

Jarno Lahti

Mobiilipelin tekninen toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

15.9.2012

|  |  |  |
| --- | --- | --- |
| Tekijä(t)  Otsikko  Sivumäärä  Aika | | Jarno Lahti  Mobiilipelin tekninen toteutus  xx sivua + x liitettä  15.9.2012 |
| Tutkinto | | Insinööri (AMK) |
| Koulutusohjelma | | Tietotekniikan koulutusohjelma |
| Suuntautumisvaihtoehto | | Ohjelmistotekniikka |
| Ohjaaja(t) | | Tehtävänimike Etunimi Sukunimi  Tehtävänimike Etunimi Sukunimi |
|  | | |
| Avainsanat |  | |

|  |  |  |
| --- | --- | --- |
| Author(s)  Title  Number of Pages  Date | First name Jarno Lahti  Technical implementation of a mobilegame  xx pages + x appendices  15 September 2012 | |
| Degree | Bachelor of Engineering | |
| Degree Programme | Information Technology | |
| Specialisation option | Software Engineering | |
| Instructor(s) | First name Last name, Title (for example: Project Manager)  First name Last name, Title (for example: Principal Lecturer) | |
|  | | |
| Keywords | |  |

Sisällys

Lyhenteet

[1 Johdanto 1](#_Toc446956659)

[2 Mobiilipelit 1](#_Toc446956660)

[3 Mobiilipelikehitys 4](#_Toc446956661)

[3.1 Työkaluja mobiilipelikehitykseen 5](#_Toc446956662)

[3.1.1 Unity 6](#_Toc446956663)

[3.1.2 Cocos2D-X 6](#_Toc446956664)

[4 Ohjelmointikielet 6](#_Toc446956665)

[4.1 C++ 6](#_Toc446956666)

[4.2 C# 7](#_Toc446956667)

[4.3 Java 7](#_Toc446956668)

[4.3.1 Olio-ohjelmointi 8](#_Toc446956669)

[5 Insinöörityössä käytetyt työkalut 10](#_Toc446956670)

[5.1 Android studio 10](#_Toc446956671)

[5.2 LibGDX viitekehys 10](#_Toc446956672)

[5.3 Git-versionhallintajärjestelmä 14](#_Toc446956673)

[6 SumTower 17](#_Toc446956674)

[6.1 Pelin tavoite 18](#_Toc446956675)

[6.1.1 Palkitseminen ja virhesiirrot 19](#_Toc446956676)

[6.2 Pelimuodot 20](#_Toc446956677)

[6.2.1 Turn mode 20](#_Toc446956678)

[6.2.2 Time mode 20](#_Toc446956679)

[6.2.3 Journey mode 20](#_Toc446956680)

[7 Pelin toteutus 21](#_Toc446956681)

[7.1 Projektin luonti 21](#_Toc446956682)

[7.2 Versionhallinta 22](#_Toc446956683)

[7.3 Pelimekaniikka 23](#_Toc446956684)

[8 Pohdinta 32](#_Toc446956685)

[Lähteet 34](#_Toc446956686)

Liitteet

Liite 1. Liitteen nimi

Liite 2. Liitteen nimi

Lyhenteet

FPS First person shooter. Peligenre, jossa pelaaja kokee pelin pelattavan hahmon näkökulmasta.

APK Android application package. Tiedostomuoto jota Android käyttöjärjestelmä käyttää applikaatioiden jakamiseen ja asentamiseen.

API Application Program Interface. Joukko työkaluja ja sääntöjä joita hyödynnetään sovelluskehityksessä.

HTTP Hypertext Transfer Protocol. Protokolla tiedon lähetykseen ja vastaanottamiseen.

HTTPS Hypertext Transfer Protocol Secure. Suojattu protkolla tiedon lähetykseen ja vastaanottoon.

# Johdanto

Opinnäytetyön aiheena on Taphold Oy:lle tuotetun mobiilipelin tekninen toteutus. Taphold Oy on kuuden opiskelijan perustama peliyhtiö joka keskittyy mobiilipelien tuottamiseen ja julkaisuun. Taphold Oy:n ensimmäinen mobiilipeli SumTower on julkaistu Androidille Googlen Play kaupassa, joka on Android sovelluksille tarkoitettu kauppapaikka. SumTower on pulmapeli, jossa yhdistyy matematiikka ja match-3 henkinen pelimekaniikka.

Tässä raportissa käyn läpi SumTowerin pelilogiikan teknistä toteutusta ja kerron tekniikoista sekä työkaluista, joita pelin tuottamiseen käytettiin. Selitän myös lyhyesti mobiilipelien historiasta sekä käyn läpi java ohjelmointikieltä ja olio-ohjelmointia. Tavoitteena opinnäytetyössä on tarkastella mobiilipelikehitystä libgdx sovelluskehyksellä ja kehittää omaa osaamista mobiilipelikehityksessä.

# Mobiilipelit

Mobiilipelit ovat videopelejä, jotka on suunniteltu pelattavaksi useimmiten esimerkiksi Matkapuhelimella, tablet–tietokoneella ja älypuhelimilla. Tästä voi olettaa, että mobiilipelejä olisivat kaikki kannettavalla laitteella pelattavat pelit, mutta näin ei kuitenkaan ole. Niinsanottujen käsikonsolien, kuten PlayStation Portable tai Nintendo Game Boy pelit eivät lukeudu mobiilipeleiksi, vaikka ne mukana kätevästi kulkevatkin.**[1.]**

Mobiilipelien historia alkoi vuodesta 1997, jolloin suomalainen matkapuhelinyhtiö Nokia julkaisi Nokia 6110 –matkapuhelimen. Tähän matkapuhelimeen oli esiasennettu Snake, joka tunnetaan myös nimellä matopeli. Matkapuhelimissa näytöt olivat silloin hyvin pieniä ja kaksivärisiä, joten peli ei ollut mitenkään graafisesti hieno, vaan yksinkertaisia mustista pikseleistä koostuvia muotoja vihreällä taustalla. Pelin tavoite oli niinkin yksinkertainen, että pelaajan täytyi ohjata pelihahmona toimivaa matoa saaliiden luo. Peli loppui jos mato törmäsi seinään tai itseensä. Mato kasvoi jokaisen syödyn saaliin jälkeen, joka toi peliin lisää vaikeusastetta kokoajan, koska madon kasvaessa pelialue käytännössä pieneni. **[2.]** Madon ohjaaminen tapahtui matkapuhelimen numeronäppäimistöä hyödyntäen, mikä ei sinäänsä ollut kovin käyttäjäystävällinen, sillä näppäimistöä ei oltu suunniteltu pelaamiseen.

Kuva 1 Pelikuvaa matopelistä

2000-luvun alkupuolella kuluttajamarkkinoille alkoi saapua värinäytöllisiä matkapuhelimia, joten mobiilipelaaminen koki uuden merkittävän käänteen, kun pelit saivat syvyyttä värien kautta. Toinen merkittävä asia, joka kiihdytti mobiilipelien kehittymistä Java 2 Micro Edition (J2ME) ja se että uudet laitteet alkoivat tukea sitä. **[2].** J2ME on sulautetuille järjestelmille tarkoitettu versio Javasta. J2ME mahdollisti sen, että samaa koodia pystyi ajamaan useilla eri laitteilla ja jopa eri valmistajien laitteilla. Pelejä ja ohjelmia pystyi myös kehittämään ja testaamaan työpöytäympäristössä, joka nopeutti kehitystyötä. **[3.]**

Kuva 2 Pelikuvaa Jamdat Bowling pelistä

Vuonna 2007 alkoi matkapuhelimien uusi aikakausi ja tämän myötä myös uusi aikakausi mobiilipeleille, kun yhdysvaltalainen yritys Apple julkaisi sen ensimmäisen matkapuhelimien iPhonen. iPhone oli kosketusnäytöllinen laite, jonka käyttöjärjestelmä oli optimoitu täysin käytettäväksi pelkästään laitteen kosketusnäyttöä hyödyntäen. **[4.]** Tämä tarkoitti mobiilipelien kannalta täysin uudenlaista tapaa kontrolloida peliä. Hyvänä esimerkkinä tästä uudesta tavasta pelata on Rovion vuonna 2009 julkaisema Angry Birds -peli, missä yksinkertaisuudessaan oli tarkoituksena lingota ritsalla lintuja possuja päin. Peli käytti hyödyksi kosketusnäyttöä siten, että ritsalla olevasta linnusta ”otettiin kiinni” painamalla sitä sormella, jonka jälkeen sormea vieritettiin laitteen ruudulla taakseppäin ja päästettiin irti. Tämä tapa lingota lintu jäljitteli hyvin tapaa, miten oikeassakin maailmassa henkilö ampuisi ritsalla.



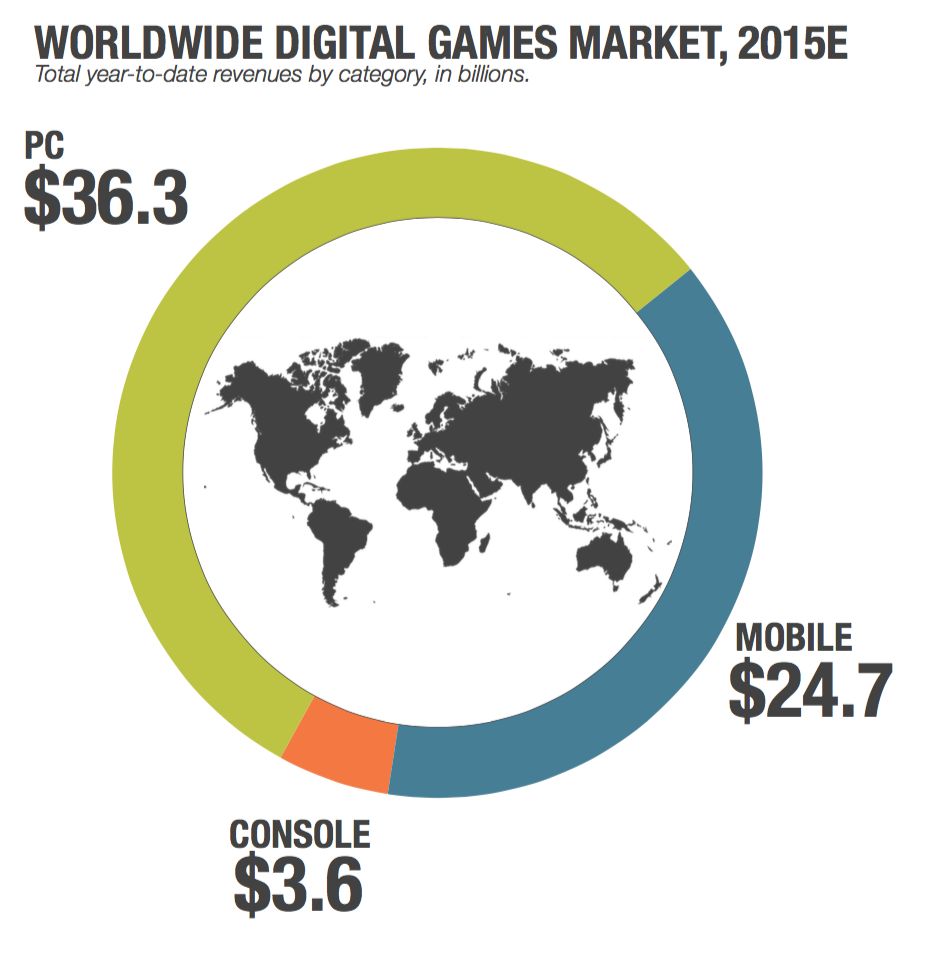
Kuva 3. Angry Birds -pelin ohjekuva

Tänä päivänä mobiilipelit ovat kehittyneet niin pitkälle, että ne muistuttavat jo erittäin paljon konsolipelejä. Tästä hyvänä esimerkkinä on Gameloftin julkaisema Modern Combat pelisarja, jotka ovat 3D FPS pelejä. Pelissä pelaaja toimii sotilaana joka suorittaa määrättyjä tehtäviä tuhoamalla kohteita käyttäen erilaisia aseita. Uusimmissa sarjan peleissä pelaaja voi pelata verkon yli toisia pelaajia vastaan. Pelisarjaa on julkaistu viisi eri versiota mobiililaitteille. Pelisarjassa on erittäin paljon vaikutteita hyvin kuuluisista konsoleille ja PC:lle julkaistuista pelisarjoista, kuten Call of duty ja Battlefield.**[5.]**

Kuva 4 Modern Combat 5: blackout

# Mobiilipelikehitys

Mobiilipelit ovat nousseet yhdeksi tärkeimmistä alustoista niin pelaajille kuin myös pelikehittäjille ja julkaisijoille. Vuonna 2015 mobiilipelien maailmanlaajuinen tuotto oli noin 25 miljardia dollaria. Vertailukohteeksi voidaan ottaa toinen erittäin pitkään markkinoilla ollut alusta eli PC jonka maailmanlaajuinen tuotto oli samana vuonna noin 36 miljardia dollaria. Ottaen huomioon sen, että mobiilipelit on vielä nuori alusta on se silti kasvanut pienessä ajassa isoksi tekijäksi maailmanlaajuisesti.**[6.]**



Kuva 5 Maailmanlaajuinen tuottojakauma eri alustojen välillä

Indie-kehittäjille mobiilipelit ovat helpoin tapa saada oma pelinsä kaikkien nähtäväksi, sillä Applen App storeen ja Googlen Play -sovelluskauppaan voi kuka tahansa julkaista omia pelejään. Kehittäjä voi tehdä peleillään rahaa lisäämällä peleihin esimerkiksi mainoksia tai pelin sisäisiä mikromaksuja, joilla pelaaja voi ostaa itselleen oikealla rahalla peliin sisältöä esimerkiksi hahmolleen uusia asusteita.

## Työkaluja mobiilipelikehitykseen

Kehittäjille on tarjolla monenlaisia työkaluja, pelimoottoreita ja sovelluskehyksiä pelien tekemiseen, mistä kukin voi valita mieleisensä. Kun mobiilipeliä lähdetään tekemään, on hyvä päättää projektin alussa esimerkiksi että millä ohjelmointikielellä projekti halutaan toteuttaa ja mille mobiilialustoille. Seuraavaksi esitellään muutama tunnettu pelinkehitys sovelluskehys.

### Unity

Unity on ehkä jopa eniten käytetty sovelluskehys mobiilipeleille. Esimerkiksi hyvin moni iOS peli on tehty käyttäen unityä. Unity tukee kolmea eri ohjelmointikieltä, jotka ovat C#, UnityScript ja Boo. Unity on järjestelmäriippumaton sovelluskehys, joka tarkoittaa sitä että sillä tehtyjä pelejä voidaan ajaa joko mobiilissa, PC:llä tai tunnetuilla konsoleilla kuten Xbox ja PS3. Unity tarjoaa kaksi versiota. Ilmaisversiolla käyttäjä voi kehittää pelejä mobiiliin ja PC:lle kun taas maksullinen versio tarjoaa myös tuen konsolikehitykselle.**[7.]**

Unity on hyvin monipuolinen ja monenlaiseen projektiin soveltuva sovelluskehys. Sillä voidaan tehdä yksinkertaisista 2D tasohyppelypeleistä aina vaativiin monipelattaviin 3D FPS-peleihin asti ja sillä kehittäminen on myös erittäin nopeaa. Tästä syystä unity on niin suosittu kehittäjien keskuudessa.

### Cocos2D-X

Cocos2D-X on avoimeen lähdekoodiin perustuva järjestelmäriippumaton sovelluskehys, jolla voidaan tehdä näyttäviä 2D-pelejä mobiilille ja PC:lle. Cocos2D-X tukee kolmea eri ohjelmointikieltä C++, Luaa ja JavaScriptä. Cocos-2D-X sisältää monia työkaluja, jotka helpottavat pelien tekemistä. Muutamia näistä työkaluista ovat esimerkiksi sisäänrakennettu fysiikkamoottori, käyttöliittymätyökalut, äänentoisto ja verkkotyökalut. **[8.]**

Cocos2D-X:ää käyttävät monet isot tekijät mobiilipelialalla. muun muassa Zynga, Glu ja suomalainen mobiilipeliyhtiö Fingersoft käyttävät Cocos2D-X:ää peliensä tuottamiseen.**[8.]**

# Ohjelmointikielet

Projektin ohjelmointikieleksi on valittu Java pääsääntöisesti siksi, koska projektin toteutuksessa käytettävä sovelluskehys LibGDX käyttää ohjelmointikielenä Javaa. Myös projektin tiimin sisällä on aikaisempaa kokemusta javalla ohjelmoimisesta, joten se helpottaa projektin etenemistä huomattavaasti. Seuraavaksi kuitenkin esitellään javan oheella lyhyesti myös C++ ja C#.

## C++

C++ on näistä kolmesta kielestä vanhin. Sen kehitti Bjarne Stroustrup vuonna 1979. C++ on käytännössä C-ohjelmointikielen lisäosa. C++ tukee proseduraalista, olio ja geneeristä ohjelmointia. Kun vertailee C++ ja javan syntakseja, niin huomaa väkisinkin yhtenäisyyksiä, sillä java on ottanut erittäin paljon vaikutteita C++:sta. Eroavaisuuksia kuitenkin löytyy myös erittäin paljon. Suurin ero on se, että C++ ohjelmaa ajetaan suoraan laitteen raudalla, kun taas java ohjelmaa ajetaan virtuaaliympäristössä. Tämän takia C++ ohjelmat ovat yleensä paljon suorituskyvyltään paljon parempia. Toinen merkittävä eroavaisuus liittyy muistinhallintaan. C++:ssa kehittäjä pystyy itse määrittämään, milloin esimerkiksi jokin olio poistetaan ja sen varaama muisti vapautetaan uudelleenkäytettäväksi. Javassa niinsanottu roskien keräys on toteutettu automaattisesti, joten kehittäjän ei siitä tarvitse huolehtia. Koska C++:ssa kehittäjä pystyy itse määrittämään milloin muistia vapautetaan, tulee ohjelmista tehokaampia, mutta se on työläämpää, koska se pitää muistaa ottaa huomioon.

## C#

C# on microsoftin luoma olio-ohjelmointikieli. C# on hyvin saman tyylinen

## Java

Java on Sun Microsystemsin vuonna 1995 julkaisema alusta ja ohjelmointikieli, joka on tänä päivänä Oraclen hallinnoima ja ylläpitämä. Java on globaali standardi sulautetuille-, mobiilisovelluksille, peleille verkkosisällölle ja yritysohjelmistoille. Yli 9 miljoonaa sovelluskehittäjää maailmassa käyttää javaa. **[9.]**

Java on korkean tason ohjelmointikieli. Java-ohjelmia ajetaan java virtuaalikoneella (Java Virtual Machine JVM). Java-ohjelmoinnissa kaikki lähdekoodi kirjoitetaan puhtaana tekstinä, joka on ihmisille täysin luettavaa. Tämä tekee siitä korkean tason ohjelmointikielen. Lähdekoodin tiedostot loppuvat päättellä *.java*, jotka sitten ohjelman koontivaiheessa kootaan .*class* tiedostoiksi. Nämä .*class* tiedostot sisältävät tavukoodia, joka on JVM:n kieli. **[10.]**



Kuva 6 java-ohjelman koontiprosessin havainnollistaminen [10.]

Koska JVM on saatavilla monelle eri käyttöjärjestelmälle tarkoittaa tämä sitä, että samaa koodia voidaan ajaa millä tahansa laitteella, joka tukee JVM:ää. Koska ohjelmia ajetaan virtuaaliympäristössä voi ohjelmien suorituskyky olla hieman heikompi, kuin laitteen natiivilla ohjelmointikielellä kirjoitettu ohjelma. **[10.]**

### Olio-ohjelmointi

Olio-ohjelmoinnilla viitataan ohjelmointityyppiin missä ohjelmoija luo datarakenteeseen myös toiminnot eli funktiot, joita tämä datarakenne voi käyttää. Tällä tavalla datarakenteesta tulee olio, joka sisältää tiedon lisäksi myös toimintoja. Tärkeimpiä etuja olio-ohjelmoinnille verrattuna proseduraaliseen ohjelmointiin on se, että ohjelmoija voi luoda moduuleja, joita ei tarvitse muuttaa, kun halutaan luoda uudentyyppinen objekti. Uusi objekti voi periä toiminnallisuutta jo toimivalta vanhalta objektilta, joka tekee olio-ohjelmointia hyödyntävien ohjelmien muokkaamisesta helppoa **[11.]**

Java on luokkapohjainen olio-ohjelmointikieli. Luokkapohjaisessa olio-ohjelmoinnissa keskeisenä asiana ovat luokat. Luokat tarjoavat tavan määritellä joukon olioita ja toiminnallisuuden manipuloida niitä. Oliot ovat ilmentymiä luokista. Luokat ovat niin sanotusti muotteja jostain asiasta. Luokka sisältää datan ja operaatiot, joita datalle voidaan suorittaa. Luokka voi esimerkiksi kuvata linkattua listaa, Tietokoneen näytöllä näkyvää ikkunaa, Tiedostoa tai vaikkapa huonekalua. **[12, s.16].** Seuraavassa esimerkkikoodissa havainnollistetaan miltä luokka voi näyttää sisältäpäin ja miten sitä käytetään pääohjelmassa. Esimerkiksi on otettu koira, josta tehdään luokka ja esitellään, kuinka tätä luokkaa käytetään.



Esimerkkikoodi 1 Luokan rakenteen, olioiden luomisen ja käyttämisen havainnollistaminen

Esimerkkikoodissa 1 on ensin esitettynä Koira -luokka, joka sisältää String -tyyppisen muuttujan *koiranNimi*, luokan konstruktori, joka ottaa vastaan String tyyppisen muuttujan *nimi*, joka sitten asetetaan edellä mainittuun *koiranNimi -*muuttujaan. Viimeiseksi luokasta löytyy funktio, joka tulostaa tekstin konsoliin. Seuraavaksi on esitelty ohjelman pääluokka, jossa on ensin määritettynä kaksi **Koira** -tyyppistä muuttujaa *musti* ja *röffe.* Seuraavaksi näistä kahdesta muuttujasta luodaan oliot kutsumalla Koira -luokan konstruktoria, joka ottaa vastaan nimen String -tyyppisenä muuttujana. Viimeiseksi molemmat oliot kutsuvat Koira -luokan funktiota *hauku,* jonka jälkeen konsoliin tulostuu tekstiä.

# Insinöörityössä käytetyt työkalut

## Android studio

Android Studio on virallinen ohjelmointiympäristö Android -applikaatioille ja se perustuu IntelliJ IDEA Java -ohjelmointiympäristöön. Se tarjoaa kehittäjälle useita työkaluja, kuten esimerkiksi:

* Gradleen perustuvan koontijärjestelmän
* Koontivariaatioita ja moni-APK tiedosto generaation
* Valmiita koodipohjia, jotka auttavat yleisten toimintojen tekemisessä
* Sommittelueditorin, jossa on tuettuna raahaa ja pudota –teemaeditointi
* Lint –työkaluja joiden avulla voidaan napata suorituskykyyn, käytettävyyteen, yhteensopivuuteen ja muihin ongelmiin liittyviä virheitä
* ProGuard ja applikaation signeeraus
* Sisäänrakennettu tuki googlen pilvipalveluille.

Android studio tarjoaa myös monia muita hyödyllisiä työkaluja. **[13.]**

## LibGDX viitekehys

Libdgx on avoimeen lähdekoodiin perustuva järjestelmäriippumaton peli- ja applikaatioviitekehys, joka tukee Windows, Linux, Mac OS X, Android, Blackberry, iOS ja HTML5 alustoja. Libgdx mahdollistaa sen, että kehittäjä voi kirjoittaa koodia ja se kääntyy kaikille yllä mainituille alustoille ilman minkäänlaisia muutoksia koodiin. Tämä mahdollistaa nopean pelin kehityksen, koska kehittäjä voi käytännössä ohjelmoida ja testata peliä, joka on tarkoitettu pelattavaksi esimerkiksi mobiililaitteella, täysin työpöytäympäristössä. Libgdx käyttää Javaa ohjelmointikielenä ja se antaa kehittäjälle käyttöön koko Javan tarjoaman ekosysteemin, mikä mahdollistaa kehittäjän olla luova. Libgdx tähtää siihen, että se ei olisi pelimoottori vaan lähempänä viitekehystä. Se antaa kehittäjälle vahvat työkalut, joista valita ja antaa kehittäjän itse päättää, miten haluaa pelin tai applikaationsa kirjoittaa. **[14.]**

Liittessä 1 on esitettynä Android- ja työpöytäsovelluksen käynnistysluokat. Nämä luokat sisältävät mahdolliset alustariippuvaiset alustukset ja asetukset. Esimerkiksi luokassa DesktopLauncher alustetaan ensimmäisenä olio config, joka on tyyppiä LwjglApplicationConfiguration. Tälle oliolle voitaisiin sitten määrittää esimerkiksi applikaation ruudun leveys ja korkeus pikseleissä tai vaikka asettaa applikaatio kokonäyttötilaan. Kun mahdolliset asetukset on määritetty, siirrytään itse applikaation applikaation avaamiseen. Työpöytäsovelluksella luodaan uusi LwjglApplication -tyyppiä oleva instanssi, jolle annetaan parametreiksi edellä mainitty config -olio ja uusi instanssi projektin pääluokasta, jossa sijaitsee applikaation toimintalogiikka. Tämä olio sitten käynnistää applikaation käyttäjälle näkyvän osuuden eli tässä tapauksessa peliruudun, joka on esimerkiksi leveydeltään ja korkeudeltaan edellä mainitun config –olion määritysten mukainen.

Libgdx on avoin viitekehys, sillä se antaa kehittäjän käyttää alhaisen tason toimintoja, kuten tiedostojärjestelmiä, syöttölaitteita, audiolaitteita sekä OpenGL:ää yhdistetyn OpenGL ES 2.0 ja 3.0 rajapinnan kautta. Näiden alhaisen tason laitteiden päälle on rakennettu useita sovellusrajapintoja, joiden avulla yleiset toiminnot, kuten esimerkiksi spritejen ja tekstin renderöinti, käyttöliittymien rakentaminen, musiikin toistaminen, erilaisten matemaattisten funktioiden suorittaminen ja eri tietotyyppien jäsentely, onnistuu vaivatta. **[14.]**

Libgdx tekee tarvittaessa kaikki natiiviin koodiin liittyvät toiminnot itsenäisesti ja ne suoritetaan yleensä koodin käännösvaiheessa, jolloin kehittäjän ei tästä tarvitse välittää **[14].** Natiivilla koodilla tarkoitetaan koodia, joka on suunniteltu ajettavaksi määrätynlaisella prosessorityypillä. Natiivia koodia ei siis pysty ajamaan muilla prosessoreilla ellei koodia ole tarkoitettu emuloitavaksi. **[15.]**

Libgdx –sovelluksen elinkaari koostuu viidestä eri vaiheesta, jotka ovat luonti(create), pysäytys(pause), jatkaminen(resume), renderöinti(render) ja hävittäminen(dispose). Näihin ohjelman vaiheisiin kehittäjä pääsee käsiksi implementoimalla applikaationsa pääluokkaan libgdx:n oman rajapinnan *ApplicationListener*. ApplicationListener rajapinta sisältää metodit create, resize, render, pause, resume ja dispose, joiden avulla kehittäjä pääsee käsiksi sovelluksen elinkaareen.**[16.]** Metodien käyttötarkoitukset ovat seuraavat:

* **Create:** Tätä metodia kutsutaan vain kerran sovelluksen käynnistyksen yhteydessä, joten siinä voi tehdä esimerkiksi sovellukseen liittyvät initialisaatiot
* **Resize:** Tätä metodia kutsutaan heti createn jälkeen ja myös aina kun ohjelman ikkunana kokoa muutetaan esimerkiksi työpöytäsovelluksessa. Tämä metodi ottaa vastaan kaksi int -tyyppistä parametriä, jotka ovat pikselileveys ja –korkeus.
* **Render:** Tätä metodia kutsutaan sovelluksen pääsilmukassa aina kun piirtämistä kuuluu tapahtua. Tätä metodia käytetään yleensä myös esimerkiksi pelilogiikan päivittämiseen.
* **Pause:** Kaikilla sovellusalustoilla tätä metodia kutsutaan, kun sovellusta ollaan lopettamassa. Jos sovellusalustana on Android, niin tätä metodia kutsutaan, kun sovellus halutaan pysäyttää. Tälläisiä tilanteita ovat esimerkiksi, kun käyttäjälle tulee puhelu tai käyttäjä painaa laitteen koti –näppäintä.
* **Resume:** Tätä metodia kutsutaan pelkästään Androidilla. Kutsu tapahtuu pause –metodin jälkeen, kun sovellukseen ollaan palaamassa esimerkiksi, kun käyttäjä lopettaa puhelun, joka keskeytti sovelluksen aikaisemmin.
* **Dispose**: Tätä metodia kutsutaan kun sovellus ollaan lopettamassa. Ennen dispose -metodin kutsumista kutsutaan pause –metodia.



Kuva 3 Libgdx –sovelluksen elinkaari suomeksi havainnollistettuna [16].

Libgdx:n ydin koostuu kuudesta eri moduulista, jotka tarjoavat keinon kommunikoida käyttöjärjestelmän kanssa. Näitä moduuleja voidaan kutsua kaikissa järjestelmissä ja ne toimivat kaikissa samalla tavalla. **[17.]** Moduulit ovat seuraavat:

* **Sovellus(Application):** Suorittaa applikaation ja pitää huolen siitä, että API on tietoinen applikaation tason tapahtumista, kuten esimerkiksi ikkunan koon muutoksista.
* **Tiedostot(Files):** Mahdollistaa tiedostojärjestelmän käytön eri alustoilla.
* **Syöttö(Input):** Informoi API:a käyttäjän syöttämistä komennoista, kuten hiiren klikkauksista ja näppäimistön tai kosketusnäytön painalluksista.
* **Verkko(Net):** Tarjoaa tavat päästä käsiksi resursseihin HTTP tai HTTPS protokollia hyödyntäen.
* **Ääni(Audio):** Tarjoaa tavat toisaa ääniefektejä ja musiikkia. Mahdollistaa myös pääsemisen äänilaiteisiin käsiksi koodissa.
* **Grafiikka(Graphics):** Mahdollistaa käsiksi pääsemisen OpenGL ES 2.0 –rajapintaan.

Näihin moduuleihin pääsee käsiksi koodissa Gdx –luokan kautta, jossa ne ovat staattisina muuttujina. **[17.]** Seuraavassa esimerkkikoodissa havainnollistan kuinka näihin moduuleihin pääsee käsiksi koodissa.

Esimerkkikoodi 2 Moduulien alustus

## Git-versionhallintajärjestelmä

Versionhallinta on tärkeä osa-alue ohjelmistokehityksessä, koska sillä helpotetaan tiimityöskentelyn toimintaa huomattavasti. Versionhallinnalla tarkoitetaan yleensä työkalua tai menetelmää, jolla pidetään projektissa olevat tiedostot ajan tasalla ja virhetilanteiden sattuessa se mahdollistaa myös paluun edelliseen versioon. Pienissä ohjelmistoprojekteissa versionhallinta voidaan toteuttaa jopa niin yksinkertaisesti, että tiedostoja kopioidaan paikallisesti vain uusiin kansioihin talteen ja esimerkiksi nimetään kansio versionumerolla. Tämä toimii tiettyyn pisteeseen saakka, mutta on erittäin herkkä ongelmatilanteille, kuten esimerkiksi kovalevyn rikkoutumiselle tai muuten vaan datan katoamiselle. **[18, s.1]**

Git on versionhallintajärjestelmä, joka perustuu hajautettuun versionhallintamalliin. Hajautetulla versionhallinnalla tarkoitetaan sitä, että projektin tiedostoilla on jokin keskitetty tietokanta, josta jokainen käyttäjä kopioi koko tietokannan paikallisesti omalle työkoneelleen. Käyttäjät tekevät muutoksia näihin paikallisiin tiedostoihin ja lopulta lähettävät itse tekemät muutokset keskitettyyn tietokantaan. Koska koko tietokannan sisältö on jokaisen omalla työkoneella ja kaikki muutokset tehdään paikallistiedostoihin, mahdollistaa tämä helpon tavan palauttaa varmuuskopio tietokannasta, jos esimerkiksi palvelin hajoaa. **[18, s.4]**

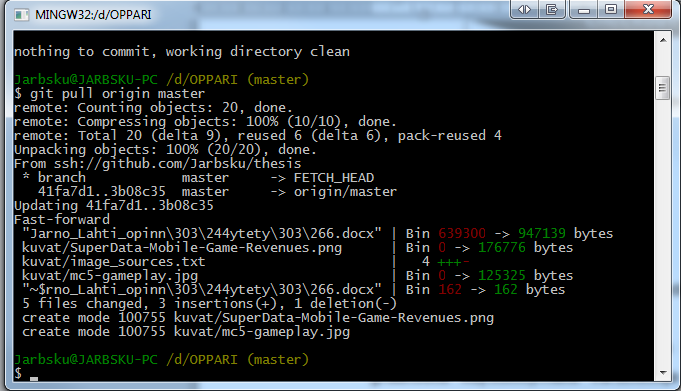


Kuva 4. Hajautetun versionhallinnan havainnollistamiskaavio.

Git:in toiminta perustuu kolmeen eri vaiheeseen, joissa tiedostot voivat olla. kommitoituna(commited), modifioituina(modified) tai esitetty(staged). Kommitoitu tarkoittaa, että data on tallennettu paikalliseen tietokantaan. Modifioitu tarkoittaa, että tiedostoissa on tapahtunut muutoksia, mutta niitä ei ole vielä kommitoitu. Esitetty tarkoittaa sitä että muokatuista tiedostoista on otettu talteen nykyinen tila ja määritetty se menemään seuraavassa tallennuksessa paikalliseen tietokantaan. Tämä tarkoittaa sitä, että git -projektissa on kolme eri lohkoa, joissa tiedostot ovat: Git -kansio, työkansio ja esitysalue. Git-kansio sisältää metadatan ja projektin tietokannan. työkansio sisältää kopiot tietokannan tiedostoista, joita käyttäjä muokkaa. Esitysalue on git-kansion sisällä oleva tiedosto, joka sisältää informaatiota seuraavaan kommittiin menevästä datasta. **[18, s.8]**



Kuva 5. Gitin kolme päätilaa ja niissä tiedostojen liikkuminen.

Työskentely giti:llä tapahtuu yleensä komentorivin kautta, mutta vaihtoehtoisia graafisella käyttöliittymällä varustettuja ohjelmistoja on myös saatavilla. Näissä graafisissa ohjelmistoissa on yleensä ongelmana se, että niillä ei voi tehdä kaikkea mahdollista, mitä komentorivillä työskentelyllä voi tehdä. Tämä johtuu siitä, että graafiset ohjelmistot usein tarjoavat käyttäjälle vain osan suoritettavista komennoista. **[18, s.9]**

Kuva 7 Gitin komentorivityökalu

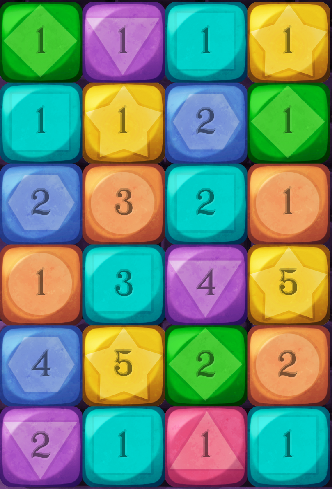
# SumTower

SumTower on matemaattinen pulmapeli, missä pelaajan on tarkoitus saada muodostettua pelilaudalle mahdollisimman suuri yhtenäinen summa. Pelilaudalla on yhteensä 24 palikkaa 4x6 ruudukossa, jotka toimivat peliobjekteina. Palikka sisältää positiivisen luvun, joka on alussa 1. Palikka voi olla yksi seitsemästä eri väristä, jotka ovat seuraavat: vihreä, punainen, turkoosi, keltainen, oranssi, sininen ja lila. Pelaaja poistaa palikoita koskettamalla peliruutua halutun palikan kohdalta, jonka jälkeen palikka ja sen sisältämä luku häviävät pelilaudalta. Tämän jälkeen poistetun palikan yläpuolella ollut palikka tippuu alas ja ”syö” mahdolliset viereiset sekä alapuolella olevan samanvärisen palikan ja lisää omaan lukuunsa kaikkien syötyjen palikoiden sisältämät luvut. Tätä kutsutaan pelissä mergeämiseksi. Uusia palikoita ilmestyy pelilaudan yläriville aina kun siellä on tilaa. Ilmestyvillä palikoilla on satunnaisesti generoitu väri seitsemästä ylempänä mainitusta väristä ja lukuarvona yksi.



Kuva 8 SumTower pelilauta pelin alussa

## Pelin tavoite

****Tavoite pelissä on saada mahdollisimman iso summa pelilaudalle poistamalla palikoita, jotka takaavat pelaajalle hyvän siirron. Esimerkiksi pelin alussa kaikki 24 palikkaa sisältävät numeron yksi, jolloin pelilaudan kokonaissumma on 24. Kun pelaaja poistaa palikan laudalta, joka esimerkiksi takaa hänelle kahden samanvärisen palikan mergeämisen. Tällöin mergeävä palikka saa luvukseen kaksi ja lauta täyttyy automaattisesti takaisin täyteen, jonka jälkeen laudan kokonaussumma on 25. Tätä sääntöä käyttäen pelaaja sitten etenee pelissä ja yrittää saada mahdollisimman ison summan laudalleen.

Kuva 9 SumTower pelilauta muutaman siirron jälkeen

Pelilaudalle voi pelin aikana tulla tilanteita, missä pelaaja huomaa, että hänellä on mahdollista tehdä vain huono siirto. Tähän ongelmaan pelaajalla on käytössään työkaluja, joita pelissä kutsutaan boostereiksi.

### Palkitseminen ja virhesiirrot

**Chain:** Kun pelaaja tekee siirron, jonka johdosta pelilaudalla tapahtuu useampi merge. Tälläinen onnistunut siirto palkitsee pelaajaa siten, että jokaiseen chainissa mukana olleeseen palikkaan lisätään siirron jälkeen vielä yksi kyseisen palikan sisältämään lukuun.

**Double merge:** Kun pelaaja tekee siirron, jonka johdosta yksi palikka mergeää kaksi samanväristä palikkaa ympäriltään. Pelaajaa palkitaan tälläisestä siirrosta. Palkinto riippuu missä pelimuodossa pelaaja siirron tekee.

**Triple merge:** Kun pelaaja tekee siirron, jonka johdosta yksi palikka mergeää kolme samanväristä palikkaa ympäriltään. Pelaajaa palkitaan tälläisestä siirrosta. Palkinto riippuu missä pelimuodossa pelaaja siirron tekee.

**Virhesiirto:** Kun pelaaja tekee siirron, jonka johdosta yksikään palikka ei mergeä. Tästä siirrosta pelaajaa rangaistaan siten, että viidestä satunnaisesta pelilaudan palikasta vähennetään yksi. Palikoiden sisältämä luku ei kuitenkaan voi mennä alle nollan.

## Pelimuodot

SumTower sisältää kolme eri pelimuotoa. Pelimuodot ovat turn, time ja journey mode. Jokainen näistä muodoista sisältää omat sääntönsä ja jokaisessa on oma uniikki palkitsemistapa ylläesitetyille double ja triple mergeille.

### Turn mode

Pelaajalle on annettu aluksi 30 siirtoa. Pelaaja yrittää saada annetuilla siirroilla mahdollisimman suuren loppusumman pelilaudalleen. Yllä mainitut double ja triple merget palkitsevat tässä pelimuodossa pelaajaa lisävuoroilla. Double mergestä pelaaja saa yhden lisävuoron ja triple mergestä kaksi lisävuoroa.

### Time mode

Pelaajalla on 45 sekunttia aikaa saada mahdollisimman suuri loppusumma pelilaudalleen. Double ja triple mergeistä pelaaja saa tässä pelimuodossa lisäaikaa. Double merge antaa pelaajalle kolme lisäsekunttia ja triple merge antaa pelaajalle viisi lisäsekunttia peliaikaa.

### Journey mode

Journey mode eroaa muista pelimuodoista siten, että se ei lopu koskaan. Tässä pelimuodossa pelaaja yrittää rakentaa systemaattisesti tornia palikoista. Ensin pelaajalle annetaan tehtäväksi saada laudalle palikka minkä arvo on 10. Kun pelaaja saa tehtyä palikan mikä vastaa tai on yli annetun luvun, se siirtyy torniin. Tämän jälkeen pelaaja saa uuden luvun, joka hänen pitää muodostaa yhteen palikkaan. Tarvittava luku nousee aina yhdellä. Ensiksi 10 sitten 11, 12, 13 jne. Tämä pelimuoto ei lopu koskaan, vaikka pelaaja sulkisi pelin. Pelaaja voi milloin tahansa jatkaa peliä siitä kohtaa mihin hän viimeksi jäi.

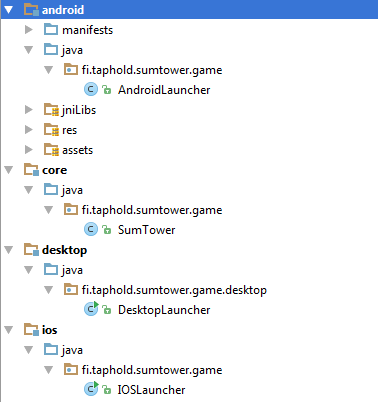
# Pelin toteutus

## Projektin luonti

Projekti aloitetaan käyttäen libgdx:n tarjoamaa graafista projekti generaattoria. Generaattoriin syötetään projektin nimi, Java-paketin nimi, pelin pääluokan nimi, projektin sijainti tietokoneella ja viimeiseksi Android SDK:n sijainti tietokoneella. Seuraavaksi valitaan mitkä kaikki alustat halutaan tukea. Vaihtoehtoina ovat Työpöytä, Android, iOS ja HTML5. Lopuksi on mahdollista vielä valita erilaisia lisäosia, mutta nämä ovat valinnaisia.

Kuva 11 Libgdx:n projekti generaattorin käyttöliittymä

Kun projekti on luotu, on avataan se kehitysympäristössä. Tässä projektissa kehitysympäristönä toimii Android Studio, mutta koska libgdx käyttää projektin riippuvuuksien ja koontiprosessin hallintaan gradlea, on mahdollista käyttää myös muita kehitysympäristöjä, kuten Eclipse tai Netbeans. Android Studio tukee gradle projekteja, mutta esim Eclipseen saattaa joutua lataamaan erillisen lisäosan, jotta sillä pystytään kehittämään libgdx sovelluksia.

Libgdx-projekti koostuu useasta java-projektista. Jokainen kehittäjän valitsema alusta on oma projektinsa ja ne sisältävät kaiken tarpeellisen koodin, joka koskee kyseistä alustaa. Ne sisältävät myös alustan pääluokan, joka on havainnollistettu liittessä 1. Itse sovelluksen tai pelin koodi kirjoitetaan core-projektiin. Kaikki grafiikat ja äänet laitetaan Android projektissa olevaan assets-kansioon, joka sitten linkataan jokaiseen projektiin, jotta ne osaavat hakea tiedostoja sieltä, kun niitä koodissa kysytään.

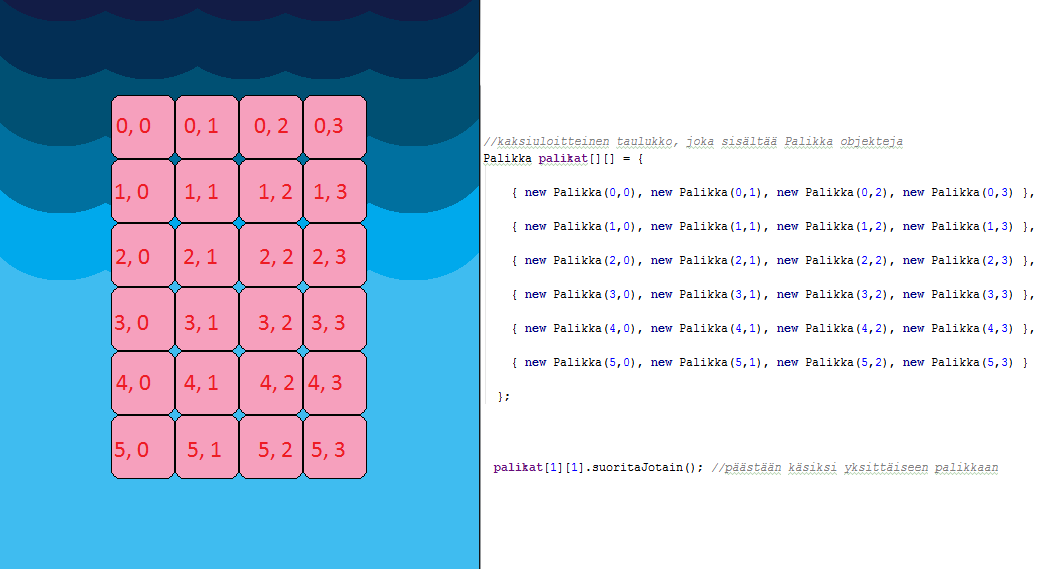
Kuva 12 Libgdx-projekti Android Studiossa

## Versionhallinta

Projektin versionhallintaan käytettiin jo aiemmin esiteltyä Git:iä. Repositorion palveluntarjoajaksi valittiin Bitbucket niminen palveluntarjoaja. Valinnan syynä oli se, että Bitbucket tajosi mahdollisuuden ilmaiseen yksityiseen repositorioon, johon pääsee käsiksi vain mukaan kutsutut henkilöt. Palvelu on ilmainen niin kauan kuin repositorion käyttöön on kutsuttu maksimissaan viisi henkilöä. Toinen mahdollinen palveluntarjoaja olisi ollut Github, mutta he laskuttavat privaateista repositorioista, joten päätös oli tällä tapaa hyvin selvä, koska kehittäjätiimi koostui kolmesta henkilöstä, jotka versionhallintaa käyttivät. Repositorion täytyi olla privaatti, ettei kuka tahansa pysty katsastelemaan pelin lähdekoodia.

## Pelimekaniikka

Alustavien määrittelyiden pohjalta lähdettiin rakentamaan peliä. Alussa tarkoitus oli saada järkevästi piirtymään pelilauta ruudun keskelle ja palikoita piti pystyä poistamaan koskemalla niitä sormella. Pelilaudan palikat täytyi sijoittaa jonkinlaiseen taulukkoon, jossa ne säilötään ja josta niitä käsitellään. Mahdollisuuksia oli tehdä joko normaali taulukko, kaksiulotteinen taulukko tai lista, joka sisälsi kaikki laudalla olevat palikka oliot. Palikat päätettiin säilöä kaksiuloitteiseen taulukkoon, koska tällä tavalla oli helppo tietää tasan tarkkaan, että mikä ruudulla näkyvä palikka oli kyseessä, kun sitä haluttiin käsitellä. Seuraava kuva havainnollistaa miltä kaksiuloitteinen taulukko näyttää ja miten taulukon sisältämiä objekteja pystyi käsittelemään.



Kuva 13 Havainnollistaminen palikoiden sijoittelusta kaksiuloitteisessa taulukossa

Yksittäiseen palikkaan päästään käsiksi koodissa erittäin yksinkertaisesti tällä toteutustavalla ja se on ihmiselle helposti luettava ja ymmärrettävä tapa, joten koodin selkeyden ja luettavuuden vuoksi lähdettiin etenemään kaksiuloitteisella taulukolla.

Palikan luokan rakennetta täyty myös miettiä tässä vaiheessa. Tiedossa oli, että palikka omistaa kokonaisluvun, mitä piti pystyä käsittelemään. Palikalla oli myös väri, mille piti keksiä joku yksinkertainen tapa kertoa se koodissa. Palikan täytyi myös omistaa sprite-tyyppinen olio, jotta palikka pystyttiin piirtämään näytölle. Sprite on libgdx:n oma luokka joka pitää sisällään tiedon tekstuurista, koosta ja sijainnista ruudulla.

Alustava Palikan luokan konstruktori ja luokan sisältämä värin asetus funktio on esiteltynä alla. Konstruktori ottaa vastaan kolme parametriä, joista kaksi ovat Int-tyyppisiä muuttujia, jotka ovat kokonaislukuja. Ensimmäinen on palikan sisältämä luku alussa ja toinen kuvastaa palikan väriä, joka palikalla on. Kolmas parametri jonka konstruktori ottaa vastaan on libgdx:n sisältämän Rectangle-tyyppinen olio. Tämä olio pitää sisällään palikan x ja y koorditaatit ja myös palikan leveyden ja korkeuden pikseleissä.

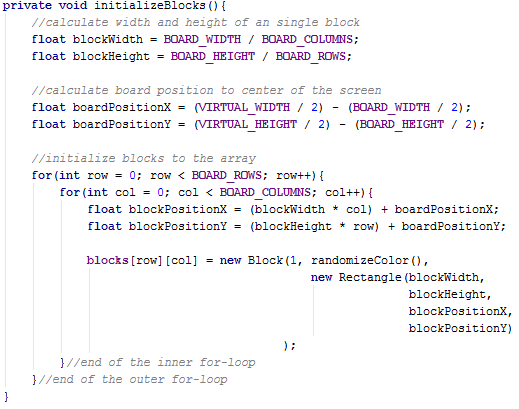


Esimerkkikoodi 3 Palikan luokan konstruktori ja luokan sisältämä setColor funktio

Muuttujien nimet jotka alkavat alaviivalla viittaavat siihen, että ne ovat luokan omistamia muuttujia. Tämä oli tiimin sisällä sovittu käytäntö, jotta koodia on helpompi lukea. Konstruktorin sisällä asetetaan ensiksi vastaanotettu Rectangle-olio luokan omaan Rectangle-olioon. Seuraavaksi asetetaan palikan lukuarvo, jonka jälkeen konstruktori käyttää luokan sisäistä funktiota setColor. Tässä funktiossa ensiksi asetetaan luokan sisäiseen muuttujaan talteen konstruktorille annettu *color.* Tämän jälkeen luodaan luokan omasta Sprite-tyyppisestä muuttujasta uusi instanssi jolle asetetaan tekstuuriksi annettua väriä vastaava kuvatiedosto, joka haetaan käyttämällä libgdx:n tarjoamaa moduulia files. Kun setColor-funktio on suoritettu niin lopuksi vielä asetetaan spriten koko ja sijainti käyttäen hyödyksi jo aikaisemmin asetettua *\_rectangle* nimistä oliota, josta saadaan koordinaatit x ja y sekä leveys ja korkeus.

Jotta palikat voitiin piirtää oikeisiin kohtiin ruudulla, täytyi ne alustaa pelin alussa jo aikaisemmin esitettyyn kaksiuloitteiseen taulukkoon. Ennen palikoiden asettamista täytyi kuitenkin määrittää, että minkä kokoinen ja missä kohtaa ruutua pelilauta sijaitsi. Tässä kohtaa täytyi määrittää haluttu virtuaalinen resoluutio pelille. Pelin virtuaaliseksi resoluutioksi valittiin 480 x 800. Jotta lauta saataisiin piirtymään ruudun keskelle, täytyi laskea hiukan matematiikkaa. Tiedettiin jo, että pelilauta koostui 24 palikasta ja ne olivat sijoitettu ruudukkoon, jonka koko oli 4x6. Jotta palikat voitiin alustaa tähän taulukkoon, ensin määrittää, että minkä kokoinen pelilaudan tulisi olla pituus- ja leveyssuunnassa. Yhden palikan koko saatiin laskettua jakamalla pelilaudan leveys neljällä ja korkeus kuudella. Palikoiden sijainti ruudulla saatiin määritettyä laskemalla ensin pelilaudan koordinaatit ruudulla, jonka jälkeen palikat voitiin alustaa taulukkoon hyödyntäen näitä tietoja ja käyttäen kahta sisäkkäistä for-silmukkaa. Alla on esitettynä

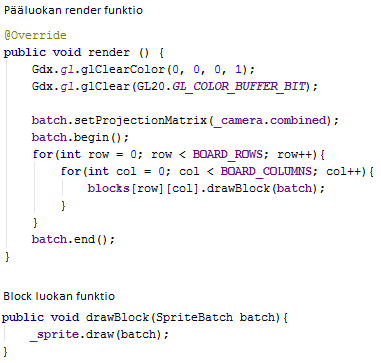
palikoiden initialisointi taulukkoon.



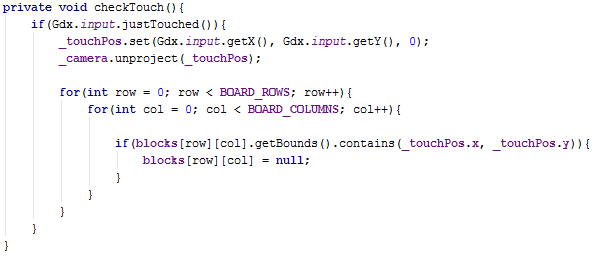
Esimerkkikoodi 4 Palikoiden alustus

Aluksi funktiossa lasketaan yksittäisen palikan leveys ja korkeus. Tämän jälkeen lasketaan laudan sijainti ruudulla. Koska peliruutu on 480 pixeliä leveä ja 800 pixeliä korkea, voidaan ruudun keskipiste selvittää helposti jakamalla nämä molemmat luvut kahdella. Ruudun keskipiste on siis kohdassa 240x400. Jotta lauta saataisiin ruudun keskelle, täytyy laudan keskipiste selvittää samalla laskutavalla ja vähentää nämä luvut ruudun keskipisteen koordinaateista. Seuraavaksi käydään kahdella sisäkkäisellä for-silmukalla koko taulukko läpi. Palikan x ja y koordinaatit saadaan laskettua hyödyntämällä silmukoiden juoksevia lukuja row ja col. Kertomalla jo aikaimmin laskettuja *blockWidth* ja *blockHeight* lukuja juoksevilla luvuilla col ja row saadaan palikka aina siirtymään jokaisella laskutoimituksella yhden palikan leveyden ja korkeuden verran ruudulla oikealle. Lisäämällä lukuihin vielä laudan sijainnin, saadaan palikoiden koordinaatit vastaamaan haluttua ruudun keskipistettä.

Seuraavaksi täytyi saada palikat piirtymään ruudulle, jotta voitiin todeta yllä olevan funktion toimivan. Piirtäminen libgdx:ssä tapahtuu käyttämällä libgdx:n omaa SpriteBatch-luokkaa. SpriteBatch hoitaa kaiken openGL:ään liittyvän toiminnallisuuden, joten siihen kehittäjän ei tarvitse puuttua ollenkaan, jossei sitä halua. Jotta palikat saatiin piirrettyä ruudulle, täyty jokainen palikka taulukossa käydä läpi ja kutsua palikan omistamaa Sprite-oliota piirtoon. Alla on esittettynä palikoiden piirto.



Esimerkkikoodi 5 Palikoiden piirtäminen

Seuraavaksi piti saada peli reagoimaan käyttäjän antamiin ruudun kosketuksiin. Libgdx:n tarjoamalla input moduulissa pystytään kysymään, että onko käyttäjä esimerkiksi koskenut ruutua tai onko käyttäjä painanut jotain näppäintä näppäimistöltään. Jotta saatiin tietää, mitä palikkaa käyttäjä oli ruudulla painanut, täytyi tehdä yksinkertainen kosketuksen tarkistus funktio. Koska jokaisella palikalla omistaa rectangle-olion, joka pitää sisällään palikan koon ja sijainnin, pystytään siltä kysymään, että sijaitseeko jokin tietty piste sen sisällä. Alla on esitettynä kosketuksen tarkistaminen.

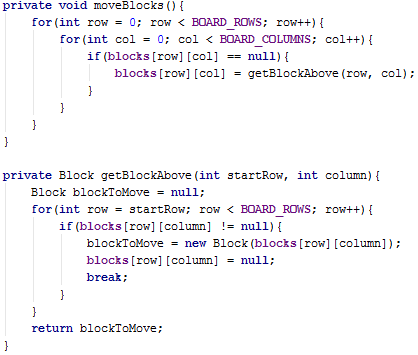
Esimerkkikoodi 6 Palikoiden kosketuksen tarkistus

Aluksi tarkastetaan, että onko pelaaja edes painanut laitteen ruutua. Tähän käytetään input moduulin tarjoamaa justTouched funktiota, joka palauttaa joko true tai false. Jos näyttöä ollaan painettu, laitetaan input moduulilta saadut x ja y koordinaatit talteen Vector3 tyyppiseen olioon \_touchPos. Vector3 sisältää kolme parametriä, jotka ovat x, y ja z. Tällaisella oliolla on siis helppo pitää sijaintia tallessa. Kun saatu sijainti on tallennettu, se annetaan pelin kameralle, joka muuttaa koordinaatit vastaamaan pelilaudan koordinaatteja. Koska input moduuli antaa x ja y arvot, mitkä vastaavat laitteen fyysisen resoluution koordinaatteja, joudutaan siksi tälläinen konversio tehdä. Seuraavaksi käydään kaikki palikat läpi taulukosta, ja kysytään palikan omistamalta Rectangle oliolta, että onko juuri saatu kosketus palikan sisällä. Jos näin on, kyseinen palikka poistetaan taulukosta.



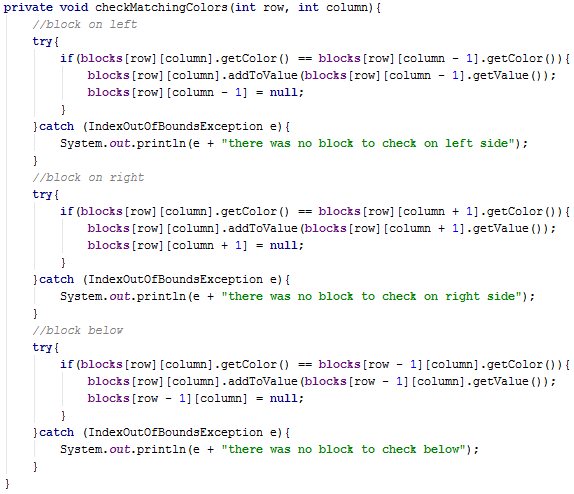
Kuva 14 SumTowerin varhaista pelikuvaa

Palikan paikan vaihtuminen pelilaudalla oli seuraavana haasteena. Koska palikat oli päätetty sijoittaa kaksiuloitteiseen taulukkoon, helpottui tämän ongelman ratkaiseminen huomattavasti. Kun palikka poistetaan pelilaudalta koskettamalla ruutua, kyseinen taulukon solu, missä palikka sijaitsi asetetaan tyhjäksi, niin kuin esimerkkikoodi 6:ssa havainnollistetaan. Jotta ylempi palikka voisi siirtyä alemmas, täytyy käydä läpi ensin kaikki solut taulukossa ja kysyä, että onko tämä solu tyhjä. Kun tyhjä solu tulee vastaan, niin voidaan tarkistaa, että onko ylemmässä solussa palikkaa. Jos ylemmästä solusta löytyy palikka, niin tämä palikka kopioidaan alempaan tyhjään soluun ja asetetaan vanha taulukon solu tyhjäksi. Alla on esitettynä palikan siirtäminen taulukossa.



Esimerkkikoodi 7 Palikoiden liikuttaminen taulukossa

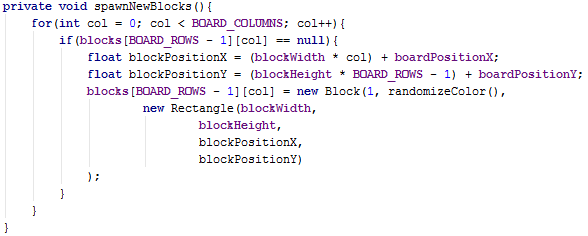
Ensimmäinen funktio moveBlocks käy läpi kaikki solut taulukosta. Kun vastaan tuleetyhjä solu, niin siirrytään käyttämään apufunktiota getBlockAbove, jonka tehtävä on etsiä ensimmäinen mahdollinen siirrettävä palikka siltä sarakkeelta, mistä alunperin löytyi tyhjä solu. Tälle funktiolle annetaan parametreinä rivi ja sarake, mistä sen kuuluu lähteä lukemaan soluja ylöspäin. Kun funktio löytää solun, missä sijaitsee palikka, niin se tekee kopion kyseisestä palikasta käyttäen Block –luokan kopiokonstruktoria, asettaa vanhan solun tyhjäksi ja hyppää ulos for –loopista käyttämällä break komentoa. Sen jälkeen se palauttaa löydetyn palikan, joka asetetaan tyhjään soluun.

Kun palikoiden siirto on tapahtunut, niin seuraavaksi tarkastetaan, sijaitseeko juuri siirtyneen palikan vieressä tai alapuolella samanvärisiä palikoita. Palikka –luokan getColor funktiota hyödyntäen tämä vertailu voidaan suorittaa. Jos viereinen tai alapuolella oleva palikka on samanvärinen, niin siirtyneen palikan arvoon lisätään jokaisen samanvärisen palikan. Alla oleva koodi havainnollistaa palikoiden yhdistämisen.

Esimerkkikoodi 8 Samanvärisien palikoiden yhdistäminen

Jokainen mahdollinen suunta käydään läpi käyttämällä try-catch metodia. Try-catch testaa halutun asian mahdollisten virheiden varalta, nappaa virheen jos sellainen sattuu ja jatkaa ohjelman suoritusta. Jos mahdollisia virheitä ei testattaisi näin, niin ohjelma kaatuisi. Syy miksi tätä metodia joudutaan käyttämään palikoiden yhdistämisessä on se, että joudumme kysymään palikoiden taulukolta, että onko esimerkiksi taulukon solussa [3][-1] olevan palikan väri sama kuin juuri taulukon soluun [3][0] siirtyneen palikan väri. Koska tiedämme että tästä tulee tapahtumaan virhe, sillä taulukossa ei ole solua [3][-1], niin varmistamme ohjelman toimivuuden ja vältämme älyttömän määrän if-lauseita käyttämällä try-catch metodia. Jokaisessa kohdassa kysytään ensin, että vastaavatko palikoiden värit toisiaan. Jos värit ovat samat, niin palikkaan lisätään vieressä tai alapuolella olevan palikan arvo käyttämällä palikka –luokan funktiota addToValue, joka ottaa vastaan lisättävän palikan arvon, joka saadaan funktiolla palikka –luokan funktiolla getValue. Tämän jälkeen palikka, joka juuri lisättiin poistetaan taulukosta asettamalle sen solu tyhjäksi.

Kun siirrot ovat tapahtuneet täytyy vielä pelilaudan ylärivi täyttää uusilla palikoilla. Laudan täyttäminen tapahtuu hyvin samalla tavalla, kuin jo aikaisemmin esitetty palikoiden alustus taulukkoon. Jos jokun taulukon ylärivin soluista on tyhjä, niin siihen asetetaan uusi palikka jonka arvo on 1 ja väri on satunnainen. Alla on esitettynä uusien palikoiden lisääminen pelilaudalle.



Esimerkkikoodi 9 Uusien palikoiden lisääminen pelilaudalle

Koodissa käydään yksinkertaisesti vain taulukon ylärivi läpi for-loopilla ja tarkastetaan onko jokin soluista tyhjä. Kun tyhjä solu löytyy, siihen asetetaan uusi palikka.

# Pohdinta

Yksi tärkeimmistä tavoitteista proketia tehdessä oli kehittää omia taitoja niin ohjelmoinnissa yleisesti kuin mobiilipelien kehityksessä. Vaikka projektissa työni oli ohjelmoida peliä, niin väistämättä tuli myös tilanteita, missä pääsi miettimään pelin designia ja pelimekaniikkoja. Myös libgdx, joka oli jo entuudestaan tuttu viitekehys avasi itsestään uusia puolia, joita pystyy käyttämään hyödyksi tulevissa projekteissa.

Ohjelmointitaitoni kehittyivät omasta mielestäni huomattavasti projektin aikana. Kun projekti aloitettiin, en ollut kovin varma omasta osaamisestani, sillä tämä projekti oli suurin, mitä koskaan olin ennen tehnyt. Ongelmanratkaisukykyni kehittyi huomattavasti ja omasta koodistani tuli tehokkaampaa. Tätä raporttiakin kirjoittaessa ja koodia läpikäydessä huomasin siellä asioita, mitä voisi tehdä paremmin. Yksi näistä huomioista on esimerkkikoodissa 7 esitetty tapa tarkastaa viereisiä palikoita. Aikaisemmin se oli toteutettu tekemällä monta peräkkäistä if –kyselyä, mutta kun koodia silmäili uudelleen, niin huomasin, että on tehokkaampikin tapa toteuttaa se. Myös taitoni lukea koodia on kehittynyt huomattavasti projektin aikana. Tässä projektissa on monia tuhansia rivejä koodia ja koska se kaikki ei ole itse kirjoitettua, niin täytyy osta lukea, että mitä jonkun muun tekemässä ongelman ratkaisussa tapahtuu.

Yhdeksi tärkeäksi kehityskohteeksi asetin itselleni projektin kokonaisuutena näkemisen ja sen pilkkomisen pienempiin osiin. Tässä on omasta mielestäni vielä kehityksen varaa, sillä välillä projektia tehdessä huomasi, että oli alkanut miettimään jotain ongelman ratkaisua liian laajasti. Loppujen lopuksi huomasi, että jos olisi ratkaissut ensin pienen osan ongelmaa, olisi kokonaisuuden ratkaisu helpottunut huomattavasti. Kuitenkin kehitystä tällä saralla tuli kiitettävästi.

Tiedonhankintataidot ovat kehittyneet huomattavasti. Kaikkia ongelmia kun ei itse voi ratkaista, niin välillä on turvauduttava tiedon etsintään ja katsoa, että onko joku toinen paininut samankaltaisen ongelman kanssa. Projektia tehdessä tuli vastaan monta uutta hyvää internet sivustoa, joista löytyi apua ongelmatilanteissa. Internet on siis hyvin tärkeä työväline ohjelmoinnissa ja se, että pystyy silmämääräisesti suodattamaan sen turhan tiedon, on hyvä oppia, koska muuten voi mennä hyvin paljon turhaa aikaa hukkaan jossain ongelmassa.

Mobiilipelien ohjelmoinnissa on omalta osaltani tapahtunut suurin kehitys. Se miten itse sen huomaan, on kun pelaan muiden tekemiä pelejä, mitä esimerkiksi olen ladannut omaan älypuhelimeeni, niin samalla pelatessani mietin, että miten joku pelistä löytyvä ominaisuus on tehty ja saatan jopa tehdä jonkun pienen testiprojektin, jossa koitan tuottaa tälläisen samanlaisen featuren. Kiinnostus alaa kohtaan on kasvanut projektin myötä huomattavasti.

Lähteet

1. Mashable. 2016. Mobile Games. Verkkodokumentti. <http://mashable.com/category/mobile-games/> Luettu 5.2.2016
2. Phone Arena. 2011. History of mobile gaming. Verkkodokumentti. <http://www.phonearena.com/news/History-of-mobile-gaming\_id17949> Luettu 15.1.2016
3. Java World. 2003. Develop state-of-the-art mobile games. Verkkodokumentti. <http://www.javaworld.com/article/2073796/mobile-java/mobile-java-develop-state-of-the-art-mobile-games.html> 7.11.2003. Luettu 3.2.2015.
4. Tom Warren. 2014. iPhone: A visual history. Verkkodokumentti. <http://www.theverge.com/2014/9/9/6125849/iphone-history-pictures>. Luettu 8.2.2016.
5. Cnet. 2015. Modern Combat 5: Blackout review. Verkkodokumentti. <http://www.cnet.com/products/modern-combat-5-blackout/>. Luettu 24.2.2016.
6. Superdata. 2015. Mobile games market. Verkkodokumentti. <https://www.superdataresearch.com/market-data/mobile-games-market/>. Luettu 26.2.2016.
7. Jesse Freeman. 2015. How to make a game part 1: Picking a framework. <http://jessefreeman.com/game-dev/getting-started-making-games-part-1-picking-framework/>. Luettu 26.2.2016.
8. Cocos2d-x. 2015. COCOS2D-X. Verkkodokumentti. < http://www.cocos2d-x.org/wiki/Cocos2d-x>. Luettu 1.3.2016.
9. Java. 2015. Learn About Java Technology. Verkkodokumentti. <http://java.com/en/about/>. Luettu 15.2.2016.
10. Oracle. 2015. About the Java Technology. Verkkodokumentti. <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html#FOOT>. Luettu 15.2.2016
11. Webopedia. 2015. OOP - Object Oriented Programming Definition. <http://www.webopedia.com/TERM/O/object\_oriented\_programming\_OOP.html>. Verkkodokumentti. Luettu 18.2.2016.
12. Craig, Iain D. 2007. Object-oriented programming languages : interpretation. London : Springer
13. Android developers. 2015. Android Studio Overview. Verkkodokumentti. <http://developer.android.com/tools/studio/index.html> Luettu 2.1.2016.
14. Github. 2015. Libgdx. Introduction. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Introduction> 29.7.2015. Luettu 2.1.2016.
15. Techopedia. 2015. Native code. Verkkodokumentti. <https://www.techopedia.com/definition/3846/native-code> Luettu 2.1.2016.
16. Github. 2015. Libgdx. The life cycle. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-life-cycle> Luettu 14.1.2016
17. Github. 2015. Libgdx. The application framework. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-application-framework> Luettu 4.1.2016
18. Chacon, Scott & Straub, Ben. 2014. Pro Git. Apress.

**Android- ja työpöytäsovelluksen sekä pääohjelman käynnistysluokat**

**Liitteen otsikko**