelinkehityksen monille eri alustoille ilman, että jokaiselle alustalle joutuu erikseen ohjelmoimaan ja kääntämään peliä. Tämä myös helpottaa pelin testaamista erityisesti, jos kohdealustoina ovat Android ja iOS, sillä kehittäjän ei tarvitse aina kääntää peliä erikseen mobiilille, vaan sen sijasta kehittäjä voi kääntää pelin suoraan omalle työkoneelleen ja kokeilla pelin toimivuutta suoraan työpöydällä.

Libgdx-sovelluskehyksellä kehittäminen tehdään natiivilla Java-ohjelmointikielellä. Libgdx-sovelluskehys tarjoaa kehittäjälle lisää työkaluja pelikehitykseen ja rakentaa projektille rungon, johon peliä voi rakentaa. Natiivilla Android SDK:lla pelin tekeminen vaatisi kehittäjän ohjelmoivan itse OpenGL-renderöinnin, tekstuurien hallinnan ja oman fysiikkamoottorin. [18.]

2.1 Libgdx-sovelluskehyn historia

Vuonna 2009 Mario Zechner aloitti projektin nimeltä AFX, jonka testaaminen oli hänelle erittäin rasittavaa, sillä hän joutui jokaisella testikerralla kääntämään projektin Android-laitteelleen. Tästä syystä hän muokkasi projektia, jotta hän kykeni testaamaan sitä suoraan omalla tietokoneellaan sen sijasta, että pitäisi joka kerta kääntää projekti Android-laitteelle. Tästä projektista syntyi myöhemmin Libgdx-sovelluskehys. [2.]

Vuonna 2010 Mario Zechner päätti tehdä AFX-sovelluskehksestään avoimen lähdekoodin projektin nimeltä Libgdx. Tässä vaiheessa Libgdx-sovelluskehys sisälsi yksinkertaiset rajapinnat grafiikalle, äänille ja I/O:lle. Samana vuonna Zechner sai projektilleen ensimmäisen avustajan, joka hänen mukaansa näytti maailmalle, että Libgdx-sovelluskehyksellä on tulevaisuus. Loppuvuodesta Zechner vaihtoi Libgdx-sovelluskehyn lisenssin Apachen 2:n avoimeen lisenssiin ja alkoi kirjoittaa Libgdx-sovelluskehksestä. [2.]

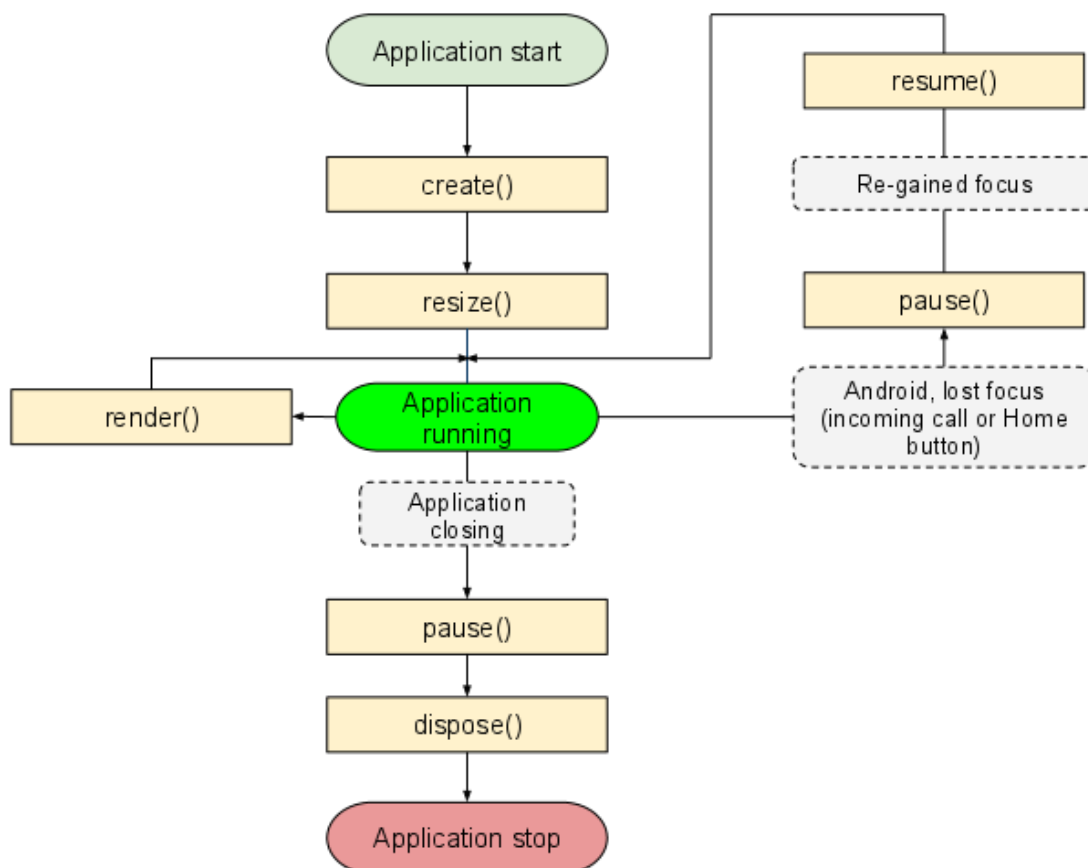
Vuosina 2011 ja 2012 Libgdx-sovelluskehys jatkoi kehitystään ja kasvuaan, ja Mario Zechnerin saatua projektiin enemmän avustajia hän pystyi keskittymään 3D-rajapinnan implementointiin Libgdx-sovelluskehukseen. Vuoden 2011 keväällä Mario Zechner sai kirjansa valmiiksi ja myöhemmin Libgdx-sovelluskehukseen tuli uusia parannuksia työpöytään liittyviin ominaisuuksiin sekä Box2D:n päivitystä. Vuonna 2012 Libgdx-sovelluskehukseen lisättiin lisäosat audion prosessoinnille ja FreeTypeille. Samana vuonna lisättiin taustajärjestelmä HTML:lle ja tehtiin asennussovellukselle käyttöliittymä, jolla pystyi helposti luomaan uusia Libgdx-projekteja. AppBrain samana vuonna ilmoitti, että 1,24 % Android-sovelluksista oli tehty Libgdx-sovelluskehystä käyttäen, mikä sillä hetkellä oli enemmän kuin muilla pelimoottoreilla ja sovelluskehyksillä tehtyjen pelien määrä Android-käyttöjärjestelmälle. Unity ohitti Libgdx-sovelluskehysten pian tehtyään mobiili-kehityksestä pelimoottorillaan ilmaisen. Samana vuonna Libgdx-sovelluskehukseen lisättiin iOS-tuki ja ensimmäiset Libgdx-sovelluskehysellä tehdyt pelit ilmestyivät App Storeen vuoden 2012 loppupuolella. [2.]

Vuonna 2013 Niklas Thernig kirjoitti Libgdx-sovelluskehukseen RoboVM-taustajärjestelmän, ja se lisättiin myöhemmin Libgdx-sovelluskehukseen. RoboVM mahdollistaa Java-ohjelmointikielellä tehtyjen natiivien mobiilisovellusten kääntämisen iOS-laitteisiin. Libgdx-sovelluskehysten päivityksen 0.9.9 myötä listättiin RoboVM-taustajärjestelmä, tuet Android x86:lle, Gradlille ja Mavenille sekä uusi rajapinta 3D:lle. [2.]

Nyt Libgdx-sovelluskehysten uusin versio on 1.9.2, joka sisältää muutamia virhekorjauksia. Libgdx-sovelluskehystä avustaa nykypäivänä 327 henkilöä, ja sovelluskehysellä on monta kolmannen osapuolen yritystä avustamassa ilmaisilla lisensseillä ja testilaitteilla. Libgdx-sovelluskehystä avustavat yritykset ovat RoboVM, Spine, Saikoa, Intel ja Excelsior JET. [3.]

2.2 Libgdx-sovelluskehysten elämäntaari

Libgdx-sovelluskehysten elämäntaaren (kuva 1) eri tapahtumat tulevat kehittäjän käyttöön implementoimalla ApplicationListener-rajapinnan. Tämän jälkeen Application-luokka kutsuu aina ApplicationListener-luokkaa, kun jokin tapahtuma käynnistyy. [4.]



Kuva 1. Libgdx-sovelluskehityksen elämänkaari [4].

ApplicationListener antaa käyttäjälle käyttöön eri metodeja, joita kutsutaan tiettyjen tapahtumien käynnistyessä. Metodit ovat create, resize, render, pause, resume ja dispose.

Create-metodi kutsutaan, kun sovellus tehdään. Resize-metodi ottaa parametreiksi vastaan kaksi kokonaislukua, joita käsitellään leveytenä ja korkeutena. Korkeus ja leveys käsitellään pikseleinä. Resize-metodi kutsutaan aina create-metodin jälkeen tai kun sovelluksen näytön kokoa muutetaan. Render-metodi kutsutaan Application-luokasta aina, kun näyttö pitää päivittää. Androidissa pause-metodi kutsutaan aina, kun sovellus piilotetaan esimerkiksi koti-nappulaa painamalla tai kun henkilö saa puhelun sovelluksen ollessa käynnissä. Työpöydässä pause-metodi kutsutaan ennen dispose-metodin kutsua suljettaessa sovellusta. Resume-metodia kutsutaan ainoastaan Android-laitteissa, kun palataan pause-metodista. Dispose-metodi kutsutaan aina, kun sovellus suljetaan.

Libgdx-sovelluskehityksessä ei ole yksittäistä pääsilmukkaa vaan ApplicationListener-luokan render-metodia yleensä käytetään pääsilmukkana, johon tehdään pelin logiikka.

2.3 Projektissa käytettyjä kirjastoja

Libgdx-sovelluskehys tarjoaa monia hyödyllisiä kirjastoja, joita insinööriöprojektissa käytettiin. Tässä esitellään joitakin projektissa käytettyjen kirjastojen ominaisuuksia ja mitä niillä saatiin aikaiseksi projektissa.

AssetManager

AssetManager-kirjasto mahdollistaa resurssien lataamisen asynkronisesti, mikä mahdollistaa esimerkiksi latauspalkin näyttämisen pelaajalle. AssetManager huolehtii, ettei toisi-taan riippuvia resursseja poisteta holtittomasti. [5.] Esimerkiksi, jos objekti A on riippu-vainen objektista B, objektia B ei poisteta, ennen kuin objekti A poistetaan.

Projektissa käytettiin tätä kirjastoa siitä syystä, että resurssien lataaminen asynkronisesti mahdollistaa alkuruutuun latauspalkin lisäämisen, joka näyttää pelaajalle prosentuaali-sesti, kuinka paljon on resurssien lataus edennyt. Tämä saa pelin näyttämään parem-malta ja valmiimmalta. AssetManager on myös hyvä paikka säilyttää kaikki resurssit.

Scene2D

Scene2D on kuin datastrukturi datan säilyttämiseen. Se tarjoaa työkaluja, joita voi käyt-tää esimerkiksi hierarkian rakentamiseen datojen kesken tai tehdä datalle ja objekteille erilaisia toimintoja. Scene2D:tä voi ajatella korkean tason sovelluskehiksenä, joka on rakennettu Libgdx-sovelluskehiksen päälle, johon oma peli rakennetaan. [6.]

Scene2D myös tarjoaa työkalut käyttöliittymän tekemiseen, ja sitä projektissa käytettiin. Peliin tehtiin päävalikko käyttäen Scene2D:n tarjoamia käyttöliittymäelementtejä ja lisät-tiin niille tarpeelliset toiminnot. Itse peliruudussa näytetään pelaajalle pelitilastoja, joiden näyttämiseen ja kokonaisuuden hallitsemiseen käytettiin Scene2D-kirjaston työkaluja.

Screen

Screen-luokka esittää luokan näkymänä, ja peli voi sisältää monta eri näkymää, kuten päävalikko ja pelinäkymä. Projektissa käytettiin tätä ominaisuutta esittämällä näkymät päävalikolle, pelinäkymälle ja pelin loppuruudun näkymälle. Libgdx-sovelluskehiksen

Game-luokka tarjoaa näkymän vaihtamiselle metodin, joka ottaa vastaan jonkin näistä näkymistä riippuen siitä, mihin näkymään halutaan vaihtaa.

Screen tarjoaa eri metodeja, joita kutsutaan Libgdx-sovelluskehiksen elinkaaren eri hetkinä. Show-metodi kutsutaan, kun näkymästä tulee päänäkö. Hide-metodi kutsutaan, kun näkymää vaihdetaan tai näkö ei ole enää päänäkönä Game-luokassa. Render kutsutaan aina, kun näkön pitäisi päivittää itseään. Dispose kutsutaan, kun näkön pitää vapauttaa kaikki näkössä olevat resurssit. Näitä metodeja käytetään hyödyksi projektissa eri tekstuurien näyttämiseen ja resurssien hallitsemiseen, kuten vapauttamiseen aina näkön ollessa piilossa.

Screen-luokka toteuttaa ApplicationListener-luokan, jonka myötä käyttöön tulevat metodit resize, pause ja resume.

2.4 Hyödyt ja heikkoudet

Libgdx-sovelluskehys antaa monia hyödyllisiä työkaluja pelinkehitykseen ja tuen kääntää peli monelle eri alustalle ilmaiseksi. Libgdx-sovelluskehiksellä on myös kattava yhteisö, joka päivittää ja lisää ominaisuuksia. Monialustainen kehitys myös säästää kehittäjältä paljon aikaa itse kehittämisessä ja pienentää oppimiskäyrää. Kehittäjän ei tarvitse esimerkiksi opetella pelinkehitystä natiivilla Android SDK:lla sillä Libgdx-sovelluskehys piilottaa kaiken natiivin Android SDK:n koodin. Libgdx-sovelluskehiksen avulla pelien testaamisen voi tehdä työpöydällä eikä peliä tarvitse jokaisen uuden ominaisuuden tai uuden testikierroksen takia kääntää puhelimelle vaan pelin voi kääntää suoraan työpöydälle ja testata peliä siinä.

Huonoja puolia Libgdx-sovelluskehiksessä on se, että kaikki pitää koodata käsin ja itse käyttöliittymäeditoreja ei ole natiivina, mikä lisää työmäärää. Tämä tarkoittaa, että jokainen elementti ja objekti on esimerkiksi siirrettävä kohdalleen koodissa eikä niitä voi raahata paikoilleen käyttöliittymän avulla kuten pelimoottoreissa. Pelimoottorit, kuten Unity ja GameMaker: Studio, tarjoavat omat käyttöliittymänsä, joilla pelinkehitys on helpompaa ja nopeampaa. Myös kolmiulotteisten pelien kehitys libgdx-sovelluskehiksellä on paljon vaikeampaa samasta syystä, eli käyttöliittymän puuttumisen vuoksi. Tästä syystä kehittäjille Unity saattaa näyttää paljon paremmalta vaihtoehdolta.

2.5 Muita sovelluskehyskiä ja pelimoottoreja

Taulukossa 1 esitellään Libgdx-sovelluskehyskiin eroavaisuuksia Unity-pelimoottoriin ja GameMaker: Studio-ohjelmaan verrattuna. Kaikissa on tuki monialustaiselle pelinkehitykselle, mutta esimerkiksi GameMaker: Studiolla joutuu mahdollisesti maksamaan, jos haluaa kääntää pelin tietyille alustoille.

Taulukko 1. Libgdx-sovelluskehyskiin, GameMaker: Studion ja Unityn eroavaisuudet. [1; 8; 9].

Sovelluskehys tai Pelimoottori	Ohjelmointikielet	Alustat	Hinta
Libgdx	Java	Kaikki tuetut	Ilmainen
Unity	UnityScript, C#	Kaikki tuetut	Ilmainen
Unity Pro	UnityScript, C#	Kaikki tuetut	75 \$/kk
GameMaker: Studio	Game Maker Language	Windows	Ilmainen
GameMaker: Studio Professional	Game Maker Language	Osittainen tuki	74,99 \$
GameMaker: Studio Master Collection	Game Maker Language	Kaikki tuetut	479,99 \$

Unity

Unity on Unity Technologiesin kehittämä monialustainen pelimoottori, jolla voi kehittää kaksi- ja kolmiulotteisia pelejä. Unityn tukemia alustoja ovat Windows, OS X, Unity Webplayer, Linux/Steam OS, Android, iOS, Blackberry 10, Tizen, Windows Phone 8, Playstation 3 ja 4, Playstation Vita, Wii U, Xbox One, Xbox 360, Oculus Rift ja Gear VR. [7.]

Unitylla on tarjolla ilmainen versio sekä maksullinen versio Unity Pro, joka antaa hieman enemmän vapautta peliä kehittäessä. Suurin eroavaisuus jonka kehittäjä huomaa Unityn ilmaisen version ja Unity Pro -version välillä on se, että Unity Pro -versiossa kehittäjä voi muokata pelin käynnistyskuvaa (kuva 2). Unityn ilmaisessa versiossa käynnistyskuvaa ei voi muokata tai vaihtaa, vaan käynnistyskuvana on Unityn oma käynnistyskuva. [8.] Unity-pelimoottoriin voi myös ladata ilmaisia tai maksullisia lisäosia, tekstuureja, ääniä, animaatioita ja muita resursseja Asset Storesta.

UNITY 5 What's included		PERSONAL EDITION	PROFESSIONAL EDITION
Engine with all features	?	✓	✓
Royalty-free	?	✓	✓
All platforms (limitations apply)	?	✓	✓
Customizable Splash Screen		✗	✓
Unity Cloud Build Pro - 12 Months	?	✗	✓
Unity Analytics Pro	?	✗	✓
Team License	?	✗	✓
Prioritized bug handling	?	✗	✓
Game Performance Reporting	?	✗	✓
Beta access	?	✗	✓
Unlimited Revenue and Funding	?	✗	✓
Future platforms included	?	✗	✓
Professional editor skin		✗	✓
Asset Store Level 11	?	✗	✓
Professional Community Features	?	✗	✓
Source code access	?	✗	\$
Premium Support	?	\$	\$
		FREE DOWNLOAD	FROM \$75/MONTH

Kuva 2. Unity-pelimoottorin ilmaisen ja maksullisen version eroavaisuudet [8].

Unity-pelimoottorilla kehittäjä voi ohjelmoida skriptejä kahdella eri ohjelmointikielellä. Ohjelmointikielet ovat C# ja UnityScript, joka muistuttaa erittäin paljon JavaScript- ohjelmointikieltä. UnityScript on patentoitu ohjelmointikieli, joka muistuttaa enimmäkseen Microsoftin JScript.NET:ä. [19.] UnityScript-ohjelmointikielessä ei ole tiettyjä ominaisuuksia, joita JavaScript-ohjelmointikieli tarjoaa, kuten with-lauseketta, delete-toimintoa tai mahdollisuutta käyttää säännöllistä lauseketta. [19.] Unityn tarjoama C#-ohjelmointikieli ei juurikaan eroa natiivista C#-ohjelmointikielestä, muuta kuin rajapintojen osalta.

Unity tarjoaa paljon enemmän alustoja, joille kehittää peli, kuin Libgdx-sovelluskehys. Erona on myös se, että Unity on kokonaisuudessaan pelimoottori toisin kuin Libgdx, joka on sovelluskehys. Libgdx-sovelluskehyksellä kehittäminen on enemmän koodipuolta, jossa ohjelmoidaan eri elementtien ja objektien sijainti. Unityn pääominaisuuksia on sen käyttöliittymäeditori, jolla voi helposti raahata ja pudottaa eri elementit ja objektit paikoilleen ja ohjelmoida eri objekteille omat toiminnot.

GameMaker: Studio

GameMaker: Studio on Mark Overmarsin tekemä ohjelma, jolla voi raahata ja pudottaa objekteihin eri skriptejä, jotka on kirjoitettu Game Maker Language -ohjelmointikielellä. GameMaker: Studio tukee useita alustoja, kuten Windows, Windows Phone, Mac OS X, Linux, HTML5, Android, iOS, Xbox One, Playstation 3, Playstation 4, Playstation Vita, Tizen ja Amazon Fire. [9.]

GameMaker: Studiosta on saatavilla riisuttu ilmainen versio ja maksulliset versiot GameMaker: Studio Professional ja GameMaker: Studio Master Collection. Maksulliset versiot tarjoavat eri ominaisuuksia, kuten mobiililaitteilla pelien testaamista, muokattavaa alku-ruutua ja pelien kääntämistä tuetuille alustoille (kuva 3). GameMaker: Studio Professionalin hinta on noin 75 dollaria ja GameMaker: Master Collectionin noin 480 dollaria jolla voi kääntää pelejä kaikille tuetuille alustoille. Suurin ero GameMaker: Studio Professionalin ja GameMaker: Studio Master Collectionin välillä on eri alustojen tuki. GameMaker: Studio Professionalissa on mahdollisuus maksaa eri alustatuista erikseen. Tämä antaa kehittäjälle mahdollisuuden ostaa tuet vain niille alustoille, joille hän pääasiallisesti kehittää. GameMaker: Studio Master Collectionin mukana tulee tuet kaikkiin alustoihin. [9.]

	Studio Free	Studio Professional	Studio Master Collection
Fully-Featured Engine	✓	✓	✓
Royalty-Free	✓	✓	✓
Marketplace Buying	✓	✓	✓
Player Export	✓	✓	✓
Desktop Exports	Windows Only	✓	✓
Customisable Splash Screen		✓	✓
Early Access		✓	✓
Marketplace Selling		✓	✓
Mobile Testing		✓	✓
Export Modules		Additional ⓘ	✓
	DOWNLOAD	FROM 149.99 \$74.99	FROM 479.99 \$479.99

Kuva 3. GameMaker: Studion eri versioiden eroavaisuudet [9].

GameMaker: Studio Professional versiossa eri alustamoduulien hinnat vaihtelevat noin 100 eurosta 150 euroon. Android-, iOS- ja Windows Phone -moduulit maksavat 149,99 \$. Tizen-moduuli maksaa 199,99 \$ ja HTML5-moduuli 99,99 \$. Xbox One-, Playstation

3-, Playstation Vita- ja Playstation 4 -moduulit tulevat ilmaiseksi GameMaker: Studio Professional -version mukana. Xbox One- ja Playstation-moduuleja käyttäessä kehittäjän täytyy olla rekistöitynyt Sony- ja Xbox-kehittäjäksi. GameMaker: Studio Professionalin mukana tulee myös 60 päivän ilmainen kokeilu-aika Amazon Fire-moduuliin. [9.]

Libgdx-sovelluskehys on GameMaker: Studioon verrattuna ilmainen ja tarjoaa myös pelin kääntämisen jokaiselle tuetulle alustalle ilmaiseksi. GameMaker: Studiossa peli ohjelmoidaan eri skriptejä kirjoittamalla ja liittämällä ne eri objekteihin Unity-pelimoottorin tavoin, toisin kuin Libgdx-sovelluskehyksessä, jossa kaiken joutuu tekemään ohjelmoidulla.

Muita

Muita mainitsemisen arvoisia vaihtoehtoja ovat Unreal Engine 4 ja Cocos2d. Unreal Engine 4 on Epic Games -yrityksen tekemä pelimoottori, joka Unity-pelimoottorin tavoin tarjoaa käyttöliittymän pelinkehitykseen. Unreal Engine -pelimoottorissa ohjelmointikielellä on UnrealScript. Cocos2D on avoimen lähdekoodin sovelluskehys, jolla voi tehdä pelejä eri alustoille, kuten Windows, OS X, Linux, iOS, Android ja Windows Phone 8. Cocos2D-sovelluskehyksellä on eri versioita, joilla voi kehittää peliä eri ohjelmointikielillä. Vaihtoehtoina ovat Ruby, Java, C++, C# ja Javascript.

3 Pelin suunnittelu

Insinööriyön pelin lajityypiksi valittiin Arcade, jossa on yleensä tarkoitus saada pisteitä ajan ja pelin etenemisen myötä. Arcade-lajityyppi valittiin oman mielenkiinnon takia ja myös sen takia, että tämä projekti olisi ensimmäinen oma Arcade-peliprojekti.

Peliin suunniteltiin myös erilaisia ominaisuuksia ja eri mahdollisuuksia, jotta omat taidot kehittyisivät ja mahdollisesti oppisi tämän insinööriyön myötä paljon uutta pelinkehityksestä. Pelistä oli tarkoitus saada nopeasti prototyyppi valmiiksi, jotta peliä voisi testata mahdollisimman aikaisin ja suunnitella sitä myötä peliin tarvittavia muutoksia ja lisäyksiä.

Pelin prototyypin ja ensimmäisen julkaisuversion mobiilille suunniteltiin olevan pelkästään Android-alustalle, sillä kehittäjällä on vain Android-alustan laitteita. Ensimmäisen julkaisun jälkeen suunnitellaan pelin kääntäminen iOS-laitteille.

Libgdx-sovelluskehys valittiin projektin tekoon siitä syystä, että aikaisempia projekteja oli tehty samalla sovelluskehyksellä ja haluttiin oppia lisää Libgdx-sovelluskehyksestä. Unity-pelimoottori ja muut vaihtoehdot eivät innostaneet siinä vaiheessa, sillä niillä pelien tekeminen oli jo entuudestaan tuttua eikä siitä olisi oppinut yhtä paljon ohjelmoinnista. Libgdx-sovelluskehys painottui enemmän pelien kehitykseen ohjelmoinnin kautta, mikä tuntui paljon opettavaisemmalta kokemukselta.

3.1 Peli-idea ja sen suunnittelu

Kriteereinä oli tehdä mahdollisimman nopeasti opittava ja yksinkertainen peli, joka myös antaisi haastetta pelaajalle. Nämä halutut kriteerit täyttää tietokonepeli Super Hexagon, josta projekti on saanut inspiraatiota.

Super Hexagon on Terry Cavanagh'n kehittämä videopeli, joka julkaistiin alun perin iOS-alustoille vuonna 2012. Kolme kuukautta myöhemmin peli julkaistiin Windows- ja OSX-käyttöjärjestelmille. Vuonna 2013 peli julkaistiin Android ja BlackBerry-alustoille ja Linux-käyttöjärjestelmille. [10.]

Super Hexagon (kuva 4) on erittäin nopeatempoinen peli, jossa liikutetaan kolmiota heksagonin ympäri ja yritetään välttää törmäystä vastaantuleviin seiniin. Peli vaatii erittäin nopeita refleksejä ja ajoitusta.



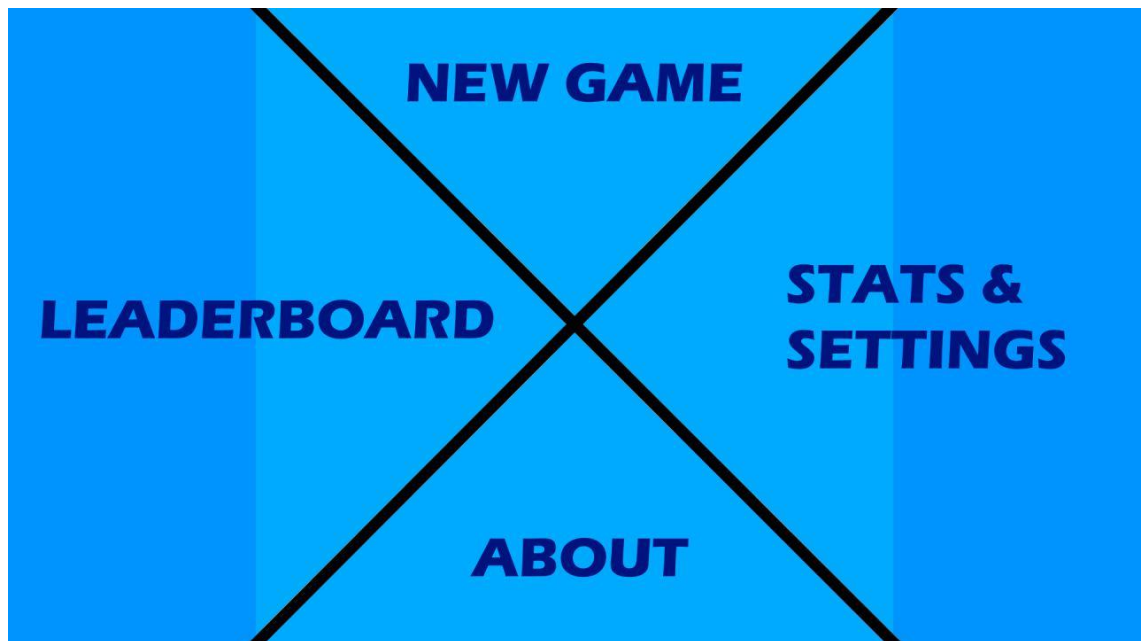
Kuva 4. Pelinäkymä Super Hexagon –pelistä.

Omaan peliin haluttiin samankaltaista vaikeutta ja mahdollisimman yksinkertaista pelinäkymää, jotta pelaaja oppisi pelin säännöt ja mekaniikat nopeasti. Peliin suunniteltiin tästä syystä yksinkertaiset valikot ja erittäin yksinkertainen käyttöliittymä sekä helposti opittavat pelimekaniikat.

Pelin prototyypin nimeksi tuli Redpill, koska pelissä esiintyy pääasiassa punaisen värisiä pillereitä, joita pelaaja joutuu tuhoamaan suuremman osan peliajasta. Projektin nimi saattaa vaihtua pelin julkaisun yhteydessä sopivammaksi ja kuvaavammaksi.

3.2 Pelin toiminta

Päävalikko pidettiin mahdollisimman yksinkertaisena, sillä pelin suunnittelussa pääprioriteetti oli pitää ulkonäkö yksinkertaisena ja mahdollisimman selkeänä pelaajalle. Päävalikossa navigoidaan koskettamalla haluttua tekstiä, minkä jälkeen siirrytään uuteen näkymään (Kuva 5).



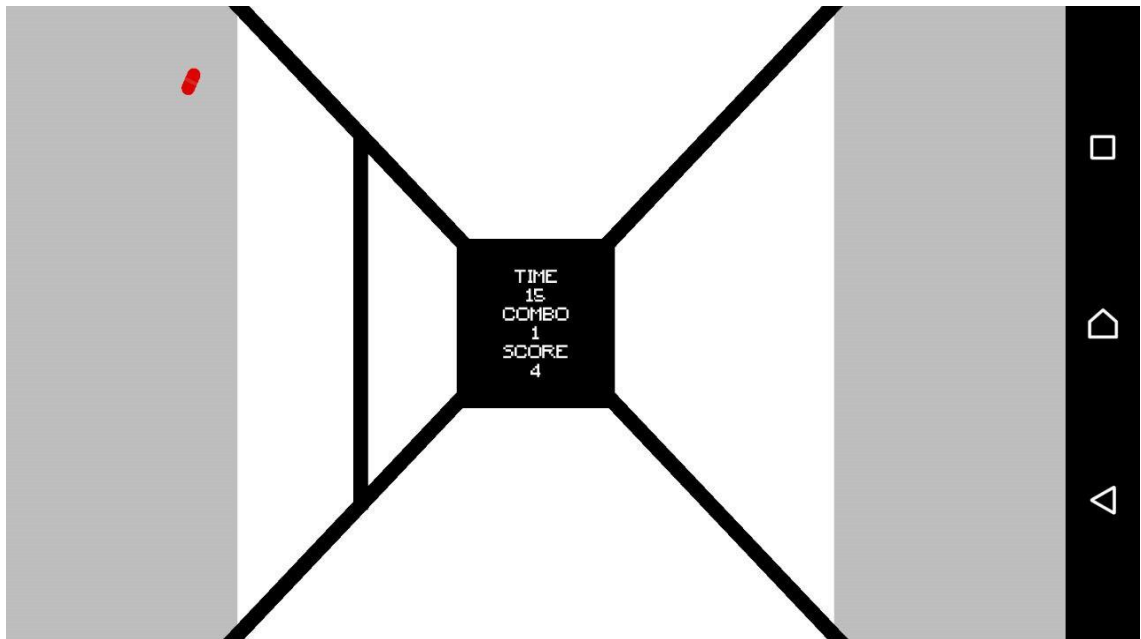
Kuva 5. Redpill-pelin päävalikko.

Peli aloitetaan New Game -vaihtoehtoa painamalla. Muita näkymiä pelin prototyyppiin ei ole toteutettu, sillä niiden toteutus oli tarkoitus tehdä valmiiksi vasta pelin jatkokehityksessä prototyypin valmistuttua.

Leaderboard-näkymässä pelaaja voisi verrata tuloksiaan muiden pelaajien kanssa ja mahdollisesti jakaa tuloksensa sosiaalisessa mediassa kuten Facebook tai Twitter. Stats & Settings -näkymässä näytettäisiin pelaajalle erilaisia tilastoja, kuten tuhottujen pillereiden määrä ja kokonaismäärä pelattuja tunteja. Näkymässä voisi myös vaihtaa asetuksia, esimerkiksi peliäänien mykistäminen.

About-näkymässä on suunniteltu kerrottavan pelin tekijöistä ja mahdollisesti käytetyistä työkaluista.

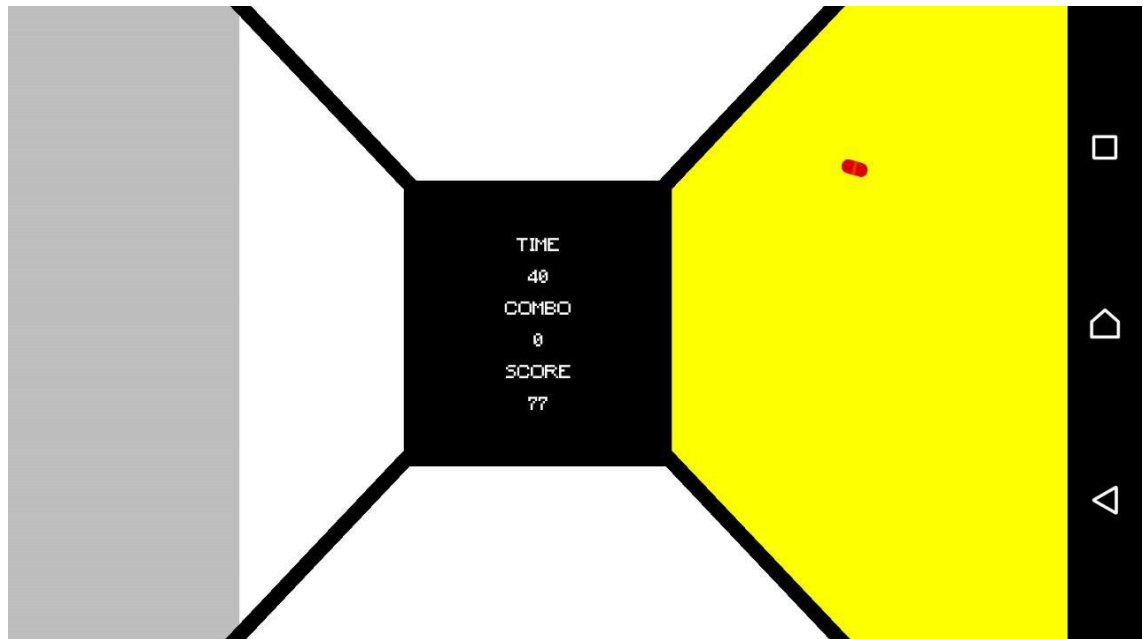
Pelin alkaessa näytön keskellä on neliö, jota pitää mahdollisimman pitkään pitää pienenä. Punaisia ja keltaisia pillereitä syntyy neljässä eri sektiossa satunnaisesti, ja ne alkavat sitä myötä lähestyä keskellä olevaa neliötä. Koskettamalla yhtä sektiota pelaaja luo viivan sektioon, joka osuessaan tuhoaa pillerin (kuva 6). Peli päättyy keskellä olevan neliön laajentuessa peliruudun ulkopuolelle.



Kuva 6. Pelinäköymä, jossa ollaan tuhoamassa punainen pilleri.

3.3 Pelin säännöt ja pelimekaniikka

Pelin tarkoituksena on siis tuhota viivan avulla punaisia pillereitä, jotta keskellä oleva neliö ei laajene peliruudun yli. Keltaisten pillereiden tuhoamista täytyy välttää, sillä jos ne osuvat keskellä olevaan neliöön, se kutistaa neliötä. Jos pelaaja tuhoaa keltaisen pillerin, sektio, jossa keltainen pilleri tuhottiin, on poissa pelaajan käytöstä muutaman sekunnin ajan, jolloin punaiset pillerit, voivat kulkea sektiossa vapaasti ja pelaaja ei pysty luomaan viivaa kyseiselle sektiolle (kuva 7). Poissa olevat sektiot värittyvät keltaiseksi, jotta pelaaja tietää, milloin viivan voi luoda sektioille.

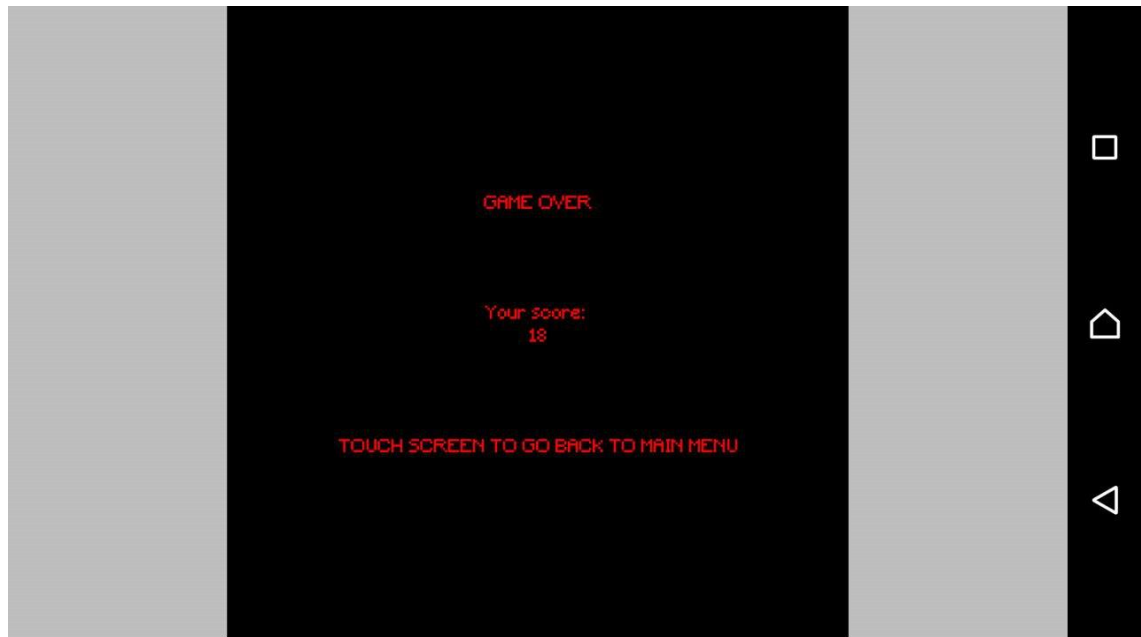


Kuva 7. Inaktiivinen sektio, jossa punainen pilleri pääsee vapaasti liikkumaan.

Keskellä olevassa neliössä näytetään pelaajalle kolme tietoa: pelin kesto, pelaajan kombokertoimien määrä ja pelaajan pistemäärä. Kombokerroin nousee yhdellä aina, kun pelaaja tuhoaa punaisen pillerin tai antaa keltaisen pillerin osua keskellä olevaan neliöön. Kombokerroin nollaantuu aina, kun punainen pilleri pääsee osumaan keskellä olevaan neliöön tai pelaaja tuhoaa keltaisen pillerin.

Pisteitä pelaaja saa aina, kun pelaaja tuhoaa punaisen pillerin tai keltainen pilleri osuu keskellä olevaan neliöön. Pisteitä annetaan aina yksi, jos kombokerroin on nolla. Jos kombokerroin on suurempi kuin nolla, pisteisiin lisätään yhden sijasta kombokerroin. Toisin sanoen kombokerroin kerrotaan annetulla pistemäärällä ja lisätään kokonaispistemäärään.

Kun keskellä oleva neliö kasvaa peliruudun ulkopuolelle, niin neliössä näytetään pelissä ansaitun loppupistemäärän tulos, ja painamalla ruutua palataan suoraan päävalikkoon (kuva 8).



Kuva 8. Pelin loppuruudun protoversio.

4 Pelin toteutus

4.1 Projektissa käytetyt työkalut

Redpill-peliä tehtäessä käytettiin erilaisia työkaluja pelinkehityksen helpottamiseksi ja pelin testaukseen. Vaikka Libgdx-sovelluskehys on alustariippumaton, saattaa silti eroavaisuuksia pelissä olla eri laitteiden välillä, esimerkiksi kuvasuhde. Tästä syystä peli käännettiin aina tietyin välein puhelimille, jotta pelin toiminta voidaan todeta myös kohdelaitteissa.

Android Studio

Android Studio IDE on virallinen ohjelmointiympäristö ohjelmistokehitykseen Android alustalle. Android Studion julkisti Google toukokuussa 2013. Android Studio on ladattavissa kaikille Apachen vapaan lisenssin alla (Apache license 2.0). [13.]

Sony Xperia Z3+ ja Oneplus 2

Sony Xperia Z3+ ja Oneplus ovat molemmat älypuhelimia, joissa on Android-käyttöjärjestelmä. Molempia puhelimia käytettiin pelin testaamiseen ja mahdollisten virheiden havaitsemiseen, joita ei esiinny välttämättä työpöytä-versiossa. Yksi ongelma, joka huomattiin vasta puhelimilla, oli kuvasuhteeseen liittyvä ongelma, jossa päävalikon tekstielementit olivat väärissä kohdissa toisin kuin työpöytä-versiossa, jossa kaikki näytti olevan oikein.

Git

Git on versionhallintaohjelmisto, jota projektissa käytettiin. Se mahdollistaa hajautetun työskentelyn ja estää datan katoamisen. [11.] Git-versionhallintaohjelmisto helpotti työskentelyä kannettavan tietokoneen ja pöytäkoneen välillä.

Bitbucket

Bitbucket tarjoaa Git-versiohallintaohjelmistoa käyttäville kehittäjille paikan, jossa voi säilyttää omia projekteja ja hallita niitä. Se tarjoaa käyttäjille yksityisiä ja julkisia säilytyspaikkoja. Bitbucket myös mahdollistaa muiden palvelua käyttävien kehittäjien osallistumisen projektiin. Verkkokäyttöliittymän avulla voi helposti hallita projektissa olevien käyttäjien oikeuksia ja itse projektia. [12.]

Projektiin lisättiin pelisuunnittelija, joka voi käydä muokkaamassa tiettyjä staattisia arvoja, kuten suhde punaisien ja keltaisten pillereiden syntymiseen. Pelisuunnittelijalle annettiin tästä syystä kirjoitus- ja lukuoikeudet.

Redpill-projektin koodit on tallennettu Bitbucketin julkiselle säilytyspaikalle kaikille nähtäväksi ja projekti aina pidetään aina avoimena muille.

4.2 Libgdx-projektin luominen

Libgdx tarjoaa oman Gradle-pohjaisen sovelluksen, joka luo projektin ja lataa sille tarvittavat riippuvaisuudet automaattisesti. Libgdx-sovelluskehys siis käyttää Gradlea eri riippuvuuksien hallitsemiseen automaattisesti. Tämä mahdollistaa sen, että eri kehittäjät voivat käyttää eri kehitysympäristöjä samassa projektissa. [14.]

Sovelluksessa täytyy täyttää muutama kohta, ennen kuin projekti luodaan. Projektia luodessa täytyy myös määrittää, missä sijaitsee Android SDK ja mihin polkuun kehittäjä haluaa projektinsa luoda. Lopuksi valitaan halutut alustat ja lisäkirjastot (kuva 9). Haluttaessa voi valita myös kolmannen osapuolen kirjastoja. Redpill-projektia varten valittiin kaikki mahdolliset alustat ja lisäkirjastoksi valittiin pelkästään FreeType.



Kuva 9. Libgdx-sovelluskehiksen asennussovelluksen käyttöliittymä.

Kun vaaditut kentät on täytetty ja halutut alustat ja kirjastot valittu, projekti luodaan Generate-nappia painamalla. Tämän jälkeen kehittäjällä on valmis Libgdx-projekti, jossa on valmiiksi generoitu perusrunko, jonka ympärille peliä lähdetään rakentamaan.

Projektihierarkiassa on tämän jälkeen monta eri kansiota, mutta tärkeimmät tälle projektille olivat kansiot android, core ja desktop. Android-luokassa voidaan tarvittaessa konfiguroida Androidille omat asetukset AndroidLauncher-luokassa. Desktop-kansion sisällä olevassa DesktopLauncher-luokassa voidaan asettaa työpöytäversion konfiguraatiot, esimerkiksi sovellusikkunan koko. Core-kansioon tehdään pelilogiikka ja itse pelin toteutus.

4.3 AndroidLauncher- ja DesktopLauncher-luokkien toteutus

Libgdx-sovelluskehys mahdollistaa Android-asetuksien konfiguroinnin. Asetuksiin pääsee käsiksi AndroidLauncher-luokassa config-muuttujan parametreja muuttamalla. Config-muuttuja on objekti AndroidApplicationConfiguration-luokasta, joka sisältää muutettavia parametreja, kuten useGyroscope, useCompass, hideStatusBar, disableAudio ja useImmersiveMode. [20.]

DesktopLauncher-luokassa voi vaihtaa työpöytään liittyviä konfiguraatioita käyttämällä DesktopLauncherissa olevaa config-muuttujaa. Config-muuttuja on objekti LwjglApplicationConfiguration-luokasta, ja sillä kehittäjä voi halutessaan muuttaa parametreja, kuten fullscreen, foregroundFPS, backgroundFPS, disableAudio, width ja height. [21.]

Redpill-projektissa Android-asetuksista muutettiin vain useImmersiveMode- ja hideStatusBar-muuttujat, joiden boolean-arvot muutettiin todeksi. Näiden asetusten muuttaminen piilottaa käyttöjärjestelmän palkit ja käyttöliittymän napit. DesktopLauncher-luokassa vaihdettiin width- ja height-arvot, jotta työpöytäversion käynnistyessä pelin resoluutio olisi mahdollisimman sopivan kokoinen eikä liian pieni tai suuri (kuva 10).

```
public class DesktopLauncher {
    public static void main (String[] arg) {
        LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
        config.width = 1280;
        config.height = 720;
        new LwjglApplication(new RedPill(), config);
    }
}
```

Kuva 10. DesktopLauncher-luokan toteutus.

Näiden kahden luokan lisäksi Libgdx-sovelluskehityksellä on vastaavanlaiset luokat HTML5- ja iOS-alustoille. Näille alustoille konfiguraatiot tehdään samalla periaatteella kuin AndoridLauncher- ja DesktopLauncher-luokissa config-muuttujan avulla. Config-muuttuja HTML5- ja iOS-alustojen konfigurointiin löytyy GwtLauncher- ja IOSLauncher-luokasta.

Redpill-projektissa ei muutettu IOSLauncher- tai GwtLauncher-luokista mitään, sillä pelin prototyypin oli tarkoitus olla käännettävänä ainoastaan Android-alustoille. Jatkokehityksessä lisätään IOSLauncher-luokkaan omat konfiguraatiot vasta, kun pelin julkaisu iOS-laitteisiin on ajankohtainen. HTML5-tukea pelille ei ole vielä suunniteltu, ja peliä ei välttämättä käännetä ollenkaan HTML5-alustalle.

4.4 AssetManager-luokan toteutus

Jotta eri tekstuureja ja resursseja voidaan kutsua, pitää tehdä ensin luokka resurssien hallintaan. AssetManager-luokan käyttö ei ole pakollista, mutta se tuo eri ominaisuuksia resurssien hallitsemiseen ja resurssit voidaan pitää yhdessä paikassa. Luokan metodeista tehtiin staattisia, jotta metodeja voi kutsua projektin jokaisessa luokassa teemmättä uutta instanssia AssetManagerista.

AssetManager-luokan lisääminen projektiin (kuva 11) on vaivatonta, minkä takia se lisättiin projektiin, vaikka projektissa ei ole paljon resursseja. Luokkaan tehtiin neljä metodia, load, getAtlas, getInGameFont ja dispose. Load-metodi kutsutaan aivan alussa, ennen kuin mihinkään näkymään siirrytään. Se lataa kaikki tarvittavat resurssit käyttövalmiiksi. GetAtlas- ja getInGameFont-metodien kutsulla saadaan tarvittaessa haluttu atlas tai pelissä käytettävä fontti. Dispose-metodi huolehtii resurssien vapauttamisesta, ja se kutsutaan peliä suljettaessa.

```

/**
 * Loads all the assets
 */
public static void load(){
    manager.load(ATLAS_INGAME, TextureAtlas.class);
}

public static TextureAtlas getAtlas(String atlasName){
    return manager.get(atlasName, TextureAtlas.class);
}

public static FileHandle getInGameFont(){
    return Gdx.files.internal("fonts/ingame.ttf");
}

/**
 * Disposes all the assets
 */
public static void dispose(){
    manager.dispose();
}

```

Kuva 11. AssetManager-luokan toteutus projektissa.

4.5 Pelinäkömön toteutus

Pelinäkön toteutus tehtiin GameScreen-nimiseen luokkaan, joka toteuttaa Screen-luokan. Tämä antaa käyttöön show-metodin, jota käytetään ikään kuin luokan konstruktorina, ja render-metodin, jossa on toteutettu pelilogiikka.

Show-metodissa alustetaan kaikki elementit, kuten keskellä oleva neliö ja eri sektiot. Metodissa myös alustetaan fontti, jota käytetään tilastojen näyttämiseen pelaajalle pelin aikana. Show-metodissa myös alustetaan PillManager-luokka, joka on erillinen luokka pillerien hallintaan. Pillerien toiminta ja logiikka haluttiin pitää erillään pääluokasta, jotta koodin lukeminen olisi helpompaa. Show-metodin lopussa aloitetaan efekti, joka kutistaa keskellä olevan neliön alkuperäiseen kokoonsa skaalatusta koosta. Efektin loppuessa itse peli alkaa. Itse pillereiden luominen hoituu vertailemalla oman ajastimen arvoa MathUtils.random-metodin palauttamaan arvoon. MathUtils.random palauttaa arvon annetun minimin ja maksimin väliltä, ja kun ajastimen arvo ylittää tämän arvon, luodaan satunnaisesti keltainen tai punainen pilleri satunnaisessa sektiossa (kuva 12). Pillerin värin luominen tapahtuu PillManager-luokassa.

```
// Timed spawn
delayTimer += delta;
if (delayTimer >= MathUtils.random(DELAY_SPAWN_MIN, delaySpawnMax)) {
    pillManager.spawn(sections[MathUtils.random(3)]);
    delayTimer = 0f;
}
```

Kuva 12. Toteutus pillereiden luomiseen PillManagerin spawn-metodia kutsumalla (liite 2).

Render-metodissa tapahtuu itse pelilogiikka, kuten kosketustapahtumien hallinta ja pillereiden luominen. Luokkaan on asetettu muuttujat pillereiden luomisen ajoittamiseen. Aina kun peliä on kulunut 20 sekuntia, muuttujan delaySpawnMax arvoa vähennetään. Tämä pienentää maksimiviivettä uuden pillerin luomiseen ja vaikeuttaa peliä ajan kuluessa.

Render-metodissa myös piirretään kaikki objektit ja elementit SpriteBatchin begin- ja end-metodikutsujen välissä. SpriteBatch huolehtii tekstuurien piirtämisen ruudulle ja ottaa tekstuurin geometrian talteen lähettämättä sitä heti näytönohjaimelle. Se ei myöskään lähetä tekstuureja näytönohjaimelle yksittäin vaan lähettää monta samanaikaisesti, mikä on tehokkaampaa.

Renderissä kutsutaan myös checkCollision- ja handleInput-metodeja, jotka ovat omia metodeja eri tapahtumien ja kosketuksien hallitsemiseen. HandleInputissa katsotaan, mihin kohtaan ruutua pelaaja koskettaa ja rakennetaan pelaajalle viiva koskettuun sektioon, millä voidaan tuhota pillereitä. Kosketuksen sijainti käytetään Libgdx-sovelluskehysen tarjoamilla Gdx.getInputX- ja Gdx.getInputY-metodeilla (kuva 13).

```

private void handleInput(){
    inputPos.set(Gdx.input.getX(), Gdx.input.getY(), 0);
    camera.unproject(inputPos);
    if(Gdx.input.isTouched()){
        // If the section is inactive then don't draw the line in the section
        for(Section s : sections){
            if(s.getSectionPolygon().contains(inputPos.x, inputPos.y) && !s.isActive()){
                playerLine.resetHitBox();
                return;
            }
        }

        // If player touches the center box then don't draw anything
        if(!centerObject.getBoundingBox().contains(inputPos.x, inputPos.y)){
            for(Section s : sections){
                if(s.getSectionPolygon().contains(inputPos.x, inputPos.y)) {
                    playerLine.draw(batch, inputPos, s.getSectionName());
                }
            }
        } else {
            playerLine.resetHitBox();
        }
    } else {
        playerLine.resetHitBox();
    }
}

```

Kuva 13. Pelaajan kosketusten toteutus (liite 1).

CheckCollision-metodissa katsotaan kaikki mahdolliset törmäykset kahden objektin välillä ja tehdään eri toimintoja törmäyksen tapahduttua, kuten annetaan lisää pisteitä tai poistetaan sektio pelaajan käytöstä hetkeksi. Jos jokin sektio on pois käytöstä, kaikki sektioon luodut pillerit lisätään removablePills-listaan, joka poistaa keltaiset pillerit (kuva 14).

```

private void checkCollision(){
    for(Pill pill : pillManager.getPills()){
        // Collision with the center object
        if(Intersector.overlapConvexPolygons(centerObject.getHitBox(), pill.getHitbox())){
            pillManager.getRemovablePills().add(pill);

            // Scale the center object if the colliding object is red pill
            if(pill.getPillColor().equals(Assets.INGAME_REDPILL)) {
                ui.setCombo(0);
                Effects.grow(centerObject, 20, 0.3f);
            } else {
                ui.setCombo(ui.getCombo() + 1);
                ui.setScore(ui.getScore() + (ui.getCombo()*2));
                Effects.shrink(centerObject, 10f, 0.3f);
            }
        }

        // Collision with the player line
        if(Intersector.overlapConvexPolygons(pill.getHitbox(), playerLine.getHitbox())){
            pillManager.getRemovablePills().add(pill);

            if(pill.getPillColor().equals(Assets.INGAME_YELLOWPILL)){
                // Check in which section the pill was and inactivate the section
                for(Section s : sections){
                    if(Intersector.overlapConvexPolygons(s.getSectionPolygon(), pill.getHitbox())){
                        ui.setCombo(0);
                        s.setActive(false);
                    }
                }
            } else {
                ui.setCombo(ui.getCombo() + 1);
                ui.setScore((ui.getScore()+1) + ui.getCombo());
            }
        }
    }

    // Remove all yellow pills from inactive sections
    for(Section s : sections){
        if(!s.isActive() && pill.getPillColor().equals(Assets.INGAME_YELLOWPILL)){
            if(Intersector.overlapConvexPolygons(s.getSectionPolygon(), pill.getHitbox())) {
                pillManager.getRemovablePills().add(pill);
            }
        }
    }
}

```

Kuva 14. CheckCollision-metodin toteutus (liite 1).

Lopuksi render-metodissa kutsutaan updateGameState-metodia, joka katsoo, onko keskellä oleva neliö ylittänyt pelirajan. Jos keskellä oleva neliö on ylittänyt pelirajan, peli loppuu ja siirrytään loppuruutuun (kuva 15).

```

private void updateGameState(){
    if(centerObject.getBoundingBox().x < 0 || centerObject.getBoundingBox().getWidth() > CAMERA_WIDTH
        || centerObject.getBoundingBox().y < 0 || centerObject.getBoundingBox().getHeight() > CAMERA_HEIGHT){
        game.setScreen(new GameOverScreen(game, ui.getScore()));
    }
}

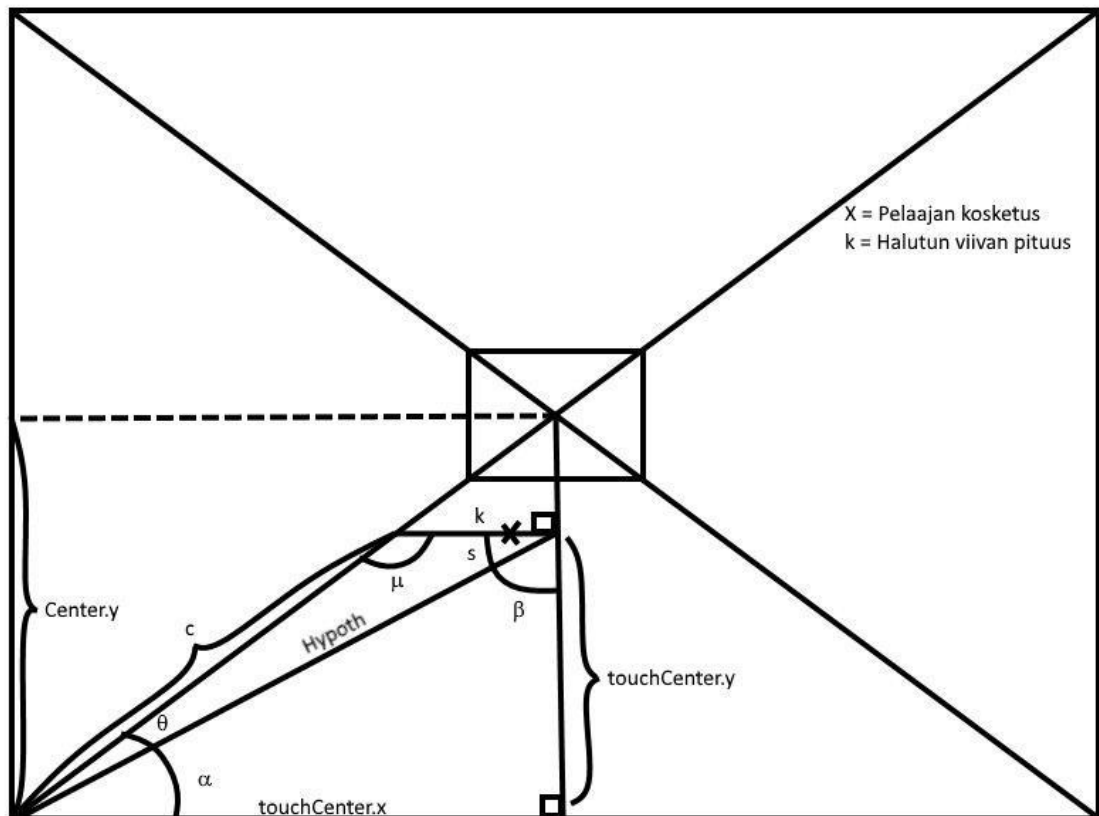
```

Kuva 15. UpdateGameState-metodin toteutus.

4.6 Pelaajan viivan luominen

Pelaajan viivan luominen oli projektin haastavin osuus, sillä siinä piirretään viiva sektorin päästä päähän riippumatta siitä, mihin kohtaan sektiota pelaaja koskee. Viivan koko vaihtelee y-akselilla liikkuesssa ja pelaajan viivan leveyttä muutetaan koodissa sen mukaan missä pelaajan kosketus on y-akselilla. Tämä toteutettiin matemaattisesti trigonometrian avulla ja trigonometrisillä kaavoilla.

Ongelmana oli siis, miten pelaajan viiva saadaan venytettyä koodissa dynaamisesti, niin että viiva alkaa alusta sektorin loppuun. Ongelmaa lähdettiin ratkaisemaan piirtämällä kuva paperille pelitilanteesta ja piirtämällä kolmiot sekä kulmat, joita halutaan ratkaista, jotta saataisiin viivan pituus selville (kuva 16).



Kuva 16. Viivan pituuden laskennan havainnollistus.

Kuvasta 16. haluttiin siis selvittää, mikä on k -kateetin pituus, kun tiedossa on ainostaan näytön korkeus jaettuna kahdella eli $Center.y$ -arvo, $touchCenter.x$, joka on näytön leveys jaettuna kahdella, sekä $touchCenter.y$, joka on pelaajan kosketuksen sijainti y -akselilla. Kosketuksen sijainti y -akselilla saadaan metodilla `Gdx.getInputY`.

Ihan aluksi selvitettiin suorakulmaisen kolmion hypotenuusan $Hypoth$ -arvo, joka saadaan laskettua Pythagoraan lauseella kaavan 1 mukaan.

$$Hypoth = \sqrt{touchCenter.x^2 + touchCenter.y^2} \quad (1)$$

Seuraavaksi laskettiin kulman α -arvo, jonka saa laskettua trigonometrisilla funktioilla. Tässä tapauksessa α -kulma laskettiin sinifunktion avulla kaavan 2 tapaan.

$$\sin(\alpha) = \frac{touchCenter.y}{Hypoth} \quad (2)$$

Tiedossa ovat nyt ensimmäisen suorakulmaisen kolmion kaikki kulmat paitsi β -kulma, joka ratkaistiin seuraavaksi. Trigonometrian sääntöjen mukaan kolmion kulmien summa on aina 180 astetta. Tämän tiedon avulla β -kulma saadaan laskettua, sillä α -kulma tunnetaan, ja koska kolmio on suorakulmainen, yksi kulmista on varmasti 90 astetta. Näillä tiedoilla saadaan muodostettua kaavan 3 mukainen laskutoimitus.

$$\beta = 180 - 90 - \alpha \quad (3)$$

Nyt tunnetaan ensimmäisen kolmion kaikki kulmat ja kateetit. Seuraavaksi laskettiin yhden sektorin muodostama suorakulmainen kolmio ja sen ensimmäinen kulma z , joka on yhtä kuin tuntemattoman θ -kulman ja α -kulman summa, kuten kaavassa 4 todetaan.

$$z = \theta + \alpha \quad (4)$$

Tiedossa ovat siis $touchCenter.x$ ja näytön korkeus jaettuna kahdella eli $Center.y$. Näiden tietojen avulla z -kulma saadaan laskettua tangenttifunktiolla kaavan 5 mukaan.

$$\tan(z) = \frac{Center.y}{touchCenter.x} \quad (5)$$

Kun tiedossa on z -kulma, tuntematon kulma θ saadaan helposti laskettua kaavan 6 mukaan.

$$\theta = z - \alpha \quad (6)$$

Seuraavaksi laskettiin kulma s , koska kolmion kateetti on suora ja se muodostuu 180 asteen kulmasta, josta tiedossa on 90 asteen kulma ja β -kulma. Jäljelle jää siis kulma s , joka saadaan helposti kaavan 7 mukaan laskettua.

$$s = 180 - 90 - \beta \quad (7)$$

Lopuksi kun tiedossa on kaikki tarvittavat kulmat ja kateettien pituudet, joilla voidaan laskea haluttu k -kateetin pituus, trigonometrian sinilauseen avulla voidaan muodostaa kaavan 8 mukainen lauseke.

$$\frac{\sin(\mu)}{\text{Hypoth}} = \frac{\sin(\theta)}{k} = \frac{\sin(s)}{c} \quad (8)$$

Kaavasta 8 haluttiin ratkaista k -arvo, ja koska tiedossa olivat kulmat μ ja θ sekä Hypoth -arvo, k -arvo saatiin ratkaistua kaavan 9 avulla.

$$k = \frac{\sin(\theta)}{\sin(\mu)} * \text{Hypoth} \quad (9)$$

Kyseinen k -arvo on kuitenkin vain puolet koko sektorin leveydestä, ja tämän takia kaavan 9 tulos pitää vielä kertoa kahdella. `CalculateLineAttribute` metodi palauttaa vektorin, jonka kaksi arvoa ovat viivan alkupiste ja loppupiste eli juuri laskettu $2k$ -arvo. Tämän avulla voidaan laskea viivan pituus dynaamisesti ja piirtää se sektiolle oikein riippumatta kosketuksen sijainnista (kuva 17).


```

private Vector2 calculateLineAttributes(String section, Vector3 inputPos){
    Vector2 centerPos = new Vector2();
    Vector2 touchCenter = new Vector2();

    if(section.equals(Section.SECTION_LEFT) || section.equals(Section.SECTION_RIGHT)){
        centerPos.set(GameScreen.GAME_AREA/2f, GameScreen.GAME_AREA/2f);
        touchCenter.set(centerPos.x, inputPos.x);
    } else {
        centerPos.set(GameScreen.GAME_AREA/2f, GameScreen.GAME_AREA/2f);
        touchCenter.set(centerPos.x, inputPos.y);
    }

    double hypoth = Math.hypot(touchCenter.x, touchCenter.y);
    double alpha = Math.toDegrees(Math.asin(touchCenter.y / hypoth));
    double beta = 180 - 90 - alpha;

    double zeta = Math.toDegrees(Math.atan(centerPos.y / centerPos.x));
    double theta = zeta - alpha;
    double sigma = 180 - 90 - beta;
    double mu = 180 - theta - sigma;

    double phase1 = Math.sin(Math.toRadians(theta)) * hypoth;
    double k = phase1 / Math.sin(Math.toRadians(mu));

    double result = touchCenter.x - k;
    return new Vector2((float)result, (float)k*2f);
}

```

Kuva 17. Toteutus viivan arvojen laskemiseen.

5 Johtopäätökset ja jatkosuunnitelmat

5.1 Johtopäätökset

Libgdx-sovelluskehys vaatii kehittäjältä paljon enemmän ohjelmointiosaamista pelin kehitykseen kuin muut esitellyt vaihtoehdot. Peliä kehittäessä huomasin, että asiat, joiden tekemiseen meni monta päivää, ja monet asiat, joita tein käsin tyhjästä, saa tehtyä Unity-pelimoottorilla vaivattomasti.

Esimerkkinä pelaajan viivan piirtäminen näytölle: Unity-pelimoottorilla olisin voinut asettaa käyttöliittymää käyttäen kaksi pistettä sektoreiden viivoihin ja piirtää viivan pisteiden välille. Libgdx-sovelluskehystä käyttäen joutuu laskemaan ensin sijainnit näille pisteille ja sen jälkeen vasta voi piirtää, eli matemaattisesta osuudesta olisin päässyt kokonaan eroon. Päävalikon tekeminen olisi vaatinut vain elementtien asettamisen kohdalleen ja

eri napeille skriptin lisäämisen, joka vain lataa peliruudun. Tämä olisi saavutettu yhdellä koodirivillä.

Unity-pelimoottorin käyttöliittymä olisi säästänyt paljon aikaa ja vaivaa peliä tehdessä jo pelkästään käyttöliittymänsä ansiosta. Jos alusta asti lähtisin tekemään samaa peliä, tekisin sen ehdottomasti Unity-pelimoottorilla, sillä se tarjoaa samat ominaisuudet kuin Libgdx-sovelluskehys ja on saatavilla ilmaiseksi. Kehitysaikaa olisi jäänyt paljon enemmän, ja olisin voinut lisätä prototyyppiin efektejä tai lisää ominaisuuksia.

Vaikka Unity-pelimoottorilla pelin tekeminen ei ole täysin ilmaista, jos haluaa tehdä oman käynnistyskuvan, se olisi silti ollut parempi vaihtoehto Libgdx-sovelluskehysten tilalle tässä projektissa sillä tavoitteena ei ollut valmis tuote markkinoille vaan pelkästään pelin prototyyppi. Jos prototyyppi vastaa odotuksia, luultavasti peli tehdään Unity-pelimoottorilla alusta asti riippuen lopputuloksesta.

Libgdx-sovelluskehystä on käytetty peliyrityksissä, mutta sen 3D-rajapinnan ja monialustaisuuden rajoitusten takia moni peliyritys, kuten CreatioSoft, on siirtynyt käyttämään Unity-pelimoottoria [22]. Uskon, että Libgdx-sovelluskehysten käyttö ei tule lisääntymään yrityskäytössä, sillä monet muut pelimoottorit tarjoavat paljon enemmän kuin Libgdx-sovelluskehys ja vaativat paljon vähemmän työtä samankaltaisten tuloksien saavuttamiseen. En usko Libgdx-sovelluskehysten tulevan suosioon peliyrityksissä lähitulevaisuudessa, sillä kehittäminen Libgdx-sovelluskehysellä on vielä vaivalloista ja hankalaa. Unity-pelimoottori on tällä hetkellä erittäin suosittu mobiilikehityksessä ja monet suuret yritykset käyttävät sitä pelinkehitykseen. Unity-pelimoottoria käyttäviä yrityksiä ovat Cartoon Network, Coca-Cola, LEGO, Electronic Arts, NASA, Microsoft, Nexon, Square, Ubisoft, Nickelodeon ja Nexon. [23.]

Unity-pelimoottori siis tarjoaa ketterää kehitystä ja säästää paljon kehitysaikaa, mikä on monessa peliprojektissa tärkeää. Vaikka Libgdx-sovelluskehys tarjoaa monia samoja ominaisuuksia, yksinkertaistenkin asioiden tekemiseen saattaa mennä viikkoja ja käyttöliittymän puuttuminen Libgdx-sovelluskehyksestä tuo paljon vaikeuksia mukanaan.

5.2 Jatkosuunnitelmat

Redpill-pelin prototyypin valmistuttua saadaan selkeämpi kuva siitä, minkälainen pelistä tulee, ja prototyypin avulla saadaan mahdollisesti parannusehdotuksia. Prototyypin valmistuttua voidaan myös testata peliä ja katsoa, onko pelimekaniikassa korjattavaa ja toimiiko pelin yleinen idea.

Peli on tarkoitus julkaista Android-alustoille ja myöhemmin vasta iOS-alustoille. Peliin on suunniteltu monta eri lisätoimintoa ja eri visuaalisia efektejä, jotta pelin pelaaminen olisi mukavampaa pelaajalle ja jotta peli pysyisi miellyttävän näköisenä.

Google Play Developer API

Google Play Developer API mahdollistaa kehittäjän tekemän sovellukselleen tai pelilleen erilaisia hallintatoimintoja julkaisua ja itse sovellusta varten [16]. Esimerkkinä ovat mahdollisuus lisätä sovellukseen sisäisiä tuotteita, joita käyttäjä voi ostaa.

Redpill-projektiin on tarkoitus integroida Google Play -ohjelmistorajapinta, jonka avulla voidaan rakentaa oma High Score -taulu, jossa näkyvät kaikkien pelaajien pistemäärät suuruusjärjestyksessä ja oma sijainti High Score -taulussa. Tämän ominaisuuden myötä säästyy paljon aikaa, sillä kehittäjän ei tarvitse kehittää omaa toteutusta High Score -taulusta.

Peliin on suunniteltu myös sisäisiä tuotteita, jotka auttavat pelaajaa pelissä esimerkiksi tuhoamalla kaikki pillerit näytöltä samanaikaisesti. Sisäisiä ostoksia ei ole vielä lukkoon lyöty, ja niiden suunnittelu tehdään pelin jatkokehityksessä.

Julkaisu Google Play -palveluun

Google Play on Googlen omistama palvelu, joka sisältää musiikkia, elokuvia, kirjoja sekä sovelluksia ja pelejä, jotka ovat yhteensopivia Android-laitteilla. Se sisältyy useimpiin Android-laitteisiin. Google Play -palvelulla on tällä hetkellä noin miljardi aktiivista käyttäjää, ja se on saatavilla yli 190 maassa. [17.]

Redpill-projekti haluttiin alusta lähtien julkaista Google Play -palveluun, mikä tuo monia eri haasteita ja hyötyjä mukanaan. Google Play -palvelussa oli vuonna 2015 marraskuussa noin 1,8 miljoonaa sovellusta. [15.] Redpill-projektilla on siis suuri mahdollisuus upota muiden sovelluksien sekaan ja jäädä huomaamattomaksi, varsinkin kun projektia ei markkinoida muualla kuin sosiaalisessa mediassa ja foorumeissa.

Julkaistu App Store -sovelluskauppa

App Store on Applen oma sovelluskauppa, joka Google Play -palvelun tavoin tarjoaa sovelluksia käyttäjille. App Store -sovelluskaupan sovellukset ovat yhteensopivia Applen iOS-laitteisiin.

Redpill-projekti halutaan myös lopuksi julkaista App Store -sovelluskauppaan, mutta pelin julkaiseminen ja kääntäminen testaamista varten iOS-laitteilla vaatii Mac-tietokoneen, jolla ohjelma käännetään yhteensopivaksi iOS-laitteille. Projektia tehdessä ei ole ollut Applen laitetta käytettävissä, minkä takia App Store -sovelluskauppaan julkaisu on ollut vain suunnitelmissa.

6 Yhteenveto

Insinööritöinä suunniteltiin pelille prototyyppi, jota voidaan testata ja suunnitella pelille lisäominaisuuksia jatkokehitystä varten. Pelin prototyypin sain sellaiseksi kuin halusin eli valmiiksi prototyyppiksi, jota voin jatkokehittää. Jatkotoimenpiteenä on testata peliä ja suunnitella jatkokehitystä, jotta jonain päivänä peli saataisiin julkaistua Google Play -palvelussa. Vaikka prototyyppissä tuli paljon ongelmia vastaan, sain kaikki ongelmat ratkaistua, vaikka siihen ylimääräistä aikaa kuluikin. Yksinkertaisten efektien lisääminen prototyypin kehityksen loppuvaiheissa antoi pelille viimeistellyn ilmeen ja sai prototyypin näyttämään paljon paremmalta ja miellyttävämmältä. Myös tiimityöskentely pelisuunnittelijan kanssa oli miellyttävää, ja hänen kanssaan ideointi oli erittäin tuottoisaa. Jatkossa toivon, että ideointia ja suunnittelua pelisuunnittelijan kanssa tehtäisiin enemmän.

Hieman parannettavaa olisi koodin dokumentoinnissa koodin kommentoinnissa, joihin seuraavaksi käytän kehitysaikani. Olisin myös prototyyppiin halunnut enemmän efektejä, jotka saisivat pelin näyttämään monta kertaa paremmalta, mutta se ei ollut olennaisinta

prototyypin tekemisessä, minkä takia jätin monimutkaisemmat efektit tekemättä. Prototyyppiä kehittäessä tuli vastaan paljon ongelmia, kuten matemaattisten kaavojen lisääminen peliin. Tämän takia aikaa prototyypin tekemiseen kului paljon enemmän kuin oli tarkoitus. Mielestäni ennen pelin kehittämistä olisi pitänyt suunnitella prototyypin kehitysvaiheet tarkemmin ja ylipäättänsä käyttää enemmän aikaa pelin prototyypin suunnitteluun.

Libgdx-sovelluskehiksen käyttäminen peliprojektiin riippuu mielestäni kehittäjän mieltymyksestä, mitä hän haluaa käyttää pelinkehitykseen. Libgdx-sovelluskehys tuo mukanaan paljon hyviä kirjastoja kehittäjän käyttöön, mikä helpottaa ohjelmointia, mutta ongelmana näin jokaisen eri toiminnon ja ominaisuuden koodaamisen. Unity-pelimoottori tuo paljon helpomman lähestymistavan pelikehitykseen, ja sillä saa paljon aikaiseksi paljon nopeammin. Minun olisi kannattanut tehdä prototyyppi mahdollisimman nopeasti Unity-pelimoottorilla testattavaksi ja sitten jatkosuunnittelussa käydä läpi, käytetäänkö muuta pelimoottoria tai sovelluskehitystä pelin toteuttamiseen vai jatketaanko Unity-pelimoottorilla ominaisuuksineen ja virheineen. Prototyyppiin ei saisi kulua paljon aikaa, ja minulla kului suunnitellun parin kuukauden sijasta neljä kuukautta pelin prototyypin tekemiseen. Sen olisi voinut hoitaa Unity-pelimoottorilla hyvässä tapauksessa kuukauden sisällä. Riippuu kehittäjästä, nauttiiko hän kaiken koodaamisesta vai haluaako hän nopeamman ja vaivattomamman vaihtoehdon pelinsä kehittämiseen. Oma lopputulos on se, että pelin prototyyppi olisi pitänyt tehdä Unity-pelimoottorilla, sillä olisin säästänyt siinä paljon aikaa ja vaivaa.

Tarkoituksena oli insinööriöraportissa myös esitellä Libgdx-sovelluskehys ja antaa kuvaa, minkälaista on kehittää peli tällä sovelluskehiksellä. Matemaattisen osuuden näyttämiseen ja selittämiseen panostettiin paljon, sillä pelkästä koodista on vaikea havaita, mitä koodissa tapahtuu.

Enemmän ja tarkemmin olisin voinut selittää muiden luokkien toteutusta ja näyttää lukijalle enemmän koodinpätkiä pelistä tarkasti selitettynä. Yksi tavoitteista oli saada kirjoitettua koko prototyypin kehitys ja toteutus vaihe vaiheelta, jotta lukija saisi käsityksen, mitä vaiheita prototyypin tekemisessä käytiin läpi. Mielestäni tässä olisi ollut parannettavaa, sillä koodipuolta näytettiin paljon tavoitettua vähemmän ja siitä puuttui vaihe vaiheelta selittäminen. Lukija ei välttämättä näiden koodipätkien perusteella näe kokonaisuuden tarkoitusta ja sitä, mitä erilaisia hyötyjä Libgdx-sovelluskehys toi mukanaan tähän projektiin.

Eri pelimoottorien ja sovelluskehysten esitleminen ja vertaileminen raportissa antaa kuvan eri vaihtoehdoista ja eroavaisuuksista. Tarkoituksena oli siis tuoda esille hintaerot ja suurimmat eroavaisuudet, jotta saa selvän kuvan, mitä hyötyjä ja haittoja kukin pelimoottori tai sovelluskehys tuo mukanaan. Tämän avulla lukija, joka haluaa kehittää omaa peliään, voi saada selvän käsityksen, mikä pelimoottori tai sovelluskehys hänelle sopisi parhaiten.

Lähteet

1. Features. 2016. Verkkodokumentti. Libgdx. <https://libgdx.badlogicgames.com/features.html> Luettu: 2.3.2016
2. Zechner, Mario. Libgdx 1.0 released. 2014. Verkkodokumentti. <http://www.badlogicgames.com/wordpress/?p=3412> Luettu: 2.3.2016
3. Libgdx / libgdx. 2016. Verkkodokumentti. Github. <https://github.com/libgdx/libgdx> Luettu: 10.3.2016
4. The life cycle. 2016. Verkkodokumentti. Github Libgdx wiki. <https://github.com/libgdx/libgdx/wiki/The-life-cycle> Luettu: 10.3.2016
5. Managing your assets. 2016. Verkkodokumentti Github Libgdx wiki. <https://github.com/libgdx/libgdx/wiki/Managing-your-assets> Luettu: 10.3.2016
6. Libgdx tutorial 9: Scene2d part 1. 2013. Verkkodokumentti. GamesFromScratch.com. <http://www.gamefromscratch.com/post/2013/11/27/LibGDX-Tutorial-9-Scene2D-Part-1.aspx> Luettu: 7.3.2016
7. Main page. 2016. Verkkodokumentti. Unity. <https://unity3d.com/unity> Luettu: 10.3.2016
8. Get Unity. 2016. Verkkodokumentti. Unity. <https://unity3d.com/get-unity> Luettu: 10.3.2016
9. Individuals, Indies and Studios. 2016. Verkkodokumentti. YoYo Games. <https://www.yoyogames.com/get> Luettu; 10.3.2016
10. Super Hexagon. 2016. Verkkodokumentti. Wikipedia. https://en.wikipedia.org/wiki/Super_Hexagon Luettu: 3.3.2016
11. Getting started. 2016. Verkkodokumentti. Git. <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control> Luettu: 10.3.2013

12. Features. 2016. Verkkodokumentti. Bitbucket. <https://www.atlassian.com/software/bitbucket/features> Luettu: 10.3.2016

13. Licenses. 2016. Verkkodokumentti. Android <https://source.android.com/source/licenses.html> Luettu: 3.3.2016

14. Setting up your Development Environment (Eclipse, IntelliJ IDEA, NetBeans). 2016. Verkkodokumentti. Github Libgdx Wiki. <https://github.com/libgdx/libgdx/wiki/Setting-up-your-Development-Environment-%28Eclipse,-IntelliJ-IDEA,-NetBeans%29> Luettu: 10.3.2016

15. Number of available applications in the Google Play Store. 2015. Verkkodokumentti. Statista. <http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/> Luettu: 3.3.2016

16. Google Play Developer API. 2015. Verkkodokumentti. Google Developers. <https://developers.google.com/android-publisher/#subscriptions> Luettu: 10.3.2016

17. The Google Play Opportunity. 2016. Verkkodokumentti. Android. <http://developer.android.com/distribute/googleplay/about.html> Luettu: 10.3.2016

18. Aleem, Mohammad Rafay. Libgdx vs Android SDK for Game Development – Which approach is better?. 2013. Verkkodokumentti. <https://www.quora.com/Libgdx-vs-Android-SDK-for-Game-Development-Which-approach-is-better> Luettu 10.3.2106

19. UnityScript versus JavaScript. 2014. Verkkodokumentti. Unity 3D wiki. http://wiki.unity3d.com/index.php?title=UnityScript_versus_JavaScript Luettu: 10.3.2016

20. AndroidApplicationConfiguration.java. 2016. Verkkodokumentti. Github Libgdx. <https://github.com/libgdx/libgdx/blob/master/backends/gdx-backend-android/src/com/badlogic/gdx/backends/android/AndroidApplicationConfiguration.java> Luettu: 10.3.2016

21. LwjglApplicationConfiguration.java. 2015. Verkkodokumentti. Github Libgdx. <https://github.com/libgdx/libgdx/blob/master/backends/gdx-backend->

lwjgl/src/com/badlogic/gdx/backends/lwjgl/LwjglApplicationConfiguration.java Lu-
ettu: 10.3.20

22. Which Companies in Delhi NCR region work on game development using LibGdx?.
2014. Verkkodokumentti. Quora. <https://www.quora.com/Which-companies-in-Delhi-NCR-region-work-on-game-development-using-LibGdx> Luettu: 7.4.2016
23. Company facts. 2016. Verkkodokumentti. Unity. <https://unity3d.com/public-relations>
Luettu: 7.4.2016

GameScreen.java

GameScreen-luokan toteutus, jossa on toteutettu pelin pääsilmut. Tässä luokassa käsitellään myös pelaajan kosketukset ja eri tapahtumien toiminta.

```
package com.kami.redpill.screens;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.Pixmap;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.g2d.freetype.FreeTypeFontGenerator;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.math.Intersector;
import com.badlogic.gdx.math.MathUtils;
import com.badlogic.gdx.math.Vector3;
import com.kami.redpill.entities.CenterObject;
import com.kami.redpill.entities.Pill;
import com.kami.redpill.entities.PlayerLine;
import com.kami.redpill.entities.Section;
import com.kami.redpill.entities.SectionLine;
import com.kami.redpill.tools.Assets;
import com.kami.redpill.tools.PillManager;
import com.kami.redpill.tween.Effects;
import com.kami.redpill.ui.GameUI;

/**
 * Base class for Game screen
 * @author Kami Nasri.
 */
public final class GameScreen implements Screen {
    public static int CAMERA_WIDTH = 1280;
    public static int CAMERA_HEIGHT = 720;

    public static int GAME_AREA = 3000; // In pixels

    // Tools
    private SpriteBatch batch;
    private ShapeRenderer debugBatch;
    private OrthographicCamera camera;

    // Timer
    private float delaySpawnMax = 10f;
    private final float DELAY_SPAWN_MIN = 1f;
    private float delayTimer;

    private final float DELAY_NEXTLEVEL = 20f;
    private float gameTimer;

    // Freetype font support
    private FreeTypeFontGenerator fontGenerator;
```

```

private FreeTypeFontGenerator.FreeTypeFontParameter fontParameter;
private BitmapFont font;
private int currentLevel;

// Objects
private PillManager pillManager;
private SectionLine[] sectionLines;
private Sprite gameOverArea;
private Section[] sections;
private CenterObject centerObject;
private PlayerLine playerLine;

// Input
private Vector3 inputPos;

private final Game game;
private GameUI ui;
private boolean isGameStarted;

public GameScreen(final Game game){
    this.game = game;
}

@Override
public void show() {
    // Init objects
    sectionLines = new SectionLine[2];

    sectionLines[0] = new SectionLine();
    sectionLines[0].initCornerLine(false);

    sectionLines[1] = new SectionLine();
    sectionLines[1].initCornerLine(true);

    // Initialise sections
    sections = new Section[8];
    // Active sections
    sections[0] = new Section(Section.SECTION_TOP, new
Color(Color.valueOf("318CE7FF")));
    sections[1] = new Section(Section.SECTION_RIGHT, new
Color(Color.valueOf("318CE7FF")));
    sections[2] = new Section(Section.SECTION_BOTTOM, new
Color(Color.valueOf("318CE7FF")));
    sections[3] = new Section(Section.SECTION_LEFT, new
Color(Color.valueOf("318CE7FF")));

    // Inactive sections
    sections[4] = new Section(Section.SECTION_TOP, Sec-
tion.COLOR_INACTIVE);
    sections[5] = new Section(Section.SECTION_RIGHT, Sec-
tion.COLOR_INACTIVE);
    sections[6] = new Section(Section.SECTION_BOTTOM, Sec-
tion.COLOR_INACTIVE);
    sections[7] = new Section(Section.SECTION_LEFT, Sec-
tion.COLOR_INACTIVE);

    for (Section s : sections) {
        s.setActive(true); // Don't draw inactive sections. Bool-
ean doesn't affect default sections
    }
}

```

```

        // Generate font
        fontGenerator = new FreeTypeFontGenerator(Gdx.files.inter-
nal("fonts/ingame.ttf"));
        fontParameter = new FreeTypeFontGenerator.FreeTypeFontParame-
ter();
        fontParameter.size = 21;
        font = fontGenerator.generateFont(fontParameter);
        fontGenerator.dispose();

        // Create filled rectangle with pixmap
        Pixmap pixmap = new Pixmap(100, 100, Pixmap.Format.RGBA8888);
        pixmap.setColor(Color.WHITE);
        pixmap.fill();
        centerObject = new CenterObject(new Texture(pixmap));
        centerObject.setColor(new Color(Color.CYAN.r - 0.2f,
Color.CYAN.g - 0.2f, Color.CYAN.b - 0.2f, 1f));
        pixmap.dispose(); // Pixmap is not needed after creating the
sprite

        // Game over area
        Pixmap gameOverPixmap = new Pixmap(720, 720, Pixmap.For-
mat.RGB888);
        gameOverPixmap.setColor(Color.WHITE);
        gameOverPixmap.fill();
        gameOverArea = new Sprite(new Texture(gameOverPixmap));
        gameOverPixmap.dispose();

        playerLine = new PlayerLine();
        pillManager = new PillManager();

        // Init tools
        batch = new SpriteBatch();
        debugBatch = new ShapeRenderer();

        camera = new OrthographicCamera();
        camera.setToOrtho(false, CAMERA_WIDTH, CAMERA_HEIGHT);
        camera.position.set(GAME_AREA / 2, GAME_AREA / 2, 0);

        Effects.init();
        inputPos = new Vector3();

        // Init timer
        delayTimer = 0f;
        gameTimer = 0;
        currentLevel = 0;
        isGameStarted = false;

        gameOverArea.setPosition(camera.position.x - CAMERA_HEIGHT /
2, camera.position.y - CAMERA_HEIGHT / 2);

        ui = new GameUI();

        // Game start effects
        centerObject.scale(7f);
        Effects.rotate(centerObject, 360, 3f); // Rotate center object
360 degrees and star the game
        Effects.shrink(centerObject, 1, 3f);
        Effects.setColor(centerObject, 0f, 0f, 0f, 3f);
        for(int i = 0; i < 4; i++){

```

```

        Effects.setColor(sections[i].getSprite(), Section.COLOR_DEFAULT.r, Section.COLOR_DEFAULT.g, Section.COLOR_DEFAULT.b, 3f);
    }
}

@Override
public void render(float delta) {
    //For android anti aliasing
    Gdx.gl.glClearColor(0f, 0f, 0f, 1f);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT | GL20.GL_DEPTH_BUFFER_BIT | (Gdx.graphics.getBufferFormat().coverageSampling ? GL20.GL_COVERAGE_BUFFER_BIT_NV : 0));
    camera.update();

    batch.setProjectionMatrix(camera.combined);
    batch.begin();

    if(centerObject.getRotation() == 360){
        isGameStarted = true; // Start the game
    }

    // Draw objects //
    // Timed spawn
    delayTimer += delta;
    if (delayTimer >= MathUtils.random(DELAY_SPAWN_MIN, delaySpawnMax)) {
        pillManager.spawn(sections[MathUtils.random(3)]);
        delayTimer = 0f;
    }

    // After 20 seconds make the game harder (Level up)
    gameTimer += delta;
    if(gameTimer >= DELAY_NEXTLEVEL){
        if(delaySpawnMax > DELAY_SPAWN_MIN) {
            delaySpawnMax -= 0.5f;
        }
        gameTimer = 0f;
        currentLevel++;
    }

    // Draw sections
    for (Section section : sections){
        if(!section.getSectionColor().equals(Section.COLOR_INACTIVE)) {
            section.draw(batch, delta);
        }
    }

    gameOverArea.draw(batch);

    // Draw yellow sections in front of the gameOverArea
    for (Section section : sections){
        if(section.getSectionColor().equals(Section.COLOR_INACTIVE)) {
            section.draw(batch, delta);
        }
    }

    for(SectionLine s : sectionLines){

```

```

        s.draw(batch);
    }

    centerObject.draw(batch);

    // Update game logic //
    //Update input
    handleInput();

    //Check collision between pills and center object
    checkCollision();

    //Update managers
    Effects.updateSpriteManager(delta);
    if(isGameStarted) {
        pillManager.update(batch);
        updateGameState();
    }

    font.setColor(Color.WHITE);
    font.draw(batch, "Level: " + (currentLevel + 1), 20, CAM-
ERA_HEIGHT - 20);

    ui.updateTable(centerObject);
    ui.setTime((int)delta); // Total game time
    ui.draw();
    batch.end();

    // DEBUGGIN // TODO:REMOVE THIS WHEN YOU DONT NEED DEBUGGING
    /*debugBatch.setAutoShapeType(true);
    debugBatch.begin();
    debugBatch.setColor(Color.RED);
    debugBatch.polygon(playerLine.getHitbox().getVertices());
    debugBatch.end();*/
}

/**
 * Handle player input
 */
private void handleInput(){
    inputPos.set(Gdx.input.getX(), Gdx.input.getY(), 0);
    camera.unproject(inputPos);
    if(Gdx.input.isTouched()){
        // If the section is inactive then don't draw the line in
the section
        for(Section s : sections){
            if(s.getSectionPolygon().contains(inputPos.x, input-
Pos.y) && !s.isActive()){
                playerLine.resetHitBox();
                return;
            }
        }

        // If player touches the center box then don't draw any-
thing
        if(!centerObject.getBoundingBox().contains(input-
Pos.x, inputPos.y)){
            for(Section s : sections){
                if(s.getSectionPolygon().contains(inputPos.x, in-
putPos.y)) {

```

```

        playerLine.draw(batch, inputPos, s.getSection-
Name());
    }
    } else {
        playerLine.resetHitBox();
    }
    } else {
        playerLine.resetHitBox();
    }
}

/**
 * Check collision between {@link com.kami.redpill.entities.Pill
Pill},
 * {@link com.kami.redpill.entities.CenterObject CenterObject} and
 * {@link com.kami.redpill.entities.PlayerLine}
 */
private void checkCollision(){
    for(Pill pill : pillManager.getPills()){
        // Collision with the center object
        if(Intersector.overlapConvexPolygons(centerObject.getHit-
Box(), pill.getHitbox())){
            pillManager.getRemovablePills().add(pill);

            // Scale the center object if the colliding object is
red pill
            if(pill.getPillColor().equals(Assets.INGAME_REDPILL))
            {
                ui.setCombo(0);
                Effects.grow(centerObject, 20, 0.3f);
            } else {
                ui.setCombo(ui.getCombo() + 1);
                ui.setScore(ui.getScore() + (ui.getCombo()*2));
                Effects.shrink(centerObject, 10f, 0.3f);
            }
        }

        // Collision with the player line
        if(Intersector.overlapConvexPolygons(pill.getHitbox(),
playerLine.getHitbox())){
            pillManager.getRemovablePills().add(pill);

            if(pill.getPillColor().equals(Assets.INGAME_YEL-
LOWPILL)){
                // Check in which section the pill was and inacti-
vate the section
                for(Section s : sections){
                    if(Intersector.overlapConvexPolygons(s.getSec-
tionPolygon(), pill.getHitbox())){
                        ui.setCombo(0);
                        s.setActive(false);
                    }
                }
            } else {
                ui.setCombo(ui.getCombo() + 1);
                ui.setScore((ui.getScore()+1) + ui.getCombo());
            }
        }
    }
}

```

```

        // Remove all yellow pills from inactive sections
        for(Section s : sections){
            if(!s.isActive() && pill.getPillColor().equals(Assets.INGAME_YELLOWPILL)){
                if(Intersector.overlapConvexPolygons(s.getSectionPolygon(), pill.getHitbox())) {
                    pillManager.getRemovablePills().add(pill);
                }
            }
        }
    }

    private void updateGameState(){
        if(centerObject.getBoundingBox().x < 0 || centerObject.getBoundingBox().getWidth() > CAMERA_WIDTH
            || centerObject.getBoundingBox().y < 0 || centerObject.getBoundingBox().getHeight() > CAMERA_HEIGHT){
            game.setScreen(new GameOverScreen(game, ui.getScore()));
        }
    }

    @Override
    public void resize(int width, int height) {
    }

    @Override
    public void pause() {
    }

    @Override
    public void resume() {
    }

    @Override
    public void hide() {
    }

    @Override
    public void dispose() {
        batch.dispose();
    }
}

```


PillManager.java

Toteutus PillManager luokasta, joka hoitaa kaikkien pillereiden luomisen peliruudulle. Luokan toimintoja kutsutaan GameScreen luokassa.

```
package com.kami.redpill.tools;

import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.math.MathUtils;
import com.badlogic.gdx.math.Vector2;
import com.kami.redpill.entities.Pill;
import com.kami.redpill.entities.Section;
import com.kami.redpill.screens.GameScreen;

import java.util.ArrayList;

/**
 * Handles everything related to {@link com.kami.redpill.entities.Pill
 * Pill}
 * @author Kami Nasri.
 */
public final class PillManager {
    private ArrayList<Pill> pills;
    private ArrayList<Pill> removablePills;

    private Vector2 pillDirection;

    private int redPillSpawnPercentage;
    private final int DEFAULT_REDPILL_PERCENTAGE = 90;
    private final float PENALTY_SPAWN_PERCENTAGE = 30;

    private int spawnPaddingX = 130;
    private int spawnPaddingY = 120;

    /**
     * Create new instance of {@link com.kami.redpill.tools.Pill-
     * Manager PillManager}
     */
    public PillManager(){
        pills = new ArrayList<Pill>();
        removablePills = new ArrayList<Pill>();
        pillDirection = new Vector2();
        redPillSpawnPercentage = 30;
    }

    /**
     * Update positions and the direction of pills. Also handles col-
     * lision detection.
     * @param batch - SpriteBatch where pills are drawn.
     */
    public void update(SpriteBatch batch){
        // Update pills
        for(Pill p : pills){
            p.draw(batch);
        }

        // Remove unnecessary pills
    }
}
```

```

        for(Pill p : removablePills){
            pills.remove(p);
        }
    }

    /**
     * Spawn pills randomly from all four sections
     */
    public void spawn(Section s){
        // When section is inactive spawn only red pills at 30% change
        // of default red pill spawn%
        if(!s.isActive()){
            redPillSpawnPercentage = 100;

            float spawnPercentage = DEFAULT_REDPILL_PERCENTAGE - (DE-
            FAULT_REDPILL_PERCENTAGE * (PENALTY_SPAWN_PERCENTAGE /100f));
            System.out.println(spawnPercentage);
            boolean spawnPill = (MathUtils.random(100)) < spawnPer-
            centage;

            if(!spawnPill){
                return;
            }
        } else {
            redPillSpawnPercentage = DEFAULT_REDPILL_PERCENTAGE;
        }

        Pill p = new Pill((MathUtils.random(100)) < redPillSpawnPer-
        centage); // redPillSpawn% chance to create a red pill

        if(s.getSectionName().equals(Section.SECTION_TOP)) {
            p.setPosition(MathUtils.random(p.getWidth() + spawnPad-
            dingX, GameScreen.GAME_AREA - p.getWidth() - spawnPaddingX),
            GameScreen.GAME_AREA);
            p.setDirection(Section.TOP_CENTER[0], Section.TOP_CEN-
            TER[1]);
        } else if(s.getSectionName().equals(Section.SECTION_RIGHT)){
            p.setPosition(GameScreen.GAME_AREA, MathUtils.ran-
            dom(p.getHeight() + spawnPaddingY, GameScreen.GAME_AREA -
            p.getHeight() - spawnPaddingY));
            p.setDirection(Section.RIGHT_CENTER[0], Section.RIGHT_CEN-
            TER[1]);
        } else if(s.getSectionName().equals(Section.SECTION_BOTTOM)){
            p.setPosition(MathUtils.random(p.getWidth() + spawnPad-
            dingX, GameScreen.GAME_AREA - p.getWidth() - spawnPaddingX), 0f);
            p.setDirection(Section.BOTTOM_CENTER[0], Section.BOT-
            TOM_CENTER[1]);
        } else if(s.getSectionName().equals(Section.SECTION_LEFT)){
            p.setPosition(0f, MathUtils.random(p.getHeight() + spawn-
            PaddingY, GameScreen.GAME_AREA - p.getHeight() - spawnPaddingY));
            p.setDirection(Section.LEFT_CENTER[0], Section.LEFT_CEN-
            TER[1]);
        }

        p.setMovementSpeed(MathUtils.random(Pill.MOVEMENT_SPEED_MIN,
        Pill.MOVEMENT_SPEED_MAX));
        pills.add(p);
    }

    /**

```

```
    * Return all pills that are in the game
    * @return - List of pills
    */
    public ArrayList<Pill> getPills(){
        return pills;
    }

    /**
    * Return all pills that are queued to be removed
    * @return - List of removable pills
    */
    public ArrayList<Pill> getRemovablePills(){
        return removablePills;
    }
}
```