Android-pelin kehitys ja julkaisu

Oskari Nurmi

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2015

Tiivistelmä

Päiväys
18.11.2015

Tekijä(t)
Oskari Nurmi

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Android-pelin kehitys ja julkaisu

Sivu- ja liitesivumäärä
20+0

Opinnäytetyön otsikko englanniksi
Development and publishing of an Android game

Tämän opinnäytetyön tavoitteena on kehittää toimiva peli Android-alustalle ja julkaista se Google Play-sovelluskaupassa. Tämä kehitysprosessin kulku kuvataan pelin rakenteen näkökulmasta.

Kuvaus aloitetaan projektissa käytetyistä työkaluista kuten ohjelmointikielestä ja kehitysympäristöstä, ja jatketaan pelin suunnitteluvaiheiden kerrontaan. Peli suunniteltiin mahdollisimman yksinkertaisesti toimivaksi jotta opinnäytetyö valmistuisi ajallaan. Suunnittelun jälkeen kuvataan pelin ohjelmoinnin eteneminen yksi Java-luokka kerrallaan. Tavoitteena on että ohjelmointia ymmärtävä pystyy seuraamaan projektin rakennetta vaikka koodiin ei yksityiskohtaisesti paneuduta. Ohjelmointiosion lopussa kerrotaan lyhyesti pelin grafiikoiden teosta ja audion valinnasta.

Tämän jälkeen kerrotaan kuinka luodaan Google kehittäjätili ja edelleen kuinka sovellus julkaistaan Googlen Play-kaupassa. Lopussa tarkastellaan ja pohditaan projektin tuloksia, onnistumista ja omaa oppimista.

Asiasanat
Android, mobiilipeli, pelinkehitys, julkaisu

Abstract

Date
18th November, 2015

Author(s)
Oskari Nurmi

Degree programme
Business Information Technology.

Report/thesis title
Write the main title of your report/thesis here**.**

Number of pages and appendix pages
20+0

The goal of this thesis work is to develop a working game for the Android platform and publish it in the Google Play store. This development process is described from the perspective of the game's structure.

The description begins from the tools used in the project, like the programming language and development environment, and continues to the game's design. The game was designed to be as simple as possible so the thesis would finish in time. After telling about the game design, the game's programming is described one Java class at a time. The aim is that a person that understands programming can follow the project's structure even when the code is not described in detail. At the end of the programming part, there's a short description of creating the game's graphics and selecting the audio.

After this, the creation of a Google developer account is told and then the publishing of an application in the Google Play store. Finally, the results of the project, it's success and own learnings are viewed and pondered upon.

Keywords
Android, mobile game, game development, publishing

Sisällys

1 Johdanto

Tämän opinnäytetyön tarkoituksena oli kehittää yksinkertainen peli Android-alustalle käyttäen Android Studio kehitysympäristöä, ja julkaista se myyntiin Google Play Storeen. Tämä kehitysprosessi kuvataan tavalla joka on parhaiten hyödyksi niille, jotka ymmärtävät Java-ohjelmointia mutta eivät välttämättä ole tehneet pelejä ennen. Kuvauksen keskiössä on sovelluksen rakenne ja toiminta.

Opinnäytetyön aiheen valintaan vaikutti kiinnostukseni pelinkehitykseen, ja koska Java on minulle kielenä tuttu, muodostui Android hyväksi valinnaksi projektin alustaksi. Androidille on myös melkoisen helppo julkaista pelejä myyntiin verrattuna pc-alustaan, joten tämäkin oli plussa Androidin valinnassa. Tämän projektin tuotos julkaistaan ilmaissovelluksena, mutta ohjeistusta voi hyvin soveltaa myös maksulliseen julkaisuun.

Työssä käydään ensin läpi projektin tietopohja, eli sovelluksen työstössä käytetyt työkalut. Tämän jälkeen kuvataan pelin suunnittelu ideasta koodausvalmiiksi suunnitelmaksi. Suunnittelu on hyvin tärkeä vaihe pelin kehitystä, ja tässä osiossa kerrotaan kuinka tämän projektin pelin idea muotoutui.

Suunnittelun jälkeen käydään pelin ohjelmointi läpi luokka kerrallaan loogisessa järjestyksessä. Tämä tehdään menemättä liiallisiin yksityiskohtiin itse koodissa, mutta siten että pelin rakennetta pystyy seuraamaan ja ymmärtämään.

Seuraavassa osassa kuvataan sovelluksen julkaisun työvaiheet Google Play -sovelluskauppaan. Lopussa käydään läpi projektin tulokset ja pohditaan projektin onnistumista ja tuloksia.

2 Tietoperusta

Käsitellään aluksi erinäiset työkalut millä tätä projektia lähdettiin työstämään. Tämä käsittää muun muassa käytetyn ohjelmointikielen kuin julkaisu/myyntipalvelunkin.

2.1 Java

Java on alun perin Sun Microsystemsin kehittämä, nykyisin Oraclen hallinnoima oliopohjainen ohjelmointikieli. Javassa on käytössä ns. vahva tyypitys, eli jokaisella muuttujalla on jokin tyyppi, ja se voi sisältää vain tässä muodossa olevaa informaatiota.

Java pyörii normaalisti tietokoneella Java-virtuaalikoneessa (JVM), minkä ansiosta Java on alustariippumaton. Androidiin kehitettiin kuitenkin Javan pyörittämistä varten DVM, koska JVM oli etenkin Androidin alkuaikojen puhelimille liian raskas pyöritettäväksi. (What is Java Technology and why do I need it? 2015.)

2.2 Android

Android on Googlen kehittämä käyttöjärjestelmä mobiilikäyttöön. Ensimmäinen Androidia käyttävä laite julkaistiin vuonna 2008. Androidia kehitetään hyvin nopeaan tahtiin ja uusia versioita ilmestyy noin puolen vuoden välein. Nykyään Android on maailman suosituin mobiilikäyttöjärjestelmä: sitä käytetään sadoissa miljoonissa mobiililaitteissa ympäri maailman.

Android perustuu Linuxin kerneliin, ja sitä julkaistaan avoimen lähdekoodin lisenssin alaisena. Suurin osa Android-sovelluksista on kirjoitettu Javalla, mikä pyörii Androidissa Dalvik-virtuaalikoneella (DVM). Android-sovelluskehitystä varten Google on julkaissut Android Studion, mikä korvaa aiemmin käytetyn Eclipsen Android-kehitystyökalut. (About Android 2015)

2.3 Android Studio

Android Studio on virallinen integroitu kehitysympäristö (integrated development environment, IDE) Android-sovelluskehitykseen. Android Studion ensimmäinen vakaa versio julkaistiin loppuvuodesta 2014. Se on ilmaiseksi ladattavissa Windows, Mac OS X ja Linux -alustoille. (Android Developer 2015.a)

Android Studiossa tulee mukana optimaalisesti esimääritelty emulaattori. Käytettävissä on laiteprofiilit yleisimmistä Android- laitteista. (Android Developer 2015.b)

2.4 Google Play

Google Play, edelliseltä nimeltään Android Market, on Googlen kauppa Android -laitteille. Playsta voi ostaa sovellusten lisäksi elokuvia, musiikkia ja kirjoja. Google play tallentaa kaiken ostamasi pilveen, ja niitä voi käyttää sieltä jokaisen Android -laitteen kautta. (What is Google Play? 2015)

2.5 Inkscape

Koska Android Studio ei sisällä grafiikkaeditoria, tehdään tämän projektin graafiset elementit Inkscapessa. Inkscape on laadultaan ammattilaistason vektorigrafiikkaohjelma, mikä toimii Windows, Mac OS X ja Linux -alustoilla. Inkscape on ilmainen, avoimen lähdekoodin ohjelma, ja käyttää avoimen standardin SVG:tä (scalable vector graphics) natiiviformaattinaan. Inkscapen piirtotyökalut ovat verrattavissa esimerkiksi Adobe Illustratoriin ja CorelDRAW:iin. Inkscape ymmärtää SVG:n lisäksi myös muun muassa AI, EPS, PDF, PS ja PNG -tiedostomuotoja. Inkscapella voi tehdä grafiikoita laidasta laitaan, kuten kuvituksia, ikoneita, diagrammeja, karttoja, sekä web-grafiikoita. (About Inkscape 2015)

3 Pelin suunnittelu

On mahdotonta lähteä kehittämään peliä, jos ei ole minkäänlaista ideaa siitä minkälaisen pelin aikoo tehdä. Tästä syystä ensimmäinen vaihe uuden pelin kehittämisessä on suunnittelu.

3.1 Peli-idea

Ennen kuin peliä voi lähteä varsinaisesti edes suunnittelemaan, täytyy olla peli-idea. Tämä tarkoittaa jonkinlaista ajatusta siitä, miten peli tulee toimimaan. Kun on olemassa ajatus siitä mikä pelissä on tavoitteena ja miten pelaaja voi tuohon tavoitteeseen päästä, voi peli-ideaa lähteä suunnittelemaan tarkemmin.

Tässä projektissa lähdettiin rakentamaan hyvin yksinkertaista peliä, jossa perusideana on että pelaajan tulee väistellä esteitä. Tämän pelaaja voisi tehdä hyppäämällä, eikä muita kontrolleja ole. Pelistä tulisi sivusta kuvattu ns. sidescroller, missä pelihahmo liikkuu itsestään eteenpäin maastossa ilman pelaajan vaikutusta.

Perusidean ollessa selvillä väritetään sitä hieman, ilman että se muuttuu. Pelaaja voisi siis vaikuttaa pelihahmoon yhdellä ainoalla kontrollilla; hyppäämällä, mikä käytännössä tapahtuisi napauttamalla mobiililaitteen näyttöä. Tällaisenaan peli olisi kuitenkin liian yksinkertainen, eikä sitä olisi kovin mielekästä pelata. Muokataan siis ideaa hieman: maaston sijasta pelaaja liikkuu eteenpäin tunnelissa, ja tavallisen loikan sijasta hyppääminen siirtää pelihahmon liikkumaan katossa, ja katosta hypätessä takaisin lattiaan. Näin peliin tulee jo huomattavasti enemmän ulottuvuutta, kun hahmo voi liikkua useammalla pinnalla.

Seuraavaksi mietitään pelin haastetta, eli esteiden väistelyä. Käytännössä pelin maastona toimiva tunneli tulee koostumaan samankokoisista laatikoista eli tiilistä, jolloin esteet mitä pelaajan tulee varoa, ovat yksinkertaisesti maaston korkeusvaihteluita. Jos pelihahmo törmää vastaan tulevaan tiileen tai putoaa tiileltä alas ilman hyppyä, hahmo ”kuolee” ja pelaaja joutuu aloittamaan alusta. Pelaajan tulee siis hypätä aina kun vastaan tulee korkeusvaihteluita, muutoin tämä häviää.

Viimeisenä mietinnän kohteena on pelin loppuminen; onko pelillä loppua ja jos on niin millainen. Käytännössä vaihtoehtoina on tehdä pelistä tasopohjainen, jolloin haasteena on läpäistä yksi tunneli, jolloin pääsee seuraavaan tasoon. Tässä tapauksessa tasot olisivat ennalta tehtyjä ja niitä olisi tietty määrä, ja kun kaikki olisi läpäisty, niin peli päättyisi. Toi-

nen vaihtoehto olisi tehdä pistepohjainen peli: pelaaja saa pisteitä siitä mitä pidemmälle tunnelissa pääsee ja tunneli itse olisi päättymätön. Käytännön toteutuksessa tämä vaatisi että peli generoisi tunnelia sitä mukaan kun pelaaja etenee, eikä ennalta rakennettuja tasoja ole. Jotta pelistä ei loppuisi pelaaminen kesken, tehdään tämä peli käyttäen satunnaisesti arvottua tunnelin etenemistä.

3.2 Luokkarakenne

Kun pelin toiminta on selvillä ajatuksellisesti, on hyvä suunnitella miten sen koodi tulee rakentumaan. Luokkarakenteen ollessa selvillä voi ohjelmoinnin aloittaa.

Ensimmäiselle Java-luokalle annetaan nimeksi Main. Sen tehtäväksi tulee oikeastaan vain käynnistää sovellus ja siirtää näkymä seuraavalle luokalle, minkä nimeksi annetaan Core. Coressa tulee sijaitsemaan sovelluksen update ja draw -metodit, minkä tehtävä on päivittää ja piirtää pelitapahtumat. Core tulee myös pyörittämään pelilooppia, mikä laitetaan omaan luokkaansa nimeltä GameLoop. Core ja GameLoop siis muodostaa parin mikä pyörittää koko peliä.

Seuraavaksi peli tarvitsee taustan mikä liikkuu itsestään: tehdään siitä uusi luokka mitä kutsutaan Coresta. Tämän luokan nimeksi tulee Background. Myös itse tunnelin seinien tulee liikkua taustan kanssa samaa tahtia. Luokan nimeksi tulee Level ja sitä kutsutaan Coresta. Myös pelihahmo tarvitsee oman luokkansa, annetaan sille nimeksi Player, ja kutsutaan sitä Coresta. Nämä luokat periaatteessa riittävät siihen että peli toimii. Kuitenkin koska peliin ei tule valmiiksi tehtyjä tasoja, täytyy edettävä taso luoda jostakin. Tätä varten tehdään luokka nimeltä LevelGenerator, ja sitä kutsutaan Levelistä.

4 Pelin kehitys

Suunnittelun ollessa mallillaan voidaan siirtyä kohti pelin ohjelmointia. Tässä projektissa käytetään kehitysympäristönä Androidin virallista kehitysympäristöä, Android Studiota. Ohjelmointi aloitetaan Android Studiossa luomalla uusi projekti (kuva 1). Säilytetään oletusarvot suurimmassa osassa valinnoista mitä ohjelma tarjoaa, ja laitetaan ensimmäisen Java-luokan nimeksi Main.

Kuva 1. Uuden Android-projektin Android-version valintaikkuna.

4.1 Main ja Core

Main -luokka käsittelee sovelluksen käynnistämisen tapahtumia, ja on sellaisenaan melkein käyttökelpoinen. Ainut asia mikä pitää lisätä Mainin Create -metodiin on näkymän asettaminen uuteen luokkaan nimeltä Core, ja Main -luokan itsensä antaminen Corelle parametriksi.

Seuraavaksi luodaan uusi Java-luokka nimeltä Core. Siitä tulee tavallaan pelin ydin koska se tulee pitämään sisällään pää- update ja draw -metodit (kuva 2). Coreen määritetään final -tyyppisinä muuttujina pelin oletuskorkeus ja -leveys, pelin nopeus sekä yhden ”tiilen” koko. Core tulee myös pyörittämään useita muita luokkia, ja nämä kaikki pitää alustaa muuttujiksi Corelle. Tärkeimpänä näistä on GameLoop -luokka, joka tulee pitämään sisällään itse peliloopin. Alustetaan Coren konstruktorissa uusi instanssi GameLoop -luokasta ja annetaan sille Core itse parametrinä.

```
public void update(){
    if(!player.isLost()){
        bg.update();
        level.update();
        player.update(level.getTunnel());
    }
}
```

Kuva 2. Coren update-metodi.

Pelin päivittämisen ja piirtämisen lisäksi Coreen tulee metodit surfaceCreated, surfaceChanged, surfaceDestroyed ja onTouchEvent. Näistä viimeisin ottaa vastaan kosketustapahtumia, ja muut hoitavat pinnan luonti-, muutos- ja tuhoamistapahtumia. SurfaceCreated on hyvä paikka alustaa muut Coren käyttämät luokat, joita ovat Background, Player, ja Level. Siellä myös käynnistetään GameLoopissa sijaitseva lanka (thread), eli käytännössä käynnistetään peli vasta siinä vaiheessa kun pinta on luotu. SurfaceChangedin voi jättää tyhjäksi ja surfaceDestroyedissä varmistetaan että lanka katkeaa kun pinta tuhotaan.

4.2 GameLoop

Vaikka Coresta jo viitattiin GameLooppiin useaan kertaan, itse luokkaa ei ole vielä olemassa, joten luodaan se seuraavaksi. Laitetaan GameLoop laajentamaan (extends) Thread, jotta sitä on helpompi käsitellä. Itse luokka tulee sisältämään kaksi metodia konstruktorin lisäksi: run -metodin missä itse pelilooppi sijaitsee, ja setterin jolla looppi käynnistetään.

Run -metodissa sijaitseva pelilooppi tulee pyörittämään käytännössä kaikkea mitä pelissä tapahtuu. Tämä looppi on tyypiltään while-looppi, jonka ehtona on ”running” boolean muuttujan true -arvo. Aluksi running on true, ja kun peli halutaan sulkea, siitä tehdään false, jolloin koodin lukija lopulta pääsee ulos peliloopista.

Loopin itsensä sisällä on useita muuttujia, joiden avulla loopin pyörimisnopeutta rajoitetaan. Rajoittamattomana se pyörisi aivan liian nopeasti jotta peliä voisi pelata. Yleensä peleissä suositaan nopeuden rajaamista joko kolmeenkymmeneen tai kuuteenkymmeneen ruutuun per sekunti (frames per second, FPS). Tässä tapauksessa valitaan 60 FPS, koska ei ole mitään syytä miksi alempaan nopeuteen tyytyminen olisi tarpeellista.

```
while(isRunning){
    startTime = System.nanoTime();
    canvas = null;

    try{
        canvas = this.surfaceHolder.lockCanvas();
        synchronized (surfaceHolder){
            this.core.update();
            this.core.draw(canvas);
        }
    }catch(Exception e){}
    finally {
        if(canvas != null){
            try{
                surfaceHolder.unlockCanvasAndPost(canvas);
            }catch (Exception e){
                e.printStackTrace();
            }
        }
    }

    timeMillis = (System.nanoTime()- startTime)/1000000;
    waitTime = targetTime - timeMillis;

    try{
        this.sleep(waitTime);
    }catch(Exception e){}

    totalTime += System.nanoTime() - startTime;
    frameCount ++;
    if(frameCount == FPS){
        averageFPS = 1000/(totalTime/frameCount/1000000);
        frameCount = 0;
        totalTime = 0;
        System.out.println(averageFPS);
    }
}
```

Kuva 3. Pelilooppi.

Kun loopin FPS on saatu toimimaan oikein, laitetaan se pyörittämään Coressa sijaitsevia update ja draw -metodeita. Updaten tarkoitus on pyörittää kaikkia pelin tapahtumia eteenpäin, ja draw piirtää grafiikat sen jälkeen näytölle (kuva 3).

4.3 Level

Palataan takaisin Coreen kun GameLoop on valmis. Corelle alustettiin muuttujana myös Level, Background ja Player -luokat, joten luodaan uusi luokka nimeltä Level seuraavaksi. Level tulee pitämään sisällään datan siitä minkä näköinen tunneli on siinä kohtaa mihin pelaaja on edennyt. Käytännössä tunnelia ei tule olemaan olemassa pelaajan näkökentän ulkopuolella, vaan sitä poistetaan/luodaan sitä mukaa kun pelaaja etenee.

Level tarvitsee konstruktorin lisäksi vain muutaman metodin: update ja draw tunnelin päivitystä ja piirtoa varten, joita kutsutaan Coresta, sekä getterin tunnelia varten. Level tarvitsee myös ArrayList tyyppisen muuttujan mikä tulee pitämään sisällään taulukon palikoita, eli yhden vertikaalisen siivun tunnelia. Tässä vaiheessa todetaan että nämä palikat tarvitsevat oman luokan, mikä pitää luoda myöhemmin. Level tarvitsee myös tavan luoda lisää uutta tunnelia pelaajan edetessä. Yksinkertaisinta on ulkoistaa tämä toiminto omaan luokkaansa, joten annetaan Levelille muuttujaksi uusi ilmentymä luokasta LevelGenerator, mikä luodaan myöhemmin.

```
public void update(){
    //move the tunnel
    for(int i = 0; i < tunnel.size(); i++){
        for(int j = 0; j < tunnel.get(i).length; j++){
            tunnel.get(i)[j].update();
        }
    }
    //check if section needs to be removed
    if(tunnel.get(0)[0].getX() + Core.TILESIZE < 0){
        tunnel.remove(0);
    }
    //check if sections need to be added
    if(tunnel.get(tunnel.size()-1)[0].getX() + Core.TILESIZE <= Core.WIDTH+5){
        if(ease){
            tunnel.add(generator.getEasedSection(tunnel.get(tunnel.size()-1)));
        }
        else{
            tunnel.add(generator.getSection(tunnel.get(tunnel.size()-1)));
        }
    }
}
```

Kuva 4. Levelin update-metodi, mikä liikuttaa tunnelia eteenpäin.

Levelin update metodissa tulee tapahtua kuvan 4 mukaisesti kolme asiaa. Koko tunnelin tulee liikkua vasemmalle, ruudun reunan yli vasemmalta menevä pala tulee poistaa, ja uusi pala tulee lisätä oikealle juuri ruudun ulkopuolelle. Käytännössä tämä tehdään niin, että aluksi täytetään ArrayList yksinkertaisilla suorilla tunnelinpaloilla jotta alku olisi helppo, ja sitten kun pelaaja liikkuu eteenpäin niin haetaan seuraavat palat LevelGenerator -luokasta, mikä luo edellisen tunnelinpätkän pohjalta seuraavan palan. Näin tunneli jatkuu loputtomiin. Level on tällaisenaan valmis, joten luodaan seuraavaksi uusi luokka.

4.3.1 Block

Annetaan uudelle luokalle nimeksi Block. Koko tunneli tulee koostumaan näistä palikoista, joten ne tarvitsevat muuttujina x- ja y-sijainnin. Blokkien korkeus ja leveys olikin jo määri-

telty Coressa muuttujana, joten käytetään sitä. Tämän lisäksi Block tarvitsee tiedon siitä onko se kiinteä vai ei, joten annetaan sille boolean -tyyppinen muuttuja kertomaan tämän. Käytännössä tämä tulee toimimaan niin, että jos palikka on kiinteä, niin pelaaja voi osua siihen, jos ei, niin sitä ei edes piirretä.

```
public class Block {
    private int x;
    private int y;
    private boolean solid;
    private Rect collision;
    private Paint paint;
    private Bitmap image;
```

Kuva 5. Block-luokan luokkamuuttujat.

Block tarvitsee myös Rect -tyyppisen muuttujan. Rect on laatikko, jonka x, y, pituus ja leveys -arvoiksi annetaan Blockin vastaavat. Blockin update -metodissa tätä laatikkoa siirretään vasemmalle simuloiden pelaajan vierimistä eteenpäin tunnelissa. Se mihin tätä laatikkoa tarvitaan, on törmäyksen tunnistaminen. Pelaajalla tulee olemaan useampi Rect-tyyppinen laatikko, ja vertaamalla ovatko nämä sisäkkäin voidaan havaita törmäys.
Juuri muuta Block ei tarvitse, siellä on metodeina update ja draw palikoiden päivitystä ja piirtoa varten.

4.3.2 Level Generator

Luodaan seuraavaksi toinen Levelin tarvitsema apuluokka, eli LevelGenerator. Tämä tulee sisältämään metodin joka ottaa vastaan tunnelin tämänhetkisen viimeisen palan, ja laskee sen pohjalta millainen seuraavan tulee olla. Tämä siksi, koska jos uusi pala vain mielivaltaisesti luotaisiin, niin voisi helposti syntyä tilanne että tunneliin tulee kohta josta on mahdotonta päästä ohi. Niinpä generaattori katsoo edellisen palan aukon korkeuden ja yläreunan y-arvon, ja arpoo niistä uudelle joko saman tai yhden - kolmen suuremman tai pienemmän y-sijainnin ja aukon koon (kuva 6).

```
public Block[] getSection(Block[] lastSection){
    int oldHoleHeight = 0;      //size of old section's passable hole
    int oldY = 0;               //y-location of old section's hole's top
    boolean foundY = false;
    for(int i = 0; i < lastSection.length; i++){
        if(!lastSection[i].isSolid()){
            if(!foundY){
                oldY = i;
                foundY = true;
            }
            oldHoleHeight ++;
        }
    }
    if(oldHoleHeight < 5){
        oldHoleHeight = 5;
    }//new hole height, cannot be smaller than 5 or bigger than 12
    int holeHeight = oldHoleHeight + random.nextInt(5)-2;
    if(holeHeight < 5 || holeHeight > 12){
        holeHeight = oldHoleHeight;
    }//new hole y-location, cannot be higher than 1 or lower than 8
    int y = oldY + random.nextInt(7)-3;
    if(y < 1 || y > 8){
        y = oldY;
    }//create new section based on given values
    Block[] section = new Block[15];
    for(int i = 0; i < section.length; i++){
        if(i == 0 || i == 14){
            section[i] = new Block(Core.WIDTH, i*Core.TILESIZE, true);
        }
        else{
            if(i >= y && i < y + holeHeight){
                section[i] = new Block(Core.WIDTH, i*Core.TILESIZE, false);
            }else{
                section[i] = new Block(Core.WIDTH, i*Core.TILESIZE, true);
            }
        }
    }
    return section;
}
```

Kuva 6. LevelGeneratorin toiminta.

4.4 Background

Seuraavaksi palataan hetkeksi Coreen ja tarkastellaan tilannetta. Core pyörittää tällä hetkellä Leveliä oikein ja peli etenee hyvin, tausta ja pelaaja vain puuttuu. Luodaan seuraavaksi Background-luokka taustaa varten, koska se on helppo hoitaa pois alta ennen pelaajaan syventymistä.

Backgroundista tulee yksinkertaisempi kuin Blockista; se pitää sisällään vain kuvan sekä x- ja y-arvon. Taustakuva ladataan Coressa ja annetaan Backgroundille kun se initialisoidaan. Update -metodiin tulee tämän taustakuvan liikkuminen vasemmalle, ja drawissa piirretään kaksi näitä taustakuvia peräkkäin niin että syntyy illuusio katkeamattomasta taus-

tasta, kunhan kuva on vain tehty siten että niitä voi saumattomasti liittää peräkkäin (kuva 7).

```
public void draw(Canvas canvas){
    canvas.drawBitmap(image, x, y, null);
    if(x < 0){
        canvas.drawBitmap(image, x + Core.WIDTH, y, null);
    }
}
```

Kuva 7. Taustan piirto Background-luokassa.

4.5 Player

Background on tällä selvä, ja voidaan luoda viimeinen pelistä puuttuva luokka, nimittäin Player. Pelaajaluokka tulee käsittelemään paitsi pelihahmon piirtämisen, myös hyppimisen ja törmäysten tunnistamisen (collision detection). Pelaajan sisältämien lukuisten toimintojen vuoksi se tarvitsee paljon muuttujia (kuva 8).

```
public class Player {
    private int x, y;
    private int width, height;
    private boolean onFloor;
    private boolean onCeiling;
    private boolean jump;
    private int direction;
    private int jumpspeed;
    private Rect collision;
    private Paint paint;
    private Bitmap[] frames;
    private int score;
    private int delay;
    private int totalDelay;
    private boolean lost;
    private int animCount;
    private int animMax;
    private int animFrame;
```

Kuva 8. Player-luokan luokkamuuttujat.

Jos pelissä olisi useampia liikkuvia hahmoja, törmäystunnistus (kuva 9) olisi hyvä sijoittaa Coreen, koska muualta on vaikea päästä käsiksi kaikkiin hahmoihin. Mutta koska tämä on niin yksinkertainen peli, on se tässä tapauksessa helpointa tehdä Player -luokassa, kunhan sitä pyörittävä update-metodi Coressa vain antaa Playerille jatkuvasti senhetkisen tunnelin listana. Playerillä on törmäyksen tarkastelua varten kolme Rect -tyyppistä muuttujaa. On ylä-, ala-, ja ydintörmäyslaatikot. Alalaatikkoa käytetään alapinnan tason tunnistamista varten, eli kun pelaaja kulkee tunnelin lattiaa pitkin, ja ylälaatikkoa puolestaan ka-

tossa liikkumiseen. Ydinlaatikko sijaitsee puolestaan pelaajan ytimessä, ja sillä katsotaan jos pelaaja törmää esteeseen niin että tämän tulee hävitä. Koska tämä laatikko sijaitsee keskellä pelaajaa, voi se käytännössä osua vain edestäpäin tulevaan esteeseen.

```
private boolean collisionCheck(Rect rect, ArrayList<Block[]> tunnel){
    for(Block[] section : tunnel){
        for(Block block : section){
            if(block.isSolid()){
                if(rect.intersect(block.getCollision())){
                    return true;
                }
            }
        }
    }
    return false;
}
```

Kuva 9. Törmäystunnistus.

Törmäyksen lisäksi pelaajan update käy läpi myös hyppäystapahtuman. Itse näytön kosketus mikä aiheuttaa hypyn käsitellään Coressa, mutta Player hoitaa pelaajahahmon liikuttamisen lattiasta kattoon ja päinvastoin. Koska tässä on kyseessä enemmänkin suoraviivainen liikkumapinnan vaihto, ei tarvita simuloida painovoimaa tai muuta vastaavaa. Pelaajaa vain liikutetaan y-akselilla ylös tai alas kunnes tämä törmää lattiaan tai kattoon.

Viimeisenä toimintonaan Player myös laskee pisteet mitä pidemmälle tunnelissa pääsee. Pisteet oli alun perin tarkoitus antaa sitä mukaan kun uusia tunnelinpätkiä syntyy, mutta käytännössä tällä ei olisi mitään eroa siihen että pisteitä vain saa aikaa myöten, joten pisteitä saa sitä mukaa kuin aikaa kuluu.

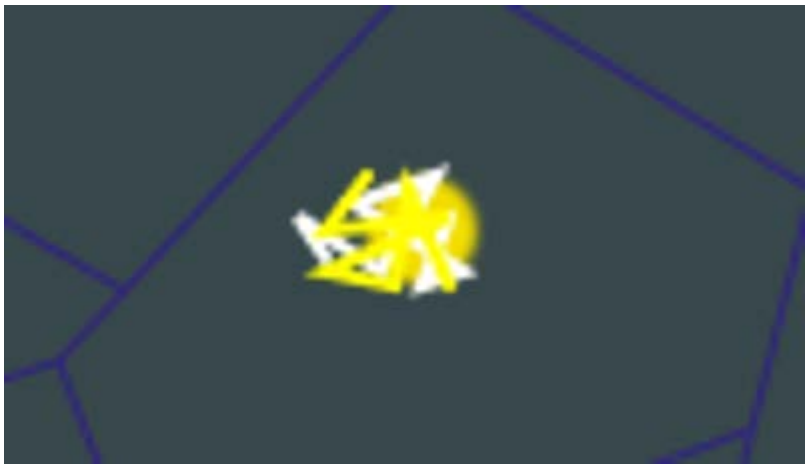
4.6 Grafiikka

Nyt peli on valmis toiminnallisesti, se vain on täysin äänetön ja näytölle piirtyy yksivärisiä neliöitä grafiikoina. Tehdään seuraavaksi pelin tarvitsemat kuvat Inkscapella, mikä on helppokäyttöinen, ilmainen vektorigrafiikkaohjelma.

Aloitetaan taustakuvasta, koska se on suurin kuva mitä peli tarvitsee. Tehdään siitä ruskean/harmaan sävyinen koska peli sijoittuu luolan sisään. Taustan on myös tarkoitus jäädä taustalle huomion kannalta, joten tehdään se tummempia sävyjä käyttäen. Tärkeintä huomioida taustakuvaa piirtäessä on että sen vasemman ja oikean reunan tulee voida saumattomasti liittyä yhteen, kuten jo Background -luokkaa koodatessa huomiotiin.

Taustan jälkeen tehdään yksittäisen palikan (Blockin) kuva. Tätä piirtäessä tulee huomioida että yksittäinen palikka on verraten pieni, mutta niitä on näkyvillä satakunta kerrallaan. Täytyy siis varoa ettei vahingossa piirrä jotain pintakuviota palikkaan mikä toistuessaan vie ärsyttävästi huomiota. Olisi kuitenkin hyvä, että palikoiden reunat erottaisivat toisistaan jotta hyppyjä on helpompi arvioida.

Viimeisenä kuvana tarvitaan itse pelihahmo. Tämä eroaa muista siten että pelihahmo on tarkoitus animoida. Käytännössä tämä tapahtuu siten että piirretään erillisinä kuvina useassa eri asennossa, joita sitten toistetaan peräkkäin pelissä. Näin syntyy illuusio liikkuvasta pelihahmosta. Koska tämän pelin pelihahmona on pieni salamapallo (kuva 10), ei sen animoinnista tule vaikeaa, vain pientä sähköistä väreilyä sen pinnalla.


Kuva 10. Pelihahmo.

Grafiikoiden ollessa valmiita ne siirretään oikeaan paikkaan projektin kansiossa. Seuraavaksi vain lisätään Coreen grafiikoiden lataus muistiin sovelluksen käynnistäessä sekä kuvien piirtäminen geometristen laatikoiden sijaan draw -metodeihin.

4.7 Audio

Lopuksi elävöitetään peliä äänillä. Helpoin tapa tähän on etsiä internetistä sivustoja joilla tarjotaan vapaaseen käyttöön ääniefektejä ja musiikkia ja ladata sieltä. Valitaan sopivan kuuloiset ääniefektit pelaajan hyppyyn ja seinään törmäykseen, sekä sopivan oloinen taustamusiikki jota voi toistaa pelin taustalla uudestaan ja uudestaan. Kun sopivat äänet ovat löytyneet, kopioidaan ne projektin kansioon ja lisätään koodia käynnistämään äänet oikeassa kohdassa.

Peli on tämän jälkeen kaikin puolin valmis. Voidaan siis siirtyä kohti julkaisua.

5 Pelin julkaisu

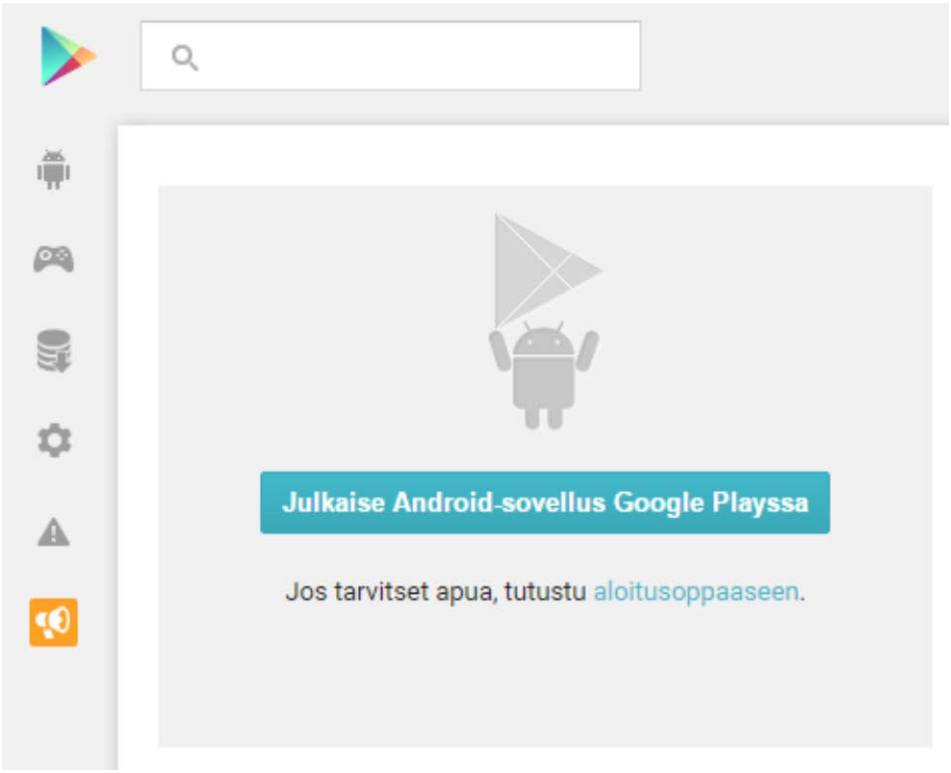
Julkaisu tapahtuu Googlen virallisessa sovelluskaupassa Google Play:ssä. Julkaisua varten täytyy joko luoda uusi Google-tili tai päivittää olemassa oleva tili.

5.1 Google Play -tilin luonti

Uuden Google-tilin luonti onnistuu accounts.google.com -osoitteen takaa, tai linkeistä joita löytyy lukuisista googlen järjestelmistä. Uutta käyttäjätiliä varten täytyy antaa seuraavat tiedot: etu- ja sukunimi, haluttu sähköpostiosoite, salasana kahteen kertaan, syntymäaika sekä sukupuoli. Halutessaan voi vielä antaa puhelinnumeron ja nykyisen sähköpostiosoitteensa. Huomattavaa on että haluttu sähköpostiosoite saattaa olla jo käytössä, jolloin täytyy kokeilla jotakin muuta. Tilin luonti on tällä selvä.

Tavallisella käyttäjätilillä ei kuitenkaan voi vielä ladata peliä Google Play kauppaan, vaan tili pitää rekisteröidä kehittäjätiliksi. Tämä vaatii kertaluontoisen 25 dollarin maksun, mutta maksun jälkeen voi ladata sovelluksiaan myyntiin rajatta. Maksun yhteydessä valitaan nimi joka näkyy sovelluskaupassa sovelluksen nimen yhteydessä, ja tämä voi olla eri kuin itse Google-tilin nimi.

Kun tili on onnistuneesti rekisteröity kehittäjäksi, aukeaa näkymä sisään Googlen kehittäjäkonsoliin (developer console, kuva 11). Konsolista pääsee käsiksi toimintoihin kuten aloitusoppaaseen tai Googlen pelipalveluihin, mutta tärkeimpänä Android-sovelluksen julkaisuun.



Kuva 11. Näkymä Googlen kehittäjäkonsolista.

5.2 Pelin lataus Play Storeen

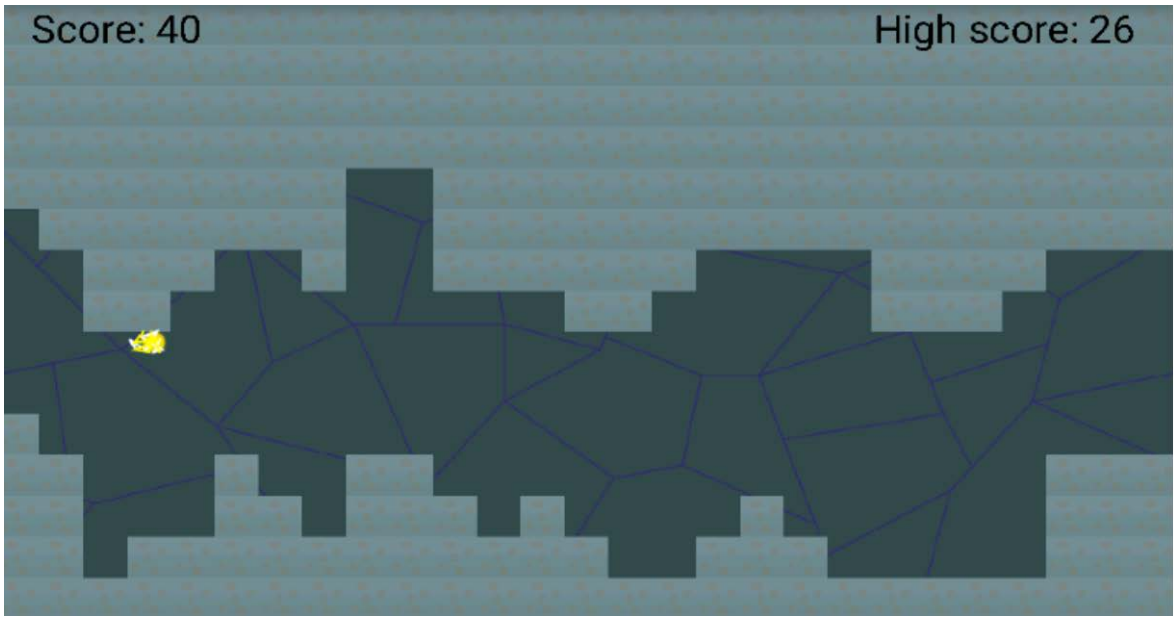
Siirrytään konsolista Android-sovelluksen julkaisuun. Järjestelmä kysyy haluaako ensin ladata APK-paketin vai valmistella kauppasivun. APK-paketti tarkoittaa latausvalmista peliä joka on allekirjoitettu Android Studiolla. Koska tätä ei ole vielä tehty, valmistellaan sivusto ensin.

Julkaisusivulla sovelluksella annetaan nimi, lyhyt kuvaus sekä pidempi kuvaus. Nämä ovat pakollisia tietoja. Tekstin lisäksi sivulle voi ja pitää laittaa graafista sisältöä. Sovelluksesta voi laittaa monenlaista ja kokoista kuvaa pelistä erilaisia Android-laitteita varten, mutta vähintään pitää laittaa korkean resoluution kuvake ja ominaisuuskuva. Tämän jälkeen sovellukselle pitää määritellä luokka, eli minkä tyyppinen sovellus on kyseessä, sekä ikäraja. Lopuksi pitää antaa yhteystiedot jotka näytetään sovelluksen yhteydessä asiakkaita varten. Yhteystiedoista sähköposti on pakollinen.

Kauppasivu on tällä valmis, joten APK-tiedosto pitää vielä ladata. Tämä aloitetaan Android Studiosta generoimalla allekirjoitettu APK. Kun APK on allekirjoitettu, voidaan se ladata kehittäjäkonsoliin. Tämän jälkeen tarkistetaan että kaikki tiedot ovat kohdallaan. Kaiken ollessa kunnossa voi sovelluksen julkaista.

6 Tulokset

Tuloksena projektista syntyi ladattavissa oleva, toimiva Android -peli. Peli kehitettiin alusta loppuun projektin aikana, ja se valmistui juuri aikataulussa. Peli sisältää kaikki suunnitellut ominaisuudet eikä toimintaa haittaavia bugeja jäänyt peliin. Edettävä tunneli luodaan satunnaisesti eteenpäin mennessä, ja mahdottomien kohtien syntyminen saatiin minimoitua/poistettua kokonaan. Peli laskee pisteet pelaajan edetessä, ja tallentaa senhetkisen korkeimman pistemäärän, tosin vain saman pelisession sisällä. Muutamia pieniä puutteita jäi peliin, mutta ne eivät merkittävästi haittaa pelaamista. Valmis peli (kuva x) käyttää itse tehtyjä grafiikoita sekä internetistä ladattuja musiikkeja suunnitelman mukaan. Peli pakattiin latausvalmiiksi Android Studiolla ja julkaistiin ilmaiseksi Google Play -kaupassa, minne ensin luotiin kehittäjäoikeudet omaava käyttäjätili.



Kuva 12. Kuvankaappaus valmiista pelistä.

Käytännön sovelluksen lisäksi projektin tuloksena syntyi myös kuvaus tämän pelin kehitysprosessista. Kuvauksessa käytiin läpi pelin kehitys aina peli-ideasta julkaisuun asti. Kuvauksessa ei juurikaan menty yksityiskohtaisen koodin tasalle, vaan kuvattiin hieman korkeammalta tasolta koodin luokkarakennetta, luokkien tehtäviä sekä niiden suhdetta toisiinsa.

7 Pohdinta

Jo alusta alkaen projektin suurimmaksi riskiksi tunnistettiin käytettävissä ollut rajattu aika. Tämä johtui siitä että ajoissa valmistumisen vuoksi projektin täytyi olla valmis viimeistään tiettynä ajankohtana, ja projektin käynnistämisessä meni sen verran aikaa että lopulta itse projektiin jäi vain reilu kuukausi aikaa. Tästä huolimatta projekti valmistui kuitenkin ajoissa kutakuinkin sellaisena kuin se oli suunniteltu.

Valmistunut peli noudatti melko tarkalleen luvussa kolme selostettua suunnitelmaa. Vaikeimmaksi osaksi arvioimani satunnaisgeneraattori osoittautui juuri siksi, ja vaati paljon hienosäätöä että mahdottomien kohtien syntyminen loppui. Pisteenlaskun lisäksi mukaan tuli myös ylimääräinen ominaisuus eli korkeimman pistemäärän näyttäminen. Tämä rajoittuu kuitenkin vain senhetkiseen pelisessioon, koska pistemäärää ei tallenneta mihinkään. Itsessään tämä ei varmasti olisi vaikea ominaisuus tehdä, mutta ajanpuutteen ja sen vuoksi, että tämä muutenkin oli ylimääräinen ominaisuus, en lähtenyt lisäämään pistemäärään tallennusta tiedostoon.

Havaitsin myös grafiikoita lisätessä että palikoiden kuvien piirto näytölle aiheutti välillä pelissä pientä hidastumista. Tämän korjaaminen olisi vaatinut optimointia, ja ehdin miettiä mahdollista korjaustapaa, mutta se jäi suunnittelun tasolla. Syy tähän oli että ongelma havaittiin myöhäisessä vaiheessa grafiikoiden lisäyksen jälkeen, eikä aikataulussa ollut joustoa optimointiin paneutumista varten.

Kuten edellä on käynyt useammassa kohtaa ilmi, rajattu aikataulu aiheutti pieniä ongelmia ja rajasi korjaustenteon mahdollisuutta. Projektisuunnitelmaan ei ollut kirjattu aikaa testaukselle ja bugien korjaukselle, ja tästä huomasi hyvin että isommassa projektissa tämä voisi hyvin olla kriittinen ongelma. Nyt tehty peli kuitenkin toimii näistä vioista huolimatta ja on pelikunnossa, joten projekti pääsi tavoitteeseensa.

Projektia tehdessä opin monia asioita Androidille ohjelmoimisesta. Vaikka Java on sama kieli pc:llä ja Androidissa, toimii se Androidilla kuitenkin eri tavalla. Huomattavin ero oli graafisen puolen kirjastojen eroavaisuus, kun Androidilla jo nelikulmio muodostettiin hämmentävän poikkeavasti ”perus” Javaan verrattuna. Pelilogiikoiden itsensä koodaus ei kuitenkaan poikennut tavallisesta mitenkään. Jatkossa osaan varautua näihin eroihin paremmin eikä tarvitse ihmetellä näennäisiä bugeja jotka johtuvatkin siitä että koodi toimi eri tavalla kuin olin tottunut olettamaan.

8 Yhteenveto

Projekti lähti liikkeelle yksinkertaisesta peli-ideasta tiukalla aikataululla. Ideasta kehitettiin yksinkertainen mobiilipeli Android-alustalle ja sen kehitystyö dokumentoitiin. Aikataulustaan huolimatta projekti valmistui lopulta aikataulussaan eikä kriittisiä ongelmia ilmennyt. Julkaisukelpoinen pelisovellus ladattiin Google Play -kauppaan ilmaisjakoon.

Kaiken kaikkiaan koin että projekti onnistui, vaikka välitavoitteisiin pääseminen vaatikin loppua kohden hyvin intensiivistä työskentelyä. Suurimmat haasteet liittyivät ajankäyttöön, erityisesti Androidin Java-kirjastoihin tottuminen sekä satunnaisgeneraattorin toimimaan saanti vaati aikansa.

Nyt kun sovellus on julkaistu, on sen jatkokehitys mahdollista, ja tulen varmaan korjaamaan tässä mainitut ongelmat/puutteet tulevaisuudessa. Tämän projektin tuomalta tietopohjalta on myös huomattavasti helpompi lähteä tekemään kokonaan uutta Android -peliä, mikä myös oli yksi tämän projektin tavoitteista.

Lähteet

About Android 2015. Luettavissa:
http://developer.android.com/about/android.html Luettu: 20.10.2015

About Inkscape 2015. Luettavissa:
https://inkscape.org/en/about/ Luettu: 20.10.2015

Android Developer 2015.a. Luettavissa:
http://developer.android.com/tools/studio/index.html Luettu: 20.10.2015

Android Developer 2015.b. Luettavissa:
https://developer.android.com/sdk/index.html Luettu: 20.10.2015

Ron Amadeo, The history of Android 2015. Luettavissa:
http://arstechnica.com/gadgets/2014/06/building-android-a-40000-word-history-of-googles-mobile-os/?all=1 Luettu: 20.10.2015

Gary Cutlack, What is Google Play 2015. Luettavissa:
http://www.techradar.com/news/phone-and-communications/mobile-phones/what-is-google-play-1073348 Luettu: 20.10.2015

What is Java Technology and why do I need it? Luettavissa:
https://www.java.com/en/download/faq/whatis_java.xml Luettu: 20.10.2015