Aro Laaksonen

MOBIILIPELIN KEHITTÄMINEN ANDROIDILLE

Tietojenkäsittelyn koulutusohjelma

2017

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

MOBIILIPELIN KEHITTÄMINEN ANDROIDILLE

Laaksonen, Aro
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Toukokuu 2018
Ohjaaja: Nuutinen, Petri
Sivumäärä: 39
Liitteitä: 0

Asiasanat: Mobiilipelit, peliohjelmointi, pelisuunnittelu

Tämän opinnäytetyön toiminnallisena osana kehitettiin mobiilipeli Androidille. Pelissä lentoalus lentää läpi kaupungin, ja pelaajan tarkoituksena on muuttaa aluksen väriä eteen ilmestyviä esteitä vastaaviksi päästäkseen mahdollisimman pitkälle.

Työn kirjallinen osa on jaettu kahteen osaan: käytettyjen tekniikoiden tarkasteluun sekä lopullisen tuotteen esittelyyn. Käytettyjen tekniikoiden tutkiminen aloitettiin Android-käyttöjärjestelmän esittelyllä, mistä jatkettiin sen rakenteen selostamiseen.

Tämän jälkeen selitettiin lyhyesti mikä on mobiilipeli, sekä tarkasteltiin pelinkehitystä ja sen vaiheita. Vaiheet eroteltiin kolmeen osaan, joita löyhästi seuraten myös tämän opinnäytetyön toiminnallinen osa on tehty.

Seuraavaksi esiteltiin lyhyesti Unity-pelimoottori sekä C#-ohjelmointikieli, joita on toiminnallisessa osassa käytetty. Tutustuttiin C#:n historiaan sekä sen perusajatuksiin yksinkertaisen esimerkkikoodin muodossa.

Käytännön osassa annettiin kuvaus pelistä kokonaisuutena, sekä selostettiin pelin kehittämisen esituotanto- ja tuotantovaiheet. Tuotantovaiheen kuvauksessa eroteltiin pelin eri komponentit ja esiteltiin niiden toimintaa.

DEVELOPING A MOBILE GAME FOR ANDROID

Laaksonen, Aro
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Data Management
May 2018
Supervisor: Nuutinen, Petri
Number of pages: 39
Appendices: 0

Keywords: Mobile games, game programming, game design

As the functional part of this thesis, an Android mobile game was developed. In the game an aircraft flies through a city, with the player's goal being changing its color to match that of the obstacles spawning in front of it and thus getting as far as possible.

The written part of this thesis is divided into two parts: the first examines the techniques and technologies used and the second presents the final product. Examining the used techniques began with an introduction to the Android operating system, followed by an explanation of its architecture.

After that came a brief explanation of a mobile game as a concept, and an examination of game development and its stages. Three stages are identified, and those were loosely followed in the functional part of this thesis.

Next the Unity game engine and the C# programming language were presented, both of which are used in the functional part of this thesis. The C# language's history is observed and also its fundamental ideas through a simple piece of code.

The second half of the written part of this thesis gives an explanation of the game, and its preproduction and production stages are explained. In production stage's description the different components of the game separated and their operations are shown.

SISÄLLYS

1 JOHDANTO

Pelit ovat aina kiinnostaneet minua. Ne yhdistelevät teknologian, viihteen ja taiteen eri puolia tavoilla, joihin muut mediat eivät pysty. Peli voi haastaa, herättää tunteita, viihdyttää ja jopa opettaa. Pelaajan ja pelin välinen interaktio luo ainutlaatuisia kokemuksia.

Juuri sellaiset kokemukset saivat minut kiinnostumaan myös pelien tekemisestä, mutta läpi peruskoulun ja vielä lukionkin vain leikittelin ajatuksella, että seuraisin tuota kiinnostusta. Armeijan jälkeen puntaroin useiden vaihtoehtojen väliltä ja lopulta valitsin alan, jonka opinnoista saisin tarvitsemani työkalut lapsuuden haaveen toteuttamiseksi.

Colour Rush on ensimmäinen alusta loppuun saakka kehittämäni peli. Eri tekniikoihin olin tutustunut jo kauan aikaa sitten, ja myös eri ideoita olin pyöritellyt päässäni ja editorissa jo pitkän aikaa, mutta ennen Colour Rushin perusajatusta minulla ei ollut ollut sellaista ideaa, jonka olisin kokenut toteuttamisen arvoiseksi.

Pelin kehittäminen opetti varsinkin projektin näkemisestä kokonaisuutena. On totta, että hyvä perusidea luo vankan ytimen lopulliselle tuotteelle, mutta myös kaiken sen ympärille rakentuvan täytyy olla toimivaa. Peliä tehdessä ideoita lisäominaisuuksista syntyi paljon, jolloin itsekritiikki ja taito erotella niin sanotusti jyvät akanoista tulivat tarpeeseen. Selkeä, jopa hieman pelkistetty design oli tavoitteeni, koska silloin keskiössä on mielestäni hyvin toimiva pelattavuus, eikä yksikään ominaisuuksista tunnu turhalta.

2 KÄYTETYT TEKNIIKAT

2.1 Android

Android on Googlen kehittämä, puhelimille ja muille mobiililaitteille tarkoitettu käyttöjärjestelmä. Se on vuoden 2007 julkaisunsa jälkeen noussut maailman myydyimmäksi mobiilialustaksi ja sen markkinaosuus kaikista maailman älypuhelimista oli vuonna 2017 85% (idc.com 2017). Vuonna 2017 Google ilmoitti Androidilla olevan 2 miljardia vähintään kuukausittain aktiivista käyttäjää ja sen sovellusjakelualustassa Google Playssä olevan yli 2,7 miljoonaa sovellusta (AppBrain 2017; Protalinski 2017). Yhtenä Androidin vahvuuksista pidetään sen avointa lähdekoodia, jonka ansiosta monet laitevalmistajat kuten Amazon ja Nokia ovat voineet julkaista omia laitteitaan jotka hyödyntävät valmistajan omaa, mukautettua versiota Androidista (Business Insider 2014).

Muokattavuus ja lisensointi ovat kuitenkin aiheuttaneet myös pirstaloitumista. Koska käyttöjärjestelmästä on olemassa monia eri versioita, on äärimmäisen vaikea kehittää sovelluksia, jotka toimisivat kaikissa sitä käyttävissä laitteissa. Pirstaloituminen johtuu osaltaan siitä, että suuri osa Android-laitteista koostuu halvoista, vanhoja käyttöjärjestelmäversioita käyttävistä puhelimista. Kehittäjän on otettava tämä osuus huomioon kattaakseen tietyn määrän kaikista Android-käyttäjistä, hidastaen näin uusien laite- ja ohjelmisto-ominaisuuksien hyödyntämistä. (Kovach 2013)

Androidin uusin versio 8, ”Oreo” julkaistiin elokuussa 2017 ja sitä käyttää 0,5% kaikista Android-käyttäjistä. Eniten käyttäjiä on versiolla 7 ”Nougat”, noin 21,1%. (Android 2017).

2.1.1 Linux-ydin

Android rakentuu Linux-ytimen (kernel) päälle, joka tarjoaa abstraktiotason sovellusten ja laitteiston välille. Kernel tulkitsee ohjelmiston input- ja output-pyyntöjä laitteistolle käyttäen esimerkiksi laitteen kameraa, näyttöä tai muistia (Hildenbrand 2012).

Suurin osa Android-laitteista käyttää Linux-kernelin versioita 3.18 tai 4.4, mutta vaihtelu on hyvinkin laitekohtaista (Amadeo 2017).

Androidille Linux-kernelin tärkeimpiä ominaisuuksia ovat:

- Muistinhallinta: Mobiilisovellusten kehityksessä muistinhallinta on tärkeässä osassa, ja kernelin tehtävä on helpottaa tätä. Se hallinnoi muistia allokoimalla ja deallokoimalla sitä tiedostojärjestelmän, eri prosessien ja sovellusten välillä.
- Prosessihallinta: Kernel on vastuussa ohjelmien käynnistyksestä ja lopetuksesta, sekä tarvittavien resurssien antamisesta niitä tarvitsevien prosessien käyttöön.
- Ajurit: Esimerkiksi näytön ja kameran hallinnointi.
- Tiedostonhallinta: Kernel ohjaa tiedostojärjestelmää joka puolestaan vastaa laitteen tallennustilasta.
- Verkko: Kernel on myös vastuussa järjestelmän verkkoyhteydestä ja sen käytöstä.
- Käyttäjähallinta: Kernel huolehtii sovelluksen ja järjestelmän välisestä turvallisuudesta. Se hallinnoi käyttäjiä ja niiden autentikointia.

(Zinoune 2012)

Androidin käyttämä muunnelma Linux-kernelistä sisältää myös joitain Googlen tekemiä muutoksia. Näitä ovat esimerkiksi ashmem; (Android Shared Memory) muistin jakamista ja säästämistä helpottava komponentti ja Viking Killer, joka muistin ollessa vähissä lopettaa prosessin, jonka käytöstä on kulunut eniten aikaa. (Love 2013)

2.1.2 Sovelluskerros

Linux-kernelin päälle rakentuu Androidin sovelluskerros. Se sisältää C-kielellä tehtyjä rajapintoja, kirjastoja, väliohjelmia, ohjelmistokehyksen sekä sovellukset (Android 2017). Versioon 5.0 asti sovellukset suoritettiin Dalvik-virtuaalikoneella (Burnette 2008).

Dalvik kääntää java-tiedostoista tehdyt dex-tiedostot dynaamisesti konekielelle, mikä säästää varsinkin vanhempien Android-laitteiden rajallista muistin määrää. Androidin versiossa 4.4 esiteltiin Android Runtime (ART) uutena vaihtoehtoisena suoritusympäristönä. Siinä missä Dalvik kääntää koodia useassa pienessä osassa (JIT, just in time), ART kääntää dex-tiedostot konekielellä ennen suorittamista (AOT, ahead of time). Kääntäminen suoritetaan jo sovelluksen asennusvaiheessa, jonka jälkeen ART tallentaa konekielisen koodin laitteen muistiin. Tämä vähentää suorittimen kuormitusta ja laitteen virrankulutusta. (Sinhal 2017) ART:sta tuli Androidin ainoa suoritusympäristö versiossa 5.0 (Toombs 2013).

Androidin sovelluskehys tarjoaa useita korkeamman tason toimintoja sovelluksille java-luokkien muodossa. Näitä ovat muun muassa:

- View System: kokoelma näkymiä käyttöliittymän luomiseen
- Notifications Manager: käyttäjälle osoitettujen ilmoitusten näyttäminen
- Resource manager: sovelluksen eri resurssien kuten kuvien, käyttöliittymän ja arvojen käyttö
- Content Provider: datan julkaisu ja jako muiden sovellusten kanssa.
- Activity Manager: näkymien ja sovelluksen elinkaarien hallinnointi

2.2 Mobiilipeli

Mobiilipeli on videopeli, jota pelataan puhelimella, tabletilla, älykellolla tai muulla vastaavalla kannattavalla laitteella. Niitä ei pidä sekoittaa käsikonsolipeleihin, joita pelataan nimenomaan pelaamiseen tarkoitetuilla laitteilla. Mobiilipelaamisen alkuaikoina pelit olivat yleensä laitteissa esiasennettuna, mutta nykyään ne usein ladataan sovelluskaupoista. Sovelluskaupat ovat yleensä alustakohtaisia, kuten Androidilla Google Play ja Applen iOS:n App Store.

Vuonna 2017 pelimarkkinoiden arvon arvioitiin nousevan 108,9 miljardiin dollariin vuoden loppuun mennessä. Mobiilipelien osuuden arvioitiin olevan noin 42% ja sen arvioitiin kasvavan yli puoleen pelimarkkinoiden kokonaisarvosta vuoteen 2020 mennessä (Brightman 2017).

Perinteisesti varsin rajallisia resursseja käyttävät mobiililaitteet ovat rajoittaneet niillä pelattavien pelien mittasuhteita. Monet mobiilipelit painottavatkin näin ollen pelaamisen helppoutta ja innovatiivista suunnittelua. (Technopedia)

2.3 Unity

Unity on usealle alustalle suunnattu pelimoottori ja editori, jolla voidaan kehittää kaksi- ja kolmiulotteisia pelejä ja simulaatioita (Unity). Sitä kehittävä Unity Technologies perustettiin vuonna 2004 ja pelimoottori julkaistiin vuonna 2005 (Takahashi 2014; Brodkin 2013). Unity tukee 27:ää eri alustaa, merkittävimpinä esimerkiksi Windows, Mac, iOS, Android, PlayStation 4 ja Xbox One (Unity).

Unity on kehityskaarensa aikana tukenut skriptikielinään JavaScriptiä, Boota ja C#:a. Tuki Boolle kuitenkin lopetettiin Unity 5:n julkaisun myötä vuonna 2015 ja prosessi JavaScriptin poistamiseksi käytöstä aloitettiin elokuussa 2017, Unity 2017.1:n julkaisun myötä (Unity; Fine 2017). Tämän opinnäytetyön käytännön osuus on toteutettu C#:lla.

Unitylla on mahdollista nopeasti ja tehokkaasti luoda objekteja, tuoda ulkoisia materiaaleja kuten grafiikkaa ja ääntä sekä yhdistää nämä koodilla. Usean alustan tuen lisäksi Unity tarjoaa skriptausrajapinnan ja sisäänrakennetun mahdollisuuden verkko-ominaisuuksien käyttöön. (Menard 2011, 15)

Unity on kehitysympäristönä suosittu. Tämän syiksi voidaan eritellä useita sen ominaisuuksia. Graafinen editori tekee pelin suunnittelemisesta toteuttamisesta helppoa. Objektien, skriptien ja vaikkapa valoefektien toteuttaminen ja muokkaus on yksinkertaista, nopeuttaen kehitysprosessia huomattavasti. Tämä helppous auttaa kehittäjää siirtymään Unityn käyttämiseen, madaltamalla oppimiskäyrää. Lisäksi Unitylla on kattava ja hyvä dokumentaatio, sekä useita opetusvideoita ja harjoituksia on saatavilla. Pelin kääntäminen eri alustoille yksinkertaista ja halutun alustan valitsemisen jälkeen tapahtuu yhtä nappia painamalla. (Shewaramani 2015)

Unity Asset Store on kauppa, jonka avulla on mahdollista nopeuttaa pelin kehittämistä entisestään. Sieltä on mahdollista ostaa mitä tahansa grafiikoiden, valmiiden skriptien ja editorilisäosien väliltä. (Shewaramani 2015)

2.4 C#

C# (C sharp) on Microsoftin kehittämä, vuonna 2000 julkaistu oliopohjainen ohjelmointikieli. .NET Framework-ohjelmistokehyksen kehityksen yhteydessä todettiin, että Microsoftin sen aikaiset kehittäjän vaihtoehdot olivat varsin sirpaloituneet. Natiivia koodia kirjoitettiin C++:lla, nopean kehityksen mallia toteutettiin Visual Basicilla ja verkkoympäristössä käytössä olivat IIS ja ASP. Jokainen kieli tarjosi omanlaisiaan ratkaisuja eri ohjelmointikysymyksiin. Kehittäjän oli vaikea siirtää taitoaan ympäristöstä toiseen, sillä ympäristö määritteli vahvasti myös käytössä olevan ohjelmointikielen. Uuden, alun perin COOLiksi (C-like Object Oriented Language) kutsutun kielen tavoitteena olikin yhdistää nämä ympäristöt. Vuonna 1999 Anders Hejlsberg perusti työryhmän kehittämään C#:a. (Hamilton 2008)

Kielelle ja sen kehitykselle asetettiin seuraavat tavoitteet (ECMA International 2006):

- Kielen on tarkoitus olla helppo, moderni, yleiskäyttöinen ja oliopohjainen
- Kielen tulee tukea ohjelmistokehityksen yleisiä periaatteita: tyyppien ja taulukkojen tarkistus, alustamattomien muuttujien käytön havaitseminen ja automaattinen roskienkeräys. Ohjelmiston vikasietoisuus, pitkä käyttöikä sekä kehittäjän tuottavuus ovat tärkeitä.
- Kieli on tarkoitettu hajautetuissa ympäristöissä käytettävien ohjelmistokomponenttien kehitykseen.
- Koodin siirrettävyys sekä kehittäjän siirrettävyys muista kielistä (etenkin C:stä ja C++:sta) on tärkeää.
- Tuki kansainväliselle käytölle on tärkeää
- C#:n on tarkoitus soveltua monenlaisiin käyttöympäristöihin, laajoista käyttöjärjestelmistä aina pienempiin, erikoistuneisiin järjestelmiin.

- Vaikka kieli on suunniteltu käyttämään laitteiston resursseja taloudellisesti, sen ei ole tarkoitus kilpailla C- tai assemblykielen kanssa tehokkuudessa tai koossa.

2.4.1 Kuvaus kielestä

C#-koodi muistuttaa paljon C++:aa ja Javaa. Koodi koostuu koodilohkoista, jotka merkitään aaltosulkeilla. Koodilohkojen sisälle kirjoitetaan lauseita, joiden loppu merkitään puolipisteillä. Lauseiden enimmäismäärää lohkossa ei ole rajoitettu, ja lohko voi olla myös tyhjä lauseista. (Watson, Nagel, Pedersen, Reid, Skinner 2010, 32)

C# on kielenä vahvasti tyypitetty. Tämä tarkoittaa sitä, että muuttujaa esiteltäessä sille on annettava tietotyyppi. C# sisältää useita valmiita perustietotyyppejä, mutta se tukee myös käyttäjän luomia tyyppejä, kuten luokkia. Tyypit voidaan jakaa arvotyyppeihin (value types) sekä viittaustyyppeihin (reference types). Arvotyyppiselle muuttujalle voidaan asettaa arvo suoraan. Näihin kuuluvat esimerkiksi numeeriset perustietotyypit kuten int ja float. Viittaustyyppisten muuttujien arvoa täytyy käsitellä metodien ja funktioiden kautta. (Telles 2001, 2)

Kaikki C#-sovellukset perustuvat luokkiin. C-kielestä ja C++:sta poiketen C#:ssa ei voi olla yksittäistä itsekseen toimivaa funktiota. Luokkiin perustuva rakenne muistuttaakin enemmän Javaa. Jokainen C#-sovellus tarvitsee myös staattisen metodin nimeltä Main, joka toimii sovelluksen suorittamisen aloituspisteenä. (Telles 2001, 5)

```
Class Esimerkki
{
    static void Main()
    {
        System.Console.WriteLine("Hello world!");
    }
}
```

Koodi 1. ”Hello world!”-esimerkki C#-kielellä.

Koodissa 1 on yksinkertainen C#-ohjelma. Main-metodi sijaitsee luokassa Esimerkki. Ohjelmaa suoritettaessa System-nimiavaruudesta haetaan Console-olio, jonka staattiselle metodille WriteLine annetaan merkkijono, joka halutaan tulostettavan konsoliin.

C# on kielenä saavuttanut useita sille asetettua tavoitetta. Helppokäyttöisyys on yksi niistä. Javasta, C-kielestä ja jopa JavaScriptistä siirtyvän kehittäjän on helppo ottaa kieli käyttöön, sillä sen syntaksi muistuttaa paljon edellä mainittuja kieliä ja on helppolukuista. Kieli on myös saavuttanut monipuolisuuteen liittyvän tavoitteensa, soveltuu monenlaisiin ympäristöihin. Sitä käytetään yleisesti työpöytäsovelluksissa, verkkosovelluksissa ja jopa mobiilisovelluksissa. Monipuolisuus on kielelle suuri etu juuri siksi että siinä saavutettu taito on helposti siirrettävissä eri ympäristöihin. (Watson 2017).

3 PELINKEHITYS

Pelinkehitys on kokoelma prosesseja ja vaiheita, joiden on lopulta tarkoitus luoda peli. Pelien tarkoitus on toimia luovana ilmaisun keinona, mutta myös tuottaa rahaa. Laadukkaat, hyvin tehdyt pelit tuottavat paremmin (Bethke 2003, 7, 12, 14). Pelin kehitystiimi voi olla kooltaan mitä tahansa yksittäisen kehittäjän ja monikansallisen projektiryhmän väliltä. Pelistudio voi olla julkaisijan omistama tai itsenäinen (McGuire & Jenkins 2009, 25). Itsenäiset studiot toimivat myös julkaisijan varassa. Ne usein kehittävät pelin pitkällekin itsenäisesti, ja sen ollessa esittelyvalmis yrittävät löytää sille julkaisijan (Chandler 2009, 82). Julkaisija saa yleensä yksinoikeudet pelin levitykseen ja usein omistaa myös mahdollisen pelisarjan immateriaalioikeudet (McGuire & Jenkins 2009, 25).

Pelin kehittäminen on mahdollista muutamassa kuukaudessa, osa peleistä on vuosia työn alla. Huolimatta siitä kuinka pitkään projekti kestää, on peliteollisuudelle muodostunut vakiintuneita tapoja, miten strukturoida pelin kehitysprosessia.

3.1 Tiimin roolit

Teoriassa pelin tekemiseen riittää ohjelmoija ja taiteilija. Ison, menestystä hakevan pelin tekemiseen tarvitaan kuitenkin myös muita taitoja. (Novak 2012, 319)

Tuottaja on henkilö, joka pitää asiat liikkeessä. Tuottajat ovat vastuussa siitä, että peli saadaan julkaistua ajoissa ja budjetin puitteissa sekä siitä, että kaikki tiimin jäsenet tekevät sitä mitä pitääkin. Tärkeä, joskin usein vähälle huomiolle jäävä vastuualue on tuottajan kyky hallinnoida ihmisiä konfliktinratkonnan, kommunikoinnin ja yhteisymmärryksen rakentamisen keinoin. Näiden taitojen puute voi muodostua suureksi ongelmaksi projektille, joka johtaa tiimin jäsenten työmoraalin laskuun ja näiden uupumukseen. Tuottajan tehtävä on saattaa käytettävissä oleva aika, raha ja laatu tasapainoon. Ulkoinen tuottaja toimii yhteyshenkilönä pelikehittäjän ja julkaisijan välillä, sisäinen tuottaja johtaa itse kehitystiimiä. (Novak 2012, 319)

Suunnittelijat keskittyvät pelattavuuden, kenttien ja käyttöliittymien suunnitteluun. Esimerkiksi luova ohjaaja (creative director) varmistaa että pelin tyyli ja sisältö ovat yhtenäisiä. Pääsuunnittelija (lead designer) johtaa suunnittelutiimiä ja osallistuu usein myös itse varsinaiseen suunnittelutyöhön. Tarinasuunnittelija (narrative designer) on henkilö, jolla voi olla kirjoitustaustaa muista medioista kuten televisiosta tai elokuvista. Tarinasuunnittelijan tehtävä on ymmärtää pelien eroavaisuudet muihin medioihin nähden ja tuoda peliin siihen sopivaa kerrontaa. Muita suunnittelijoita ovat käyttöliittymä- sekä kenttäsuunnittelijat. (Novak 2012, 323)

Taiteilijat vastaaja konseptitaiteen ja pelin lopullisen grafiikan luomisesta. Pelitaiteen osa-alueita ovat piirtäminen, mallintaminen, tekstuurien teko sekä animaatio. Taiteilija voi olla erikoistunut myös tietynlaisten kohteiden tai asioiden tekemiseen kuten hahmoihin, ympäristöihin tai vaikkapa erikoistehosteisiin. Taiteilijat voidaan jakaa myös sen mukaan, että tekevätkö he kaksi- vai kolmiulotteista grafiikkaa. (Novak 2012, 329)

Peliohjelmointiin voi sisältyä mitä tahansa pelimoottorin tai tietokannan luomisen ja grafiikoiden tai äänen ohjelmoinnin väliltä. Näillä kullakin osa-alueella on niihin erikoistuneet ohjelmoijat. Tekninen johtaja (technical director) pitää huolen projektin

teknisestä puolesta sekä määrittää käytettävät työkalut ja menetelmät. Johtava ohjelmoija (lead programmer) johtaa ohjelmointitiimiä ja usein osallistuu myös itse ohjelmointityöhön. (Novak 2012, 332)

Pelin äänet jaetaan yleensä musiikkiin, äänitehosteisiin ja dialogiin. On mahdollista, että pelistudiolla ei ole omaa ääniosastoa, jolloin äänipalvelut ostetaan joltain muulta osapuolelta. Ääniohjaaja (audio director) johtaa ääniosastoa ja on tiiviissä yhteistyössä ääniohjelmoijan kanssa. Ääniohjaaja myös yleensä vastaa muiden ääneen liittyvien tehtävien kuten säveltäjien, äänisuunnittelijoiden ja ääninäyttelijöiden valitsemisesta. (Novak 2012, 338)

Testaus- ja laadunhallintatiimi pitää huolen siitä, että peli testataan perinpohjaisesti ennen julkaisua ja että peli on toimiva, mahdollisimman bugiton, yhtenäinen ja viihdyttävä. Testausjohtaja (testing manager) hallinnoi koko testaus- ja laadunhallintaprojektia. Omia tiimejään ohjaavat johtavat testaajat (lead tester) raportoivat testausjohtajalle. Testauksen osa-alueita ovat yhteensopivuustestaus, laatutestaus, pelattavuustestaus ja käyttettävyystestaus. (Novak 2012, 339)

3.2 Pelin suunnittelun periaatteita

Pelinkehittäjän tulee tietää mitä pelaajan mielessä liikkuu milläkin hetkellä. Tämä empatiakyky on ehdottoman tärkeä, sillä pelin suunnittelijan tulee voida kuvitella itsensä pelaajan asemaan ennustaakseen tämän reaktioita pelissä esiintyviin asioihin. On kyettävä ajattelemaan pelaajan lailla ja osattava aavistaa mitä pelaaja tekee pelissä annetuilla vaihtoehdoilla, ja pelin on osattava vastata näihin valintoihin. Pelaajaempatia ei ainoastaan luo parempaa pelattavuutta, vaan auttaa tunnistamaan ja eliminoimaan pelin ongelmia jo suunnitteluvaiheessa, jolloin muutosten tekeminen on paljon helpompaa, nopeampaa ja halvempaa tuotantovaiheeseen verrattuna. (Bates 2004, 17)

Pelaajan ja pelin välinen interaktio on yksinkertaista: pelaaja tekee jotain ja peli vastaa siihen tekemällä jotain muuta. Tämä interaktio erottaa pelin kaikista muista viihteen

muodoista. Siksi kaikki komennot mitä pelaaja antaa tulisi ottaa vastaan. Kaikesta tulee siis antaa jonkinlainen palaute pelaajalle tavalla tai toisella. Palaute voi olla esimerkiksi ääni tai jotain visuaalista. Tärkeintä on se, että pelaajan ja pelin välinen yhteys ei tunnu katkeavan, vaikka pelaajan antamalla komennolla ei olisikaan pelissä järkevää toimintoa. (Bates 2004, 18)

Pelaajan ja pelin välisen interaktion kannalta olennaista on myös toimiva käyttöliittymä. Käyttöliittymä vaikuttaa osaltaan pelin ulkonäköön ja välittää tietoa pelaajalle, sekä määrittää sen, miten pelaaja antaa pelille komentoja. Käyttöliittymän suunnittelussa on hyvä pyrkiä yksinkertaisuuteen niin paljon kuin mahdollista, kuitenkaan vaikeuttamatta pelaamista. Tärkeän informaation on oltava selkeästi näkyvillä tai vähintään helposti saatavilla. Joskus ratkaisuna toimii HUD (heads-up display), joka asettelee tiedon selkeästi pitkin näkymää, mutta joskus peliin saattaa soveltua paremmin yksi tietty alue näkymässä, johon kaikki tarvittava tieto on asetettu. Myös ohjauksen on oltava selkeä, ja pelaajan käytössä olevien komentojen tulisi olla mahdollisimman helppoja suorittaa. (Bates 2004, 26)

Pelin suunnittelussa on tärkeä tavoitella yksinkertaisuutta. Hyvä suunnittelija sisällyttää tuotokseensa vain ne asiat, jotka vaaditaan halutun vaikutuksen saavuttamiseksi. Muu on ylenpalttista, ja sillä on huono vaikutus lopputulokseen. Yksinkertaisuus auttaa myös projektin aikatauluttamisessa ja budjetoimisessa: kun kaikki tarvittava on tarkkaan määritelty, ei tule vahingossa hukattua resursseja turhaan materiaaliin jota ei lopulta hyödynnetä. Kun projektin aikana tulee ideoita uusista ominaisuuksista, tulee niiden tarpeellisuutta arvioida pelin pääajatus mielessä. Jos ominaisuus ei auta saavuttamaan pelin perimmäistä tavoitetta, niin se kannattaa jättää pois. (Bates 2004, 36)

3.3 Konseptointi

Pelin kehittäminen alkaa konseptoinnista. Tämä vaihe alkaa heti kun jollekin tulee mieleen tehdä peli, ja se päättyy esituotannon alkamiseen. Kehitystiimi on pienimmillään, ja siihen kuuluu yleensä vain muutama henkilö: suunnittelija, ohjelmistoarkkitehti (tech lead), konseptitaiteilija sekä tuottaja. Konseptoinnin tarkoituksena on saada

kuva pelistä kokonaisuutena ja saada se sellaiseen dokumentoituun muotoon, että kuka tahansa sen näkevä ymmärtää selkeästi mistä on kyse. Tässä vaiheessa päätetään esimerkiksi pelin pelaamisen pääpiirteet, suurpiirteinen graafinen kuvaus pelistä sekä pelistä riippuen myös tiivistelmä pelin tarinasta. (Bates 2004, 203)

3.4 Esituotanto

Esituotantoon kuuluu kaikki peliin liittyvä toiminta ennen varsinaista tuotantoa. Vaiheen päätavoitteita ovat pelisuunnittelun valmistuminen, art biblen luonti, tuotantopolun muodostaminen, projektisuunnitelman kirjoittaminen sekä prototyyppi (Bates 2004, 207). Alkuna on pelin niin kutsuttu high concept, joka muutamalla sanalla kuvaa sitä mistä pelissä on kyse (Moore & Novak 2010, 70). Tätä ajatusta laajennetaan kuvaamaan tarkemmin peliä. Esituotantotiimi koostuu tuottajista, suunnittelijoista, ohjelmoijista, taiteilijoista ja kirjoittajista, jotka kukin oman osa-alueensa näkökulmasta lopulta laativat eri tahot yhdistävän game design documentin. Dokumentti kuvaa pelistä kaiken olennaisen, käsittäen kaiken kenttäsuunnittelusta aina pelimekaniikoihin (Edwards 2006). Game design document elää kuitenkin koko projektin ajan, ja siihen tehdään tarpeen vaatiessa muutoksia, usein viikoittain, ellei jopa päivittäin (Oxland 2004, 241).

Art bible on pelin suunnittelijan, graafisen suunnittelijan sekä konseptitaiteilijan yhteistyönä luoma dokumentti, joka määrittelee pelin taiteellisen ja graafisen tyylin. Konseptitaiteilija luo mallikuvia joiden pohjalta muut graafikot työskentelevät, tuloksena yhtenäinen tyyli. Art biblen luonti kannattaa tehdä mahdollisimman varhain projektin aikana, jotta kaikki projektiin liittyvät taiteilijat ovat samalla viivalla ja grafiikan yhtenäisyys säilyy läpi koko projektin. (Bates 2004, 208)

Tuotantopolku on prosessi, jonka avulla ideasta tehdään todellinen. Esimerkkinä toimii esimerkiksi hahmon luominen peliin: suunnittelijan kuvauksesta siirrytään konseptitaiteilijan luomaan kuvaan. Sen pohjalta tehdään 3D-malli, jolle seuraavaksi määritellään tekstuurit. Malli voidaan tämän jälkeen animoida, ja seuraavaksi sille voidaan

asettaa tekoälyä. Viimeinen vaihe on hahmon kokeileminen käytännössä. Kaikkien tämän polun varrella käytettyjen työkalujen pitää toimia hyvin yhteen, jotta voidaan siirtyä vaiheesta toiseen. (Bates 2004, 209)

Technical design document kuvaa prosessin jonka avulla suunnitelmissa esiintyvät asiat muutetaan ohjelmistoksi. Siihen on määritelty siis projektin tekniset vaiheet ja arviot niihin kuluvasta ajasta. Dokumentissa on myös kuvattuna pelin kehittämiseen tarvittavat työkalut sekä tieto siitä, onko niitä jo käytettävissä vai pitääkö ne luoda tai ostaa. Lisäksi dokumentissa on tietoa tarvittavasta laitteistosta ja ohjelmistosta sekä mahdollisesta tarpeesta tehdä muutoksia organisaation infrastruktuuriin projektin tukemiseksi. (Bates 2004, 209)

Projektisuunnitelma kasaa kaikkien muiden suunnitelmien asiat samaan dokumenttiin ja se sisältää myös kaiken muun olennaisen mitä ei muualla ole käsitelty. Näihin kuuluvat esimerkiksi aikataulut, budjetti, henkilöstötarve ja virstanpylväiden määrittelyt. (Bates 2004, 210)

Esituotannon käsinkosketeltava tulos on prototyyppi. Sen tarkoitus on antaa elävä kuvaus pelistä ja esitellä niitä ominaisuuksia, jotka erottavat sen muista kaltaisistaan (Bates 2004, 211). Jos peli hakee ulkopuolista rahoitusta, niin prototyypin tärkeys korostuu entisestään, sillä se todistaa käytössä olevien menetelmien toimivan ja projektin etenevän (Novak 2012, 354).

3.5 Tuotanto

Tuotantovaiheessa kehitystiimi laajenee ja mukaan tuodaan lisää tuottajia, suunnittelijoita, taiteilijoita, ohjelmoijia ja kirjoittajia. Tuottajan toimenkuvaa voi verrata elokuva-alan vastaavaan. Tuottajan tarkoitus on toimia siltana projektin eri osastojen kanssa ja pitää huoli, että projekti etenee aikataulun mukaisesti. Suunnittelijan rooli on edelleen tärkeä, vaikka game design document onkin projektin tässä vaiheessa yleensä jo tehty. Suunnittelijan tehtävä on myös pitää huoli, että suunnitelmat implementoidaan oikein, ja että niihin voidaan tarpeen vaatiessa tehdä muutoksia. Näin voi käydä esimerkiksi siinä tapauksessa, että jonkin yksityiskohdan tai suunnitelman todetaan

olevan yksinkertaisesti toimimaton. Taiteilijat nimikkeensä mukaisesti vastaavat kaikesta pelissä nähtävästä taiteesta, grafiikasta ja animaatiosta. Heidän tehtävänsä on tehdä pelistä elävän ja hyvän näköinen. Jos pelissä käytetään liikkeenkaappausta (motion capture), on projektilla yleensä myös siitä vastaava tiimi. Ohjelmoijat työskentelevät pelin kirjastojen, moottorin ja tekoälyn parissa. Usein studioilla on valmiita kirjastoja ja moottori, jota käytetään useassa studion pelissä. Näitä räätälöidään projektin aikana sen tarpeita vastaavaksi. Lisäksi ohjelmoijat vastaavat pelaajan ja pelin välisestä interaktiosta sekä kaikesta, mitä itse pelissä tapahtuu. (Edwards 2006)

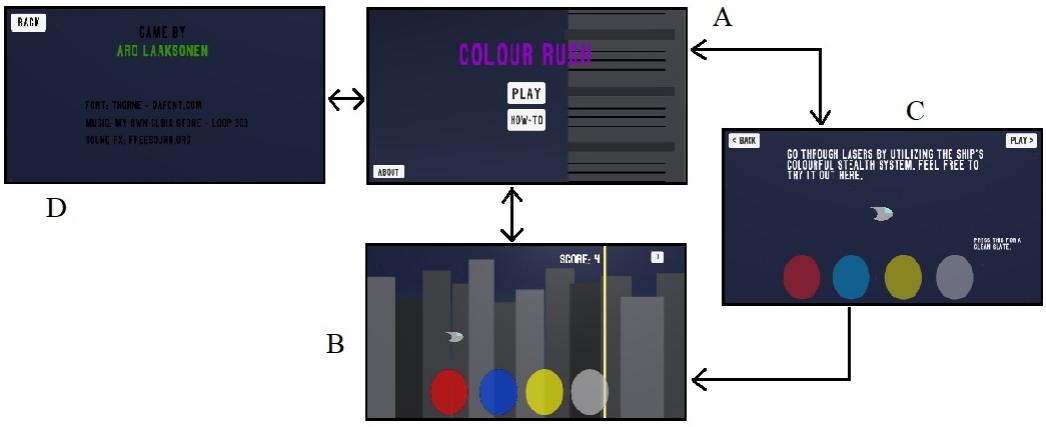
Alfa-version määritelmä vaihtelee organisaatiosta toiseen, mutta se yleensä tarkoittaa versiota pelistä, jonka voi pelata melkein kokonaan alusta loppuun. Työtä voi olla vielä jäljellä esimerkiksi grafiikan puolella, mutta pelimoottori, käyttöliittymä ja muut tärkeimmät toiminnot ovat valmiita. Alfan aikana kehityksen painotus siirtyy rakentamisesta viimeistelyyn. Tavallisesti projektiin otetaan tässä vaiheessa myös lisää testaajia, jotta suurin osa bugeista löydetään. (Novak 2012, 359)

Beta-versiossa kaikki kehitystyö on valmista, ja projektissa keskitytään bugien löytämiseen ja niiden korjaamiseen. Pientä hienosäätöä on vielä grafiikan ja tekstien puolella mahdollista tehdä, mutta tämän vaiheen tarkoitus on stabilisoida projekti ja eliminoida mahdollisimman paljon bugeja ennen julkaisua. Jos peli aiotaan julkaista konsoleille, täytyy ne hyväksyttää konsolivalmistajilla ennen julkaisua, jotta mahdollisiin ongelmiin ehditään vielä reagoida. Betan viimeinen vaihe on pahamaineinen crunch time. Sen tarkoituksena on kiristää tahtia vielä viimeisten viikkojen ja päivien aikana. Projektin työntekijät saattavat yöpyä työpaikalla useiden päivien ajan ollakseen mahdollisimman tuottavia. Sen aikana tehdään vaikeita päätöksiä lopulliseen tuotteeseen liittyen, vaikkapa arvio siitä kuinka tärkeitä eri bugit ovat. Kun crunch time on lopulta ohi, siirrytään code freezeen, jolloin peliin ei enää tehdä muutoksia ja se valmistellaan julkaisuun ja levitykseen. (Bates 2004, 214)

4 TOIMINNALLINEN OSA: COLOUR RUSH

4.1 Pelin kuvaus

Tämän työn toiminnalliseksi osaksi muodostui Colour Rush, nopeatempoinen mutta pelattuvuudeltaan yksinkertainen mobiilipeli. Pelaajahahmoa kuvastaa alus, joka kaahaa läpi öisen kaupungin. Pelaajan eteen ilmestyy tiheään tahtiin esteitä, jotka voivat väriltään olla yksi pääväreistä tai kahden päävärin yhdistelmä. Pelaajan on reagoitava nopeasti muuttamalla aluksen väriä seuraavaa estettä vastaavaksi ennen kuin osuu siihen. Jos aluksen väri ei vastaa estettä, niin alus tuhoutuu ja peli on ohi. Muussa tapauksessa pelaaja läpäisee esteen. Pelin tavoite on päästä mahdollisimman pitkälle ja pistemäärä kuvastaa pelaajan menestystä pelissä.



Kuva 1. Pelin näkymät.

Pelissä on neljä näkymää (scene) joiden välillä kuljetaan (Kuva 1). Ensimmäinen näkymä A on pelin aloitusvalikko, josta on pääsy kaikkiin muihin näkymiin. Näkymä B on varsinainen pelinäkymä, jossa pelaaja suorittaa pelin tavoitetta. Tästä näkymästä on mahdollisuus palata takaisin aloitusvalikkoon haluttuna aikana tai esimerkiksi silloin kun peli on päättynyt. Pelaajan on myös mahdollisuus aloittaa peli alusta suoraan tästä näkymästä, jolloin se ladataan uudestaan. Näkymä C on nimetty How-To-näkymäksi. Siinä selitetään pelin idea lyhyesti ja pelaajan on myös mahdollista kokeilla

värinäppäinten toimintaa kaikessa rauhassa, aloittamatta varsinaista peliä. Tästä näkymästä voidaan siirtyä suoraan varsinaiseen peliin tai takaisin aloitusvalikkoon. Näkymä D sisältää maininnat pelin tekijöistä ja siihen osallistuneista.

4.2 Esituotanto

Colour Rush –pelin esituotantovaihe oli varsin lyhyt, sillä pelin idea oli muotoutunut jo jonkin aikaa ennen työn varmistumista. Pelin high concept oli lyhyt ja ytimekäs: ”nopetempoinen mobiilipeli.” Kun aloin laajentaa tätä ajatusta etsin inspiraatiota ja vaikutteita muista mobiilipeleistä, varsinkin niistä jotka ovat menestyneet ja vastaavat oman pelini high conceptia. Jaoin pelit kahteen luokkaan perustuen niiden ”pelaamismentaliteettiin”, selkeämmin ilmaistuna siihen, mikä saa pelaajan pelaamaan peliä toistuvasti. Ensimmäinen luokka on pelit, joiden perusajatuksena on eteneminen; pelaaja saattaa kehittää hahmoa tai edetä kentästä toiseen. Ne vetoavat pelaajaan haluun edetä ja myöhemmin siihen investoituun aikaan ja saatuihin saavutuksiin. Huomasin tämän tyyppisen pelin olevan kiinnostava sinänsä, mutta se ei synnyttänyt minussa toimivia tai innostavia ideoita. Toinen luokka koostuu peleistä, joiden perusajatuksena on saada korkea pistemäärä. Pelaaja aloittaa pelin alusta joka kerta, toivon mukaan parantaen tulostaan pelikokemuksen karttuessa. Itse olen pelannut huomattavasti enemmän ensimmäiseen luokkaan kuuluvia pelejä, mutta sen takia juuri kiinnostuin toisesta luokasta. Siihen kuuluvien pelien avain toistuvaan pelaamiseen on tulkintani mukaan koukuttava pelattavuus. Pelaamisen on siis tunnuttava hyvältä ja hauskalta, jotta pelaaja haluaa käynnistää pelin myös toistamiseen. Esimerkkejä päättymättömistä peleistä voisivat olla vaikka Imangi Studiosin Temple Run ja GEARS Studion Flappy Bird.

Colour Rush -pelin ytimeksi muodostui siis koukuttava pelattavuus, ja pian lisäsin ajatukseen myös yksinkertaisuuden. Mobiililaitteet asettavat rajoituksia, joista esimerkiksi PC- ja konsolipelit eivät kärsi. Komennot annetaan laitteen näytön kautta, mikä sulkee monimutkaiset komento- ja näppäinasetelmat pois vaihtoehdoista. Näin yksinkertaisen pelattavuuden soveltuvan mobiiliympäristöön paremmin, ja pian muotoutui idea neljästä napista, joita käyttämällä pelaaja etenee pelissä.

4.3 Tuotanto

Unity-projekti sisältää skriptitiedostoja joissa haluttu toiminnallisuus, tässä tapauksessa C#:lla kirjoitettuna. Tiedostot osoitetaan halutulle peliobjektille (game object), joiden kautta skriptiä voi käsitelle esimerkiksi antamalla arvoja näkyviksi asetetuille muuttujille.

4.3.1 Taustan liikuttaminen

Koska pelaajahahmo on koko pelin ajan suurin piirtein näkymän samassa kohdassa, on liikettä helpompi simuloida taustaa vierittämällä. Tämän lähestymistavan ansiosta ei tarvitse liikuttaa sekä kameraa että pelaajaa. Quad on Unityssa taustaksi soveltuva komponentti litteän muotonsa vuoksi ja siksi, että sille voi asettaa kuvan.

```
void Update () {
    offset = new Vector2(Time.time * speed, 0);
    GetComponent<Renderer>().material.mainTextureOffset = offset;
}
```

Koodi 2. Taustakuvan liikuttaminen

Vector2-tyyppistä oliota luodessa sille annetaan parametreina kaksi float-tyyppistä arvoa. Ensimmäinen arvo on x-akselia varten, toinen y-akselia varten. Vector2-tyyppisen offset-muuttujan arvoksi tulee pelin alkamisaika kerrottuna nopeuden määrittävän speed-muuttujan arvolla sekä 0, sillä taustaa halutaan tässä tapauksessa siirtää vain horisontaalisesti. Pelin aloitusnäkymää siirtävässä koodissa nämä arvot ovat offset-muuttujaa luodessa toisin päin, jotta kuvaa voidaan vierittää pystysuunnassa. Muuttujan luonnin jälkeen se asetetaan taustan Renderer-komponentin mainTextureOffsetin arvoksi, jolloin kuva siirtyy halutun verran joka kerta kun pelikuva piirretään (Koodi 2).

4.3.2 Esteet, niiden luonti ja liikuttaminen

Pelaaja kohtaa pelin aikana esteitä, joita on useita eri värisiä. EnemySpawner-objekti sisältää komponenttinaan enemySpawner-skriptin joka pitää huolen esteiden ilmestymisestä ja niiden erilaisuudesta. Skriptissä käytetään coroutinea, joka on kuin metodi, jonka suorittamisen voi jättää kesken ja siihen voi palata myöhemmin. Lähestymistavan vuoksi on mahdollista luoda uusia esteitä toistuvasti, ja jokaisen esteen jälkeen voidaan arpoa uusi ajankohta, jolloin seuraava este luodaan.

```
void Start () {
    StartCoroutine(spawner());
}

void Update () {

    spawnWait = Random.Range(minWait, maxWait);

}

IEnumerator spawner() {
    yield return new WaitForSeconds(startWait);

    while(!stop) {
        enemyNo = Random.Range(0,6);
        Instantiate(enemies[enemyNo], transform.position, transform.rotation);

        yield return new WaitForSeconds(spawnWait);
    }
}
```

Koodi 3. Vihollisten luonti.

Start-tapahtumassa aloitetaan coroutine nimeltä spawner (Koodi 3). Coroutinen alussa määritetään startWait-muuttujalla aika, joka odotetaan ennen ensimmäisen esteen luontia. Lopulliseksi odotusajaksi on pelissä asetettu 2 sekuntia. Niin kauan kuin stop-muuttujan totuusarvo on false, luodaan uusia esteitä. Silmukan alussa arvotaan luku, jonka perusteella enemies-taulukosta valitaan satunnainen este. Instantiate-metodi luo tämän luvun ja mallipohjana käytetyn esteen kordinaattien perusteella kloonin halutusta esteestä kuvan ulkopuolelle. Tämän jälkeen coroutine odottaa spawnWait-muuttujan arvon verran ennen silmukan toistamista. SpawnWait-muuttuja arvotaan Update-

tapahtumassa, määritettyjen vähimmäis- ja enimmäisodotusaikojen väliltä. Lopullisessa pelissä vähimmäsajaksi on määritelty 0,7 sekuntia ja enimmäisajaksi 1,5 sekuntia.

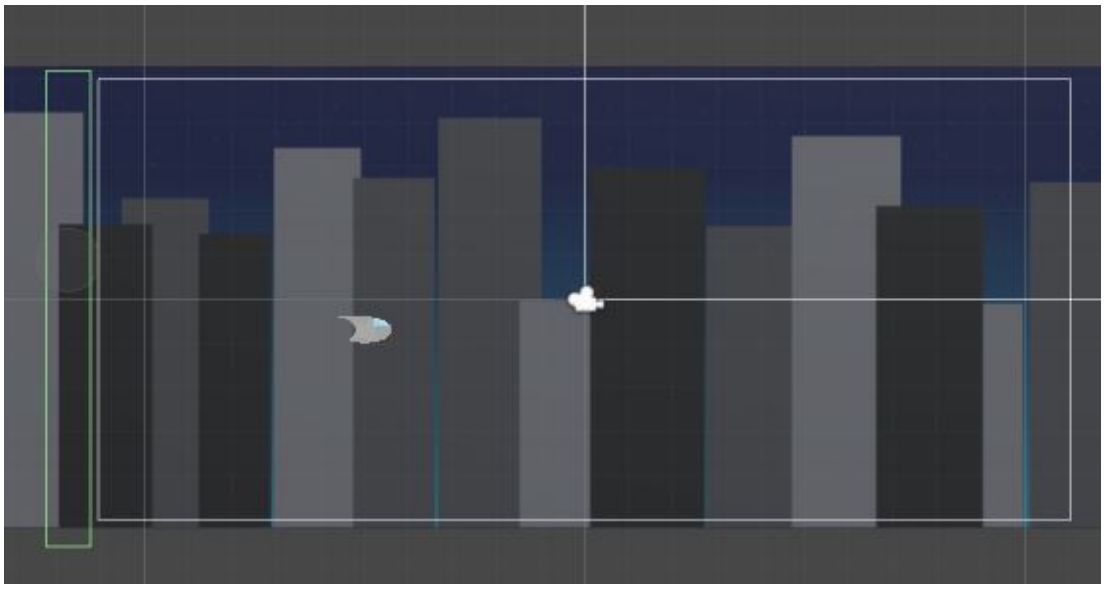
Pelin vaikeutta suunnitellessa päädyin lopulta siihen, että en halua pelistä nopeutuvaa, vaan haluan sen olevan vaikea jo alusta alkaen. Näin arvoltaan pienikin pistemäärä voi olla hyvä suoritus, mikä tuntui hauskalta ajatukselta.

```
void Update () {
    transform.Translate(new Vector3(-1,0,0) * speed * Time.deltaTime);
}
```

Koodi 4. Esteen liikuttaminen.

Esteen liikuttaminen on hyvin yksinkertainen lause (Koodi 4). Update-tapahtumassa objektia siirretään sen transform-komponentin Translate-metodin avulla. Joka kuvassa sille annetaan uusi arvo kertomalla liikkumissuunta Vector3-tyyppisessä oliossa, editorissa määritelty nopeus ja edellisen kuvan piirtämiseen kulunut aika keskenään. Time.deltaTimen käyttö on tärkeää, jotta haluttu objekti liikkuu oikein. Sen avulla liikkuminen riippuu ajasta eikä piirtonopeudesta. Ilman sitä este liikkuisi tehokkaammalla laitteella merkittävästi nopeammin.

Koska pelin aikana voidaan teoriassa luoda ääretön määrä esteitä, on tärkeää säästää laitteen resursseja tuhoamalla jo ohitetut esteet. Pelinäkymän ulkopuolelle on asetettu peliobjekti nimeltä Destroyer, johon osuessaan esteet tuhoutuvat, poistuen käytöstä (Kuva 2).



Kuva 2. Valkoiset ääriviivat kuvaavat pelaajalle näkyvissä olevaa aluetta. Vihreä laatikko on Destroyer, asetettuna kameran ulkopuolelle.

EnemyDestroyer-skriptissä on tapahtuma OnTriggerEnter2D, joka saa parametrinaan siihen osuvan Collider-komponentin. Ehtolauseessa tarkistetaan, onko osuneen peliobjektin tunnisteeksi merkattu estettä merkitsevä ”Enemy”. Jos niin on, osunut objekti tuhotaan (Koodi 4).

```
void OnTriggerEnter2D(Collider2D other) {
    if(other.gameObject.tag.Equals("Enemy")) {
        Destroy(other.gameObject);
    }
}
```

Koodi 4. Ohitetun esteen tuhoaminen.

4.3.3 Pelaajahahmo

Colour Rush-pelissä pelaajahahmoa kuvaa pieni lentävä alus, joka peliteknisesti ei liiku eteen tai taakse, mutta kylläkin vertikaalisesti ylös ja alas leijuvan näköisesti. Se muuttaa väriä pelaajan komentojen mukaan, sekä reagoi osuessaan esteisiin. Jos este on erivärinen aluksen kanssa, se päästää räjähdysäänen ja vajoaa pelinäkymän ulkopuolelle. Esteen ollessa kanssaan saman värinen alus läpäisee sen, jolloin tulee erilainen ääni.

```
void Start () {
    color = "gray";
    gameOver = false;
    startingYPos = this.transform.position.y;
    audioSource = GetComponent<AudioSource>();
}
```

Koodi 5. Pelaajahahmon alustus.

Start-tapahtumassa pelaajahahmolle alustetaan muutama arvo. Color-muuttuja yksinkertaisesti määrittää mikä väri on aktiivinen, mutta se ei varsinaisesti määritä minkä värinen versio pelaajahahmosta on näkyvissä. GameOver-muuttuja kertoo, onko peli käynnissä ja se aktivoi muuttuessaan muita toimintoja. StartingYPos-muuttujaan tallennetaan pelaajahahmon sijainti y-akselilla pelin alussa. AudioSource-muuttujaan tallennetaan AudioSource-komponentti myöhempää käyttöä varten. (Koodi 5).

```
void Update () {
    if(gameOver == false) {
        transform.position = new Vector3(transform.position.x, startingYPos +
        ((float)Mathf.Sin(Time.time) * maxYPos), transform.position.z);
    }
}
```

Koodi 6. Pelaajahahmon liikuttaminen ylös ja alas.

Update-tapahtumassa liikutetaan pelaajahahmoa ylös ja alas. Lopputuloksena on elävämpi, leijuva liike. Jos gameOver-muuttujan arvo on epätosi eli pelaaja ei ole hävinnyt, siirretään pelaajahahmoa asettamalla sen positiolle uusi Vector3-tyyppinen arvo. Tämä uusi arvo saadaan antamalla x- ja z-akselin arvot muuttumattomina, mutta y-akselin arvo luodaan lisäämällä pelaajahahmon alkuperäiseen y-kordinaattiin erikseen määritellyn enimmäiskorkeuden (maxYPos) ja Mathf-luokan Sin-metodilla saatavan sinin tulo (Koodi 6).

```
void OnTriggerEnter2D(Collider2D other) {
    if(!other.gameObject.name.EndsWith(color + "(Clone)")){
        gameOver = true;
        gameObject.GetComponent<Rigidbody2D>().gravityScale = 1;
        audioSource.PlayOneShot(explosionClip);
        uIManager.GameOver();
    }
    else {
        audioSource.PlayOneShot(energyClip);
    }
}
```

Koodi 7. Pelaajahahmon reagointi osumiin.

Kuten vihollisen osuessa Destroyer-objektiin, pelaajahahmoa hallitsevassa skriptissäkin on OnTriggerEnter2D-tapahtuma, joka saa parametrinaan siihen osuneen Collider-komponentin. Jos pelaajahahmoon osuneen esteen nimi ei vastaa pelaajahahmon aktiivista väriä, niin alkavat pelin päättymiseen liittyvät toimenpiteet. GameOver-muuttujan arvoksi tulee true, jolloin esimerkiksi pelaajahahmon leijuva liike pysähtyy. Tämän jälkeen hahmon fysiikoita simuloivalle Rigidbody2D-komponentille annetaan gravityscalen arvoksi 1, jolloin pelaajahahmoon sovelletaan painovoiman simulointia. Tämä saa aikaan sen, että hävitessään pelaajan hahmo vajoaa alas näkymän ulkopuolelle. Ohjelmallisesti tämän jälkeen, mutta pelaajan näkökulmasta samalla hetkellä osuman kanssa AudioSource-komponentti soittaa räjähdysäänen. Lopuksi tieto pelin päättymisestä välitetään uIManager-muuttujan kautta UIManager-peliobjektille, joka alkaa suorittaa omia pelin päättymiseen liittyviä toimenpiteitään. (Koodi 7).

```
public void changeColor(Button button) {
    string buttonName = button.name;
    string newColor = determineColor(buttonName);

    color = newColor;
    switch(color) {
        case "gray":
        this.GetComponent<SpriteRenderer>().sprite = sprites[0];
        break;

        case "red":
        this.GetComponent<SpriteRenderer>().sprite = sprites[1];
        break;

        case "blue":
        this.GetComponent<SpriteRenderer>().sprite = sprites[2];
        break;

        case "yellow":
        this.GetComponent<SpriteRenderer>().sprite = sprites[3];
        break;

        case "green":
        this.GetComponent<SpriteRenderer>().sprite = sprites[4];
        break;

        case "orange":
        this.GetComponent<SpriteRenderer>().sprite = sprites[5];
        break;

        case "purple":
        this.GetComponent<SpriteRenderer>().sprite = sprites[6];
        break;
    }
}
```

Koodi 8. ChangeColor-metodi.

Pelaajahahmon värin vaihtaminen tapahtuu changeColor-metodin ja determineColor-metodin yhteistyönä.

```
public string determineColor(string buttonColor) {
    string newColor= "";
    switch(buttonColor) {
        case "ButtonRed" :
            if(color.Equals("gray")){
                newColor = "red";
            }
            else if(color.Equals("blue")){
                newColor = "purple";
            }
            else if(color.Equals("yellow")){
                newColor = "orange";
            }
            else {
                newColor = "noChange";
            }
            break;
        case "ButtonBlue" :
            if(color.Equals("gray")){
                    newColor = "blue";
            }
            else if(color.Equals("red")){
                    newColor = "purple";
            }
            else if(color.Equals("yellow")){
                    newColor = "green";
            }
            else {
                    newColor = "noChange";
            }
            break;
        case "ButtonYellow" :
            if(color.Equals("gray")){
                    newColor = "yellow";
            }
            else if(color.Equals("red")){
                    newColor = "orange";
            }
            else if(color.Equals("blue")){
                    newColor = "green";
            }
            else {
                    newColor = "noChange";
            }
            break;
        case "ButtonClear" :
            newColor = "gray";
            break;
    }
    return newColor;
}
```

Koodi 9. DetermineColor-metodi.

Kaikki olisi voinut tietysti tapahtua samassa metodissa, mutta koin selkeämmäksi tehdä kaksi pitkää metodia yhden todella pitkän sijasta. ChangeColor-metodi saa parametrinaan napin, joka on rekisteröinyt pelaajan painalluksen. Napista otetaan talteen sen nimi, joka annetaan determineColor-metodille. Siellä käydään läpi switch casella minkä on pelaajahahmon uusi väri. (Koodi 9) Lopulta determineColor palauttaa newColor muuttujassa merkkijonon, joka kertoo tämän värin. Takaisin changeColor-metodissa newColor asetetaan pelaajahahmon colorin arvoksi. Kun uusi arvo on asetettu, valitaan sen perusteella sprites-taulukosta pelaajahahmolle uuden värinen sprite eli kuva (Koodi 8).

4.3.4 UI Manager

Käyttöliittymää pelissä ohjaa UIManager-peliobjekti, jolla on komponenttinaan skripti nimeltä UIManager.

```
void Start () {
    score = 0;
    gameOver = false;
    InvokeRepeating("UpdateScore", 1.0f, 0.5f);

    foreach(Button button in buttons){
        button.gameObject.SetActive(false);
    }
}
```

Koodi 10. UIManagerin alustus.

UIManagerin Start-tapahtumassa asetetaan pistemäärää kuvaavan score-muuttujan arvoksi luonnollisesti 0, ja koska peli on vasta alkanut, asetetaan gameOverin arvoksi epätosi. InvokeRepeating-metodi antaa mahdollisuuden toistaa jotain muuta metodia tietyin väliajoin. Tässä tapauksessa toistetaan pistemäärää päivittävää UpdateScore-metodia sitten, kun pelin alkamisesta on kulunut sekunti ja tämän jälkeen puolen sekunnin välein. (Koodi 10). UpdateScore-metodin päivittämää lukua hyödynnetään jokaisessa piirretyssä näkymässä, sillä pistemäärän näyttäminen pelaajalle tapahtuu Update-tapahtumassa (Koodi 11). ScoreText-muuttuja edustaa pelikomponenttia joka näkyy käyttöliittymässä.

```
public void UpdateScore() {
    if(gameOver != true){
        score += 1;
    }
}

void Update() {
    scoreText.text = "Score: " + score;
}
```

Koodi 11. Pisteiden päivittäminen.

Pelaaja voi halutessaan pysäyttää pelin painamalla oikeaan yläkulmaan sijoitettua Pause-nappia. Se on asetettu käyttämään UIManager-skriptin Pause-metodia, jonka toiminta riippuu siitä, onko peli käynnissä vai pysäytettynä. Tähän käytetään Unityn Time-luokan timeScale-muuttujaa, jonka arvo 1 tarkoittaa pelin kulkevan normaalilla nopeudella. Arvo 0 tarkoittaa pelin olevan pysähtynyt. Pause-metodiin tullessa tarkestellaan timeScalen arvoa, ja sen perusteella peli joko jatkuu tai pysähtyy. Lisäksi Replay- ja Menu-näppäimet tuodaan esiin tai poistetaan näkyvistä kulloinkin voimassa olevan timeScalen mukaan. Pysähtyneenä peli tarjoaa siis mahdollisuuden aloittaa alusta tai siirtyä aloitusnäkymään. (Koodi 12).

```
public void Pause() {
    if(Time.timeScale == 1) {
        Time.timeScale = 0;

        foreach(Button button in buttons){
            button.gameObject.SetActive(true);
        }
    }
    else if(Time.timeScale == 0) {
        Time.timeScale = 1;

        if(!gameOver) {
            foreach(Button button in buttons){
                button.gameObject.SetActive(false);
            }
        }
    }
}
```

Koodi 12. Pelin pysäyttäminen.

TimeScalea käytetään myös skriptin Play- ja Menu-metodeissa, jotka nimiensä mukaisesti käynnistävät uuden pelin tai siirtyvät aloitusnäkymään. Siinä tapauksessa, että pelaaja on ensin pysäyttänyt pelin ja painanut tämän jälkeen ilmestyneitä Replay- ja Pause-nappeja, on TimeScalen arvo 0 myös siirryttäessä uuteen näkymään ja peli siis pysähtynyt. Tämän takia TimeScalelle asetetaan arvo 1 aina näitä näppäimiä käytettäessä. (Koodi 13).

```
public void Play() {
    SceneManager.LoadScene("level1");
    Time.timeScale = 1;
}

public void Menu() {
    SceneManager.LoadScene("openingScene");
    Time.timeScale = 1;
}
```

Koodi 13. Play- ja Menu-metodit

UIManagerin viimeinen osa on GameOver-metodi, jonka tehtävä on varsin yksinkertainen. Sen lisäksi, että gameOver-muuttujan arvoksi asetetaan aluksi tosi, myös pelaajan käytössä olevat värinäppäimet poistetaan näkyvistä ja tilalle tulevat pysäytetystä näkymästä tutut Replay- ja Menu-näppäimet. (14).

```
public void GameOver() {
    gameOver = true;
    foreach(Button button in buttons){
        button.gameObject.SetActive(true);
    }
    foreach(Button colorButton in colorButtons) {
        colorButton.gameObject.SetActive(false);
    }
}
```

Koodi 14. GameOver-metodi.

4.3.5 Äänet ja musiikki

Colour Rush käyttää kolmea eri äänitiedostoa. Kaksi niistä on esteisiin osuttaessa soitettavia äänitehosteita. Pelaajan hävitessä soittaa AudioSource-komponentti ääniklipin

joka kuulostaa räjähdykseltä. Kun pelaaja läpäisee esteen, soitetaan klippi joka kuulostaa hieman vaimeammalta, hieman jopa sähköiseltä energiapurkaukselta. Koin tämän jälkimmäisen äänitehosteen lähes yhtä tärkeäksi kuin negatiivisen räjähdysäänen, sillä pelaajan on hyvä saada positiivista palautetta onnistumisistaan. Pistemäärän seuraamisen sijasta tämä positiivinen palaute tulee äänen muodossa.

Kolmas ääni mikä pelissä kuullaan on musiikkikappale, joka soi pelin alkunäkymästä saakka keskeytyksettä eri näkymien välillä. Musiikkia soittaa BackgroundMusic-niminen peliobjekti joka sijaitsee pelin aloitusnäkymässä. Tällä komponentilla on oma Audio Source-komponentti, jolle kappale on määritetty. Lisäksi peliobjektilla on lyhyt skripti, jonka estää objektin tuhoutumisen silloin kun siirrytään pelin muihin näkymiin. Skriptin Awake-tapahtuman suoritus heti kun siitä ladataan instanssi. Objects-taulukkoon laitetaan kaikki peliobjektit, joille on annettu tunniste ”Music”. Se on annettu vain BackgroundMusic-objektille, jollainen luodaan aina siirryttäessä pelin aloitusnäkymään. If-ehtolauseessa tarkistetaan, sisältääkö taulukko enemmän kuin yhden objektin, ja jos niin on, niin uusin niistä tuhotaan. Ainoastaan siis ensimmäisenä luotu objekti lopulta suorittaa metodin DontDestoryOnLoad, joka estää sen tuhoutumisen muihin näkymiin siirryttäessä. (Koodi 15)

```
public class MusicPlayer : MonoBehaviour {

    void Awake() {
        GameObject[] objects = GameObject.FindGameObjectsWithTag("Music");
        if(objects.Length > 1) {
            Destroy(this.gameObject);
        }
        DontDestroyOnLoad(this.gameObject);
    }
}
```

Koodi 15. Musiikin soittaminen keskeytyksettä.

Pelin musiikiksi valikoitui My Own Cubic Stonen kappale ”Loop 303”. Kappale on mielestäni sopivan synkkä ja sen kulku kehittyy varsinkin alkuvaiheessa juuri sopivasti suhteessa siihen mitä pelaajalla on pelissä näkyvissä sillä hetkellä. Sen tunnelma luo mielikuvia tieteisfiktiosta, kokeellisesta teknosta, cyberpunkista ja jopa vanhoista hirviöelokuvista. Kappale on vapaasti ladattavissa Free Music Archive –sivustolla.

Molemmat äänitehosteet on ladattu Free Sound –sivustolta. Lähestymistapani oikeiden äänien löytämiseen oli täysin päinvastainen musiikin löytämiseen nähden. Musiikin kanssa minulla oli käsitys siitä mitä haluan ja mitä etsiä, mutta äänien kanssa ajattelin pikemminkin tunnistavani etsimäni äänet kuullessani ne, ilman varsinaisia ennakko-odotuksia.

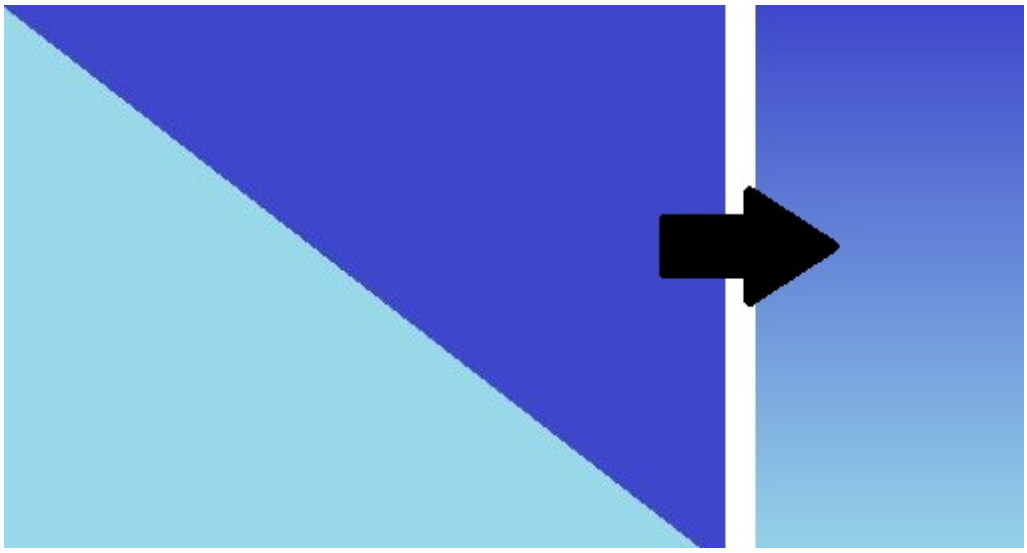
4.3.6 Grafiikka

Grafiikan tekeminen oli yksi pelin kehittämisen vaikeimmista osista. Varsin rajallisen taiteellisen ja graafisen osaamiseni takia odotukset ja vaatimukset oli asetettava sellaiselle tasolle, mille voin käytettävissä olevan ajan ja välineiden puitteissa ylettää. Ensimmäinen adjektiivi, jonka määrittelin grafiikalle olikin ”yksinkertainen”. Tämän jälkeen määritelmään lisäsin myös sanan ”selkeä”. Eri ideoita pyöritellessäni totesin, että nämä varsin rajoittavat määritelmät voisivat kuitenkin toimia myös grafiikkani tyylikeinona. Jopa liioiteltu yksinkertaisuus ja yksityiskohtien vähäisyys olivat ratkaisu, johon lopulta olin tyytyväinen. Yksityiskohtia on kuitenkin muutamissa pelin grafiikoissa, kuten pelaajahahmossa ja taustan yötaivaassa (Kuva 3).

Kuva 3. Pelaajahahmon oranssi versio.

Varsin yksinkertaisen rakenteensa vuoksi peli ei lopulta vaatinut kovin montaa kuvaa. Pelaajahahmon aluksesta on seitsemän versiota, joita pelin aikana vaihdellaan. Esteinä toimivia lasersäteitä on kuusi erilaista. Pelattaessa käytettävät värinäppäimet ovat käytännössä vain 4 eriväristä ympyrää jotka on asetettu näppäinkomponenttien kuviksi.

Taustakuvien yötaivaissa on hyödynnetty kuvan toistuvasta kaventamisesta ja venyttämisestä aiheutuvaa efektiä, jolla kaksi eri väriä saadaan yhdistymään sujuvasti yhteen. (Kuva 4).



Kuva 4. Kaksi väriä ennen ja jälkeen kuvan venyttämisen.

5 LOPUKSI

Tämä opinnäytetyö ja sen sivutuote Colour Rush ovat olleet varsin haasteellinen, mutta ennen kaikkea opettavainen kokemus. Olen oppinut paljon pelialasta sekä työssä käyttämistäni teknologioista ja alustoista. Suurimmat projektista saamani opetukset liittyvät kuitenkin itse työskentelyyn. En voi väittää, että aina projektia työstäessäni olisin palanut halusta koodata ja kirjoittaa. Minä en vain yksinkertaisesti toimi niin. Silloin on ollut tärkeä pitää mielessä se, että olen vastuussa vain itselleni projektin valmistumisesta. Hammasta purrenkin tuloksia kuitenkin saavuttaa, kunhan ei anna periksi. Kääntöpuolena opin myös sen, että kun flown saavuttaa ja tekeminen tuntuu hyvältä, niin se tuntuu todella todella hyvältä. Kun näkee oman projektin muuttuvan paperiin kirjatuista hatarista ajatuksista todellisuudeksi, on vaikea keksiä vastaavaa tunnetta.

Työn molempia osia, kirjallista ja toiminnallista oli tarkoitus viedä eteenpäin samaan tahtiin. Pian kirjallinen osa jäi kuitenkin tauolle pitkäksi aikaa, toiminnallisen osan edetessä pala kerrallaan. Osittain ilmiön voi selittää toiminnallisen osan välittömällä palautteella: onnistumiset koukuttavat. Uusien ominaisuuksien toteuttaminen ja ihailu jättivät kirjallisen osan varjoonsa. Kirjallisen osan tullessa eteen sai tutustua toisenlaisiin onnistumisiin. Asiat joista olen kirjoittanut ovat mielestäni todella mielenkiintoisia ja voin todeta, että kirjoitettua lukua voi ihailla melkein leveä hymy kasvoilla siinä missä toimivaa peliäkin.

Työni toiminnallisena lopputuloksena on peli, joka on mielestäni hauska ja jossa näkyvät alkuperäisen suunnitelmani pääpointit. Kirjallinen osuus on mielestä hyvä läpileikkaus käyttämistäni tekniikoista sekä peliteollisuuden menetelmistä.

Toivon että tämä opinnäytetyö rohkaisisi lukijaansa tutustumaan pelialaan, pelinkehitykseen ja peleihin. Näillä aiheilla on paljon tarjottavaa viihteestä, teknologiasta ja luomisesta kiinnostuneille.

LÄHTEET

Amadeo, R. 2017. Android execs get technical talking updates, Project Treble, Linux, and more. Ars Technica. Viitattu 6.2.2018. https://arstechnica.com/gadgets/2017/05/ars-talks-android-googlers-chat-about-project-treble-os-updates-and-linux/

Android. Viitattu 6.2.2018. https://developer.android.com/about/versions/oreo/index.html

AppBrain. 2017. Number of Android applications. Viitattu 6.2.2018. https://web.archive.org/web/20170210051327/https://www.appbrain.com/stats/number-of-android-apps

Bates, B. 2004. Game Design (2nd ed.). Course Technology.

Bethke, E. 2003. Game development and production. Texas: Wordware Publishing, Inc.

Brightman, J. 2017. Mobile games booming as global games market hits $108.9B in 2017. GamesIndustry.biz. Viitattu 15.3.2018. https://www.gamesindustry.biz/articles/2017-04-20-mobile-games-booming-as-global-games-market-hits-usd108-9b-in-2017-newzoo

Brodkin, J. 2013. How Unity3D Became a Game-Development Beast. Dice. Viitattu 8.2.2018. https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

Burnette, E. 2008. Patrick Brady dissects Android. ZDNet. Viitattu 6.2.2018. http://www.zdnet.com/article/patrick-brady-dissects-android/

Chandler, H. 2009. The Game Production Handbook (2nd ed.). Hingham, Massachusetts: Infinity Science Press.

ECMA International. 2017. C# Language Specification. Viitattu 14.2.2018. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Edwards, R. The game production pipeline: Concept to completion. IGN. Viitattu 16.3.2018. http://www.ign.com/articles/2006/03/16/the-game-production-pipeline-concept-to-completion?page=1

Hamilton, N. 2008. The A-Z of Programming Languages: C#. Computerworld. Viitattu 14.2.2018. https://www.computerworld.com.au/article/261958/a-z_programming_languages_c_/

Hildenbrand, J. 2012. What is a kernel? Android Central. Viitattu 6.2.2018. https://www.androidcentral.com/android-z-what-kernel

International Data Corporation. Smartphone OS. Viitattu 5.2.2018. https://www.idc.com/promo/smartphone-market-share/os

Kovach, S. 2013. Android's Massive Fragmentation Problem In Two Handy Charts. Business Insider. Viitattu 6.2.2018. http://www.businessinsider.com/android-fragmentation-report-2013-7?r=US&IR=T&IR=T

Kovach, S. 2014. Microsoft Is Still Going To Sell Nokia's Android Phones. Business Insider. Viitattu 6.2.2018. http://www.businessinsider.com/microsoft-selling-nokia-x-android-phones-2014-4?r=US&IR=T&IR=T

Love, R. 2013. What Are The Major Changes That Android Made To The Linux Kernel? Forbes. Viitattu 6.2.2018. https://www.forbes.com/sites/quora/2013/05/13/what-are-the-major-changes-that-android-made-to-the-linux-kernel/#575db2fd7bb6

McGuire, M.; Jenkins, O. 2009. Creating Games: Mechanics, Content, and Technology. Wellesley, Massachusetts: A K Peters.

Menard, M. 2011. Game Development with Unity. Course Technology. Viitattu 22.5.2018. https://ebookcentral.proquest.com/lib/samk/detail.action?docID=3136415

Novak, J. 2012. Game Development Essentials: An Introduction, Third Edition. Delmar Cengage Learning

Oxland, K. 2004. Gameplay and design. Addison Wesley.

Shewaramani M. 2015. 14 Reasons why developers love Unity 3D as Cross Platform. Credencys. Viitattu 22.5.2018. https://www.credencys.com/blog/14-reasons-why-developers-love-unity-3d-as-cross-platform/

Sinhal, A. 2017. Closer Look At Android Runtime: DVM vs ART. AndroidPub. Viitattu 7.2.2018. https://android.jlelse.eu/closer-look-at-android-runtime-dvm-vs-art-1dc5240c3924

Takahashi, D. 2014. John Riccitello sets out to identify the engine of growth for Unity Technologies (interview). VentureBeat. Viitattu 8.2.2018. https://venturebeat.com/2014/10/23/john-riccitiello-sets-out-to-identify-the-engine-of-growth-for-unity-technologies-interview/

Technopedia. Viitattu 16.3.2018. https://www.techopedia.com/definition/24261/mobile-games

Telles M. 2001. C# Black Book. Paraglyph Press. Viitattu 21.4.2018 https://ebookcentral.proquest.com/lib/samk/detail.action?docID=3384714

Toombs, C. 2013. Meet ART, Part 1: The New Super-Fast Android Runtime Google Has Been Working On In Secret For Over 2 Years Debuts in KitKat. Android Police. Viitattu 7.2.2018.
http://www.androidpolice.com/2013/11/06/meet-art-part-1-the-new-super-fast-android-runtime-google-has-been-working-on-in-secret-for-over-2-years-debuts-in-kitkat/

Unity. Viitattu 8.2.2018.
https://unity3d.com/

Unity. Viitattu 8.2.2018.
https://unity3d.com/unity/features/multiplatform

Watson K., Nagel C., Pedersen J., Reid J., Skinner M. 2010. Beginning Visual C# 2010. John Wiley & Sons, Incorporated. Viitattu 20.4.2018
https://ebookcentral.proquest.com/lib/samk/detail.action?docID=510105

Watson, M. 2017. Why .NET Core and C# are the Next Big Thing. Stackify. Viitattu 23.5.2018.
https://stackify.com/net-core-csharp-next-programming-language/

Zinoune, M. 2012. Why is Android built on Linux Kernel? Unixmen. Viitattu 21.5.2018.
https://www.unixmen.com/why-is-android-built-on-linux-kernel/

Äänet ja musiikki:

“Bomb - Small”
https://freesound.org/people/Zangrutz/sounds/155235/

“Energy Whip 2”
https://freesound.org/people/ejfortin/sounds/49695/

My Own Cubic Stone – Loop 303
http://freemusicarchive.org/music/My_Own_Cubic_Stone/Loops_1972/13_-_Loop_303