

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](#_Toc444246920)

[2 Mobiilipelit 1](#_Toc444246921)

[3 Mobiilipelikehitys 4](#_Toc444246922)

[3.1 Unity 5](#_Toc444246923)

[3.2 Cocos2D 5](#_Toc444246924)

[4 Java 5](#_Toc444246925)

[4.1 Olio-ohjelmointi 6](#_Toc444246926)

[5 Insinöörityössä käytetyt työkalut 8](#_Toc444246927)

[5.1 Android studio 8](#_Toc444246928)

[5.2 LibGDX viitekehys 9](#_Toc444246929)

[5.3 Git-versionhallintajärjestelmä 12](#_Toc444246930)

[5.4 Asana –projektinhallintajärjestelmä 15](#_Toc444246931)

[6 SumTower 15](#_Toc444246932)

[6.1 Pelimuodot 15](#_Toc444246933)

[7 Pelin toteutus 15](#_Toc444246934)

[7.1 Projektin luonti 15](#_Toc444246935)

[7.2 Pelimekaniikka 15](#_Toc444246936)

[7.3 Käyttöliittymä 15](#_Toc444246937)

[8 Pohdinta 15](#_Toc444246938)

[Lähteet 16](#_Toc444246939)

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

# 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.



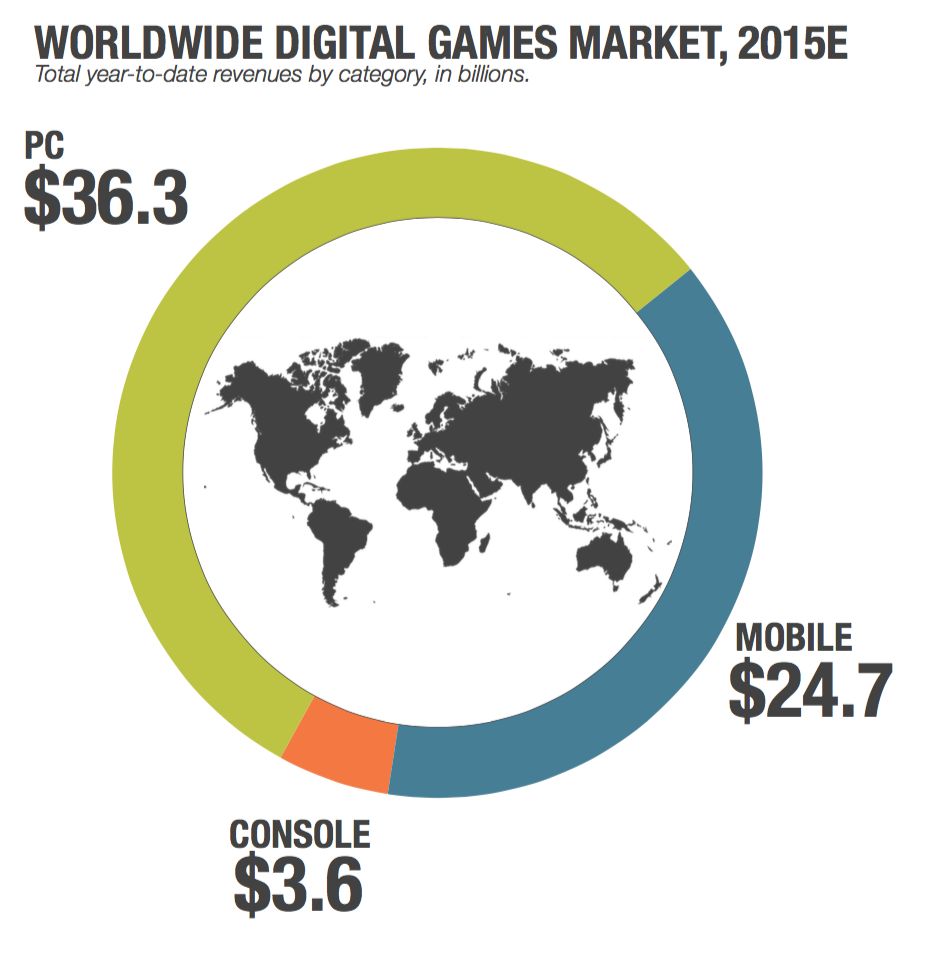
Kuva 3. Angry Birds -pelin ohjekuva

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.

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 ja pelimottori.

## 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 cross-platform 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

Cocos2D on avoimeen lähdekoodiin perustuva cross-platform sovelluskehys, jolla voidaan tehdä näyttäviä 2D-pelejä mobiilille ja PC:lle. Cocos2D tukee kahta eri ohjelmointikieltä C++ ja JavaScriptä.

# Java

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.

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. **[8.]**

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. **[9.]**



Kuva 6 java-ohjelman koontiprosessin havainnollistaminen [9.]

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. **[9.]**

## 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 **[10.]**

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. **[11, 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.



Kuva 7 Luokan rakenteen, olioiden luomisen ja käyttämisen havainnollistaminen

Kuvassa 5 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. **[12.]**

**//TODO: lisää asiaa android studiosta**

## 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. **[13.]**

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. **[13.]**

Libgdx tekee tarvittaessa kaikki natiiviin koodiin liittyvät toiminnot itsenäisesti ja ne suoritetaan yleensä koodin käännösvaiheessa, jolloin kehittäjän ei tästä tarvitse välittää **[13].** Natiivilla koodilla tarkoitetaan koodia, joka on suunniteltu ajettavaksi määrätynlaisella prosessorityypillä. Natiivia koodia ei siis pysty ajamaan muilla prosessoreilla ellei koodia ole tarkoitettu emuloitavaksi. **[14.]**

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.**[15.]** 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 [15].

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. **[16.]** 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. **[16.]** Seuraavassa kuvassa havainnollistan kuinka näihin moduuleihin pääsee käsiksi koodissa.

Esimerkkikoodi 1 Moduulien alustus

**//TODO: lisää asiaa libGDX:stä**

## 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. **[17, 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. **[17, 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. **[17, 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. **[17, s.9]**

//kuva gitin komentorivistä

## Asana –projektinhallintajärjestelmä

# SumTower

## Pelimuodot

# Pelin toteutus

## Projektin luonti

## Pelimekaniikka

## Käyttöliittymä

# Pohdinta

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. Java. 2015. Learn About Java Technology. Verkkodokumentti. <http://java.com/en/about/>. Luettu 15.2.2016.
9. Oracle. 2015. About the Java Technology. Verkkodokumentti. <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html#FOOT>. Luettu 15.2.2016
10. Webopedia. 2015. OOP - Object Oriented Programming Definition. <http://www.webopedia.com/TERM/O/object\_oriented\_programming\_OOP.html>. Verkkodokumentti. Luettu 18.2.2016.
11. Craig, Iain D. 2007. Object-oriented programming languages : interpretation. London : Springer
12. Android developers. 2015. Android Studio Overview. Verkkodokumentti. <http://developer.android.com/tools/studio/index.html> Luettu 2.1.2016.
13. Github. 2015. Libgdx. Introduction. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Introduction> 29.7.2015. Luettu 2.1.2016.
14. Techopedia. 2015. Native code. Verkkodokumentti. <https://www.techopedia.com/definition/3846/native-code> Luettu 2.1.2016.
15. Github. 2015. Libgdx. The life cycle. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-life-cycle> Luettu 14.1.2016
16. Github. 2015. Libgdx. The application framework. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-application-framework> Luettu 4.1.2016
17. Chacon, Scott & Straub, Ben. 2014. Pro Git. Apress.

**Android- ja työpöytäsovelluksen sekä pääohjelman käynnistysluokat**

**Liitteen otsikko**