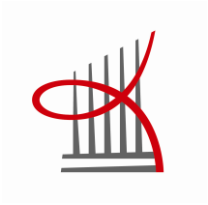


Mobiilipelin tekninen suunnittelu ja toteutus Unity-pelimoottorilla

Case: Royal Crow Squadron

Juho Tervo

Opinnäytetyö
Toukokuu 2014
Tietojenkäsittely
Ohjelmistotuotanto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

TERVO, JUHO:
Mobiilipelin tekninen suunnittelu ja toteutus Unity-pelimoottorilla
Case: Royal Crow Squadron

Opinnäytetyö 48 sivua
Toukokuu 2014

Suomessa on tällä hetkellä alati kasvava määrä erikokoisia peliyrityksiä, jotka tarvitsevat nyt ja tulevaisuudessa erilaisia pelialan ammattilaisia palvelukseensa. Royal Crow Squadron on mobiilipeli, jonka synty lähti tarpeesta tuoda esiin tekijöidensä osaaminen tällä alalla. Opinnäytetyössä käydään pelin kehitystä beta-vaiheeseen asti teknisten ratkaisujen osalta.

Tämän opinnäytetyön tavoitteena oli hankkia tietoa ja kokemusta mobiilipelin teknisestä suunnittelusta ja toteutuksesta. Sen tarkoituksena oli teknisesti suunnitella ja toteuttaa mobiilipeli Royal Crow Squadron.

Työn teoreettisessa osiossa keskitytään ensin peliohjelmointiin yleisesti, kartoitetaan mobiilipelejä ja niiden nykyistä tilannetta sekä käydään läpi teknisten ratkaisujen vaihtoehdot ja syyt, miksi valittuihin ratkaisuihin päädyttiin. Tämän jälkeen kerrotaan yleisesti Unity-pelimoottorista ja sen eri ominaisuuksista.

Opinnäytetyön lopputuotteena on pelattava ja toimiva mobiilipelin beta-versio, joka on suunnittelun mukainen. Sen tekeminen myös kartutti tekijöidensä taitoja ja kokemusta. Tästä syystä opinnäytetyötä voi pitää onnistuneena. Tarkoitus on kuitenkin vielä viedä pelin kehitys loppuun ja julkaista siitä lopullinen versio Play Storessa.

Asiasanat: mobiilipelit, peliohjelmointi, unity

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

TERVO, JUHO:
Technical Design and Implementation of a Mobile Game on the Unity Game Engine
Case: Royal Crow Squadron

Bachelor's thesis 48 pages
May 2014

Finland's game industry is experiencing a boom: an ever increasing number of domestic game studios have sprung up and still are continue to emerge. Those studios need manpower to make their games. Royal Crow Squadron is a mobile game which began as its makers' instrument for showing their skill and understanding of the industry. This thesis is the story of developing that mobile game from its beginning to beta version concentrating on technical solutions.

The objective of this thesis was to accumulate knowledge and experience on technically designing and implementing a mobile game. Its purpose was to technically design and implement a mobile game, Royal Crow Squadron.

The first half of the theory part in this thesis discusses general game programming, mobile games and the industry's current state. It also addresses the multitude of technical choices made during the development of this game and why those choices were made. The latter half focuses on Unity game engine and its key elements.

The result is a playable and functioning beta version of a mobile game. It works as designed. The development process of this particular mobile game accumulated knowledge and experience for its makers so it is safe to say this thesis was a success. However the game is not quite ready to be published so the next step should be to finalize it and publish it on Google Play.

Key words: mobile games, game programming, unity

SISÄLLYS

ERITYISSANASTO

Shoot ’em up	videopelilajityyppi, jossa tarkoituksena on ampua isoa määrää vihollisia samalla väistellen vihamielistä tulta
2.5D	tekniikka, jossa yhdistellään kaksi- ja kolmiulotteisia graafisia elementtejä
AAA-peli	peli, jonka tuotantoarvot ovat niin korkeat, että sitä pidetään alaa määrittelevänä
Ohjelmistokehys	ohjelmistokehitystä helpottava valmis runko, jonka päälle rakennetaan ohjelmia
Luokkakirjasto	valmis kokoelma luokkia ja metodeja
Alustariippuumaton	sovellus tai tekniikka, jota ei ole sidottu tiettyyn käyttöjärjestelmään
Avoin lähdekoodi	menetelmä, jossa käyttäjä voi osallistua ohjelmiston kehitykseen
Skripti	komentosarja, joka antaa käskyjä tietylle ohjelmalle

1 JOHDANTO

Joulukuussa 2009 suomalainen peliyritys Rovio julkaisi Applen iOS-mobiilikäyttöjärjestelmälle pelin nimeltä Angry Birds, joka kuusi kuukautta myöhemmin saavutti ykkössijan Applen App Storessa (Sandstrom, 2010). Sittemmin Angry Birdsia on ladattu eri alustoilla yli kaksi miljardia kappaletta (Long, 2014), ja se on synnyttänyt ympärilleen kokonaisen tuoteperheen huvipuistoa myöten (Bond, 2012).

Yksittäistä menestystarinaa tärkeämpänä voi kuitenkin pitää Angry Birdsin aloittamaa buumia. Tällä en tarkoita varsinaisesti suomalaista pelaamisbuumia vaan pelintekemisbuumia. Suomalainen pelialan yhdistys Neogames arvioi tällä hetkellä olevan noin 200 suomalaista pelinkehitykseen keskittynyttä yritystä. Samainen yhdistys kertoo pelikoulutusta tarjottavan Suomessa neljässä toisen asteen oppilaitoksessa, kymmenessä ammattikorkeakoulussa sekä kolmessa yliopistossa. Nämä peliyritykset tarvitset luonnollisesti palvelukseensa suuren määrän erilaisia alan ammattilaisia, joilla on intohimoa sekä pelaamiseen että pelien tekemiseen. Merkittävänä tekijänä näiden yritysten palvelukseen pääsemisessä on portfolio, josta käy ilmi hakijan kokemus pelien tekemisen parissa. (Neogames)

Royal Crow Squadronin kehitys lähti tarpeesta tuoda ilmi tekijöidensä osaaminen. Sen konsepti kertoo tekijöidensä historiasta videopelaamisen parissa yli 20 vuoden ajalta käyttäen esikuvinaan modernien mobiilipelien sijaan Nintendo Entertainment Systemille ja muille 80- ja 90-lukujen konsoleille tehtyjä shoot ’em up -peligenren klassikoita Gradiuksesta lähtien. Emme kuitenkaan halunneet tehdä pelistä pelkästään retropelikokemusta pikselipainoitteisella sprite-grafiikalla vaan toimme graafiseen ulkoasuun nykyajan tuulahduksen muun muassa Trinen hengessä.

Tämä opinnäytetyö kertoo Royal Crow Squadronin tekemisestä teknisten ratkaisujen tasolla. Aloitan kertomalla yleisesti peliohjelmoinnista, ja sen eroista yleiseen ohjelmointiin. Tämän jälkeen keskityn mobiilipelien maailmaan sekä kerron pelinteossa käytetystä työkalusta, Unitysta. Opinnäytetyön lopussa käydään läpi Royal Crow Squadronin tekovaiheet, ongelmakohdat ja ajatukset siitä, mitä peliin vielä tulisi tehdä.

2 TAUSTAA

Opinnäytetyön aiheena on teknisesti suunnitella ja toteuttaa 2.5D-mobiilipeli ensisijaisesti Android-alustalle käyttäen Unity-pelinkehitystyökalua. Käsiteltävä peli on Royal Crow Squadron -niminen sivustapäin kuvattu lentotaistelupeli, joka sijoittuu toista maailmansotaa mukailevaan konfliktiin fiktiivisessä maailmassa. Pelin tekijätiimiin kuuluu itseni lisäksi Heikki Mäkelä, joka vastaa pelin visuaalisen ilmeen suunnittelusta ja toteutuksesta, ja Elina Koskunen, joka tekee peliin 2D- ja konsepti-grafiikkaa. Tärkeää on huomata, että pelin budjetti on alustavasti nolla euroa, mikä heijastuu vahvasti kehityksessä tehtyihin päätöksiin. Opinnäytetyön lopputuote on pelin beta-versio, mutta lopullisena tavoitteena on saada se valmiiksi ja julkaistuksi Google Playssa.

2.1 Tavoite ja tarkoitus

Opinnäytetyön tavoitteena on hankkia tietoa ja kokemusta pelin teknisestä suunnittelusta ja toteutuksesta Unity-pelimoottorilla.

Opinnäytetyön tarkoituksena on teknisesti suunnitella ja toteuttaa peli Unity-pelimoottorilla ensisijaisesti Android-alustalle.

2.2 Peliohjelmointi

Kun puhutaan pelin tai oikeastaan minkä tahansa ohjelman teknisestä toteuttamisesta, yleensä ennemmin kuin myöhemmin tulee vastaan ohjelmointi. Peliohjelmointi kuitenkin eroaa monesti dramaattisilla tavoilla esimerkiksi toimistosovelluksien ohjelmoinnista, mutta yhtäläisyydet ovat myös ilmiselviä. Tietoja tallennettaan eri muuttujiin, olioita alustetaan ja metodeja kutsutaan ihan samalla tavalla kuin muutenkin ohjelmoinnissa. Se mikä toimistosovellusten ohjelmoijille tulee yleensä vieraana ja massiivisena uutena asiana peleissä on jatkuvasti päivittyvä ympäristö, jonka elementeistä valtaosa on interaktiivisia. Esimerkiksi monet toimistosovellukset tyytyvät odottamaan käyttäjien antamia syötteitä, mutta valtaosa peleistä toimii silmukassa, joka päivittää pelitapahtumia odottamatta käyttäjän toimia. Yhden silmukan aikana peli

menee eteenpäin tietyn verran: animaatiot etenevät, äänet kuuluvat, pelin sisäiset objektit liikkuvat (Nystrom). Pelit ovatkin myös toimistosovelluksia laajempia kokonaisuuksia: niissä yhdistyy usein suuri määrä videoita, 2D-grafiikkaa, 3D-malleja, ääntä, animaatiota ja ohjelmakoodia (McShaffry, Graham, 2013, 15). Peliohjelmoijan tehtävä on saada rakennettua näistä asioista kokonaisuus.

Läheskään kaikkea ei onneksi tarvitse tehdä itse, sillä jopa AAA-pelejä ohjelmoitaessa käytetään jonkun verran valmiita luokkakirjastoja sekä ohjelmointirajapintoja. Windows-käyttöjärjestelmälle suunnatuissa peleissä näistä ylivoimaisesti suosituin on Microsoftin DirectX-ohjelmointirajapinta, joka tarjoaa käyttöjärjestelmän tekijän suunnittelemat ja monia kertoja testatut kontrollit grafiikkaan ja ääneen ja moneen muuhun pelien tarvitsemaan ominaisuuteen. (McShaffry, Graham, 2013, 7)

Hyödyllisyydestään huolimatta edes DirectX ei tee peliä valmiiksi. Koska merkittävä osa peleistä ovat jonkinasteisia simulaatioita oikeasta maailmasta, joka toimii tiettyjen sääntöjen mukaan, nämä säännöt ovat myös ohjelmoitava peliin itseensä. Nämä säännöt voivat olla sellaisia yksinkertaisia asioita kuin esimerkiksi ettei lattialle astuva pelihahmo tipahda lattiasta läpi. Kun nämä asiat ovat valmiita, voidaan peliin oikeastaan vasta alkaa tehdä sisältöä.

Tätä ”säännöstöä” voidaan kutsua yleisesti pelimoottoriksi, joka on siis ohjelmistokehys peleille. Pelimoottorien kehittäminen on monesti uskomattoman aikaa vievää työtä, mutta silti sitä tehtiin hyvin paljon ennen 2000-lukua, jolloin pelien kehityskaaret saattoivat hyvinkin venyä jopa viiden vuoden mittaisiksi. Valmiin pelimoottorin käyttäminen säästää siis useita miestyötunteja ja vapauttaa peliohjelmoijat tekemään varsinaista pelin ohjelmointia pelimoottorin ohjelmoinnin sijaan.

2.3 Mobiilipelit

Ensimmäisenä merkittävänä mobiilipelinä voidaan pitää Nokian vuonna 1997 tiettyjen puhelimien mukana tullutta Snakea, joka myöhempien versioidensa kanssa oli parhaimmillaan asennettu noin 350 miljoonalle laitteelle (Nokia, 2009). Snake oli kaksivärinen yksinkertainen 2D-peli, jossa ohjattiin matoa muistuttavaa pötkylää keräämään ruudulle ilmestyneitä neliöitä. Melkein kahdessakymennessä vuodessa mobiilipelit ovat kuitenkin kokeneet huiman kehityskaaren, jonka toisena päänä voidaan

pitää iOS:lle Chairin vuonna 2013 julkaisemaa Infinity Blade III:a, joka fotorealistisen grafiikkansa kanssa on kuin esittelykappale siitä mihin Epicin Unreal Engine 3 -pelimoottori oikein pystyy mobiilialustalla (Fahey, 2013).

Voidaan kuitenkin sanoa, ettei mobiilipelien kirkkain kärki ole lähtenyt mihinkään kotimaastaan: suomalainen Supercell on tehnyt muutamiakin suosittuja mobiilipelejä, joista suosituin, Clash of Clans, tuottaa noin miljoona dollaria vuorokaudessa (Thinkgaming, 2014). Myös jo johdannossa mainittu Rovio on edelleen merkittävä tekijä mobiilipelimarkkinoilla.

Clash of Clans – kuten moni muukin mobiilipeli – luottaa ansaitamallissaan free to play -ratkaisuun, mikä tarkoittaa, että peli on täysin ilmainen pelata ja käyttää, mutta siinä on mahdollista avata uusia ominaisuuksia tai vauhdittaa etenemistä käyttämällä rahaa (Sinha, 2014). Muita mahdollisuuksia mobiilipeleistä ansaitsemiselle on niiden myyminen rahaa vastaan tai mainosten asettaminen peliin sisälle. Näistä kuitenkin nimenomaan free to play tuntuu olevan kannattavin menetelmä, sillä jo vuonna 2011 sitä käyttävät pelit ohittivat tuotossaan maksulliset pelit Applen App Storessa (Valadares, 2011). Itse emme kuitenkaan jääneet Royal Crow Squadronin kohdalla ansaitamenetelmää, sillä tarkoituksemme ei ollut tienata rahaa tällä pelillä.

Videopelit ovat yleisesti tällä hetkellä todella kannattava toimiala: sen markkinoiden arvo oli vuonna 2013 93 miljardia dollaria, josta mobiilipelien osuus oli 15 miljardia dollaria (Gartner, 2013). Asiantuntijoiden mielestä se on kuitenkin vain alkua, sillä mobiilipeleien markkinoiden epäillään kasvavan 60 miljardin dollarin arvoisiksi vuoteen 2017 mennessä (Brightman, 2014). Euroissa se on noin 43,4 miljardia, ja vertailun vuoksi voidaan sanoa, että Suomen valtion vuoden 2014 talousarvion loppusumma on noin 54,1 miljardia euroa (Eduskunta, 2013).

3 MENETELMÄT

Royal Crow Squadronin suunnittelun alussa oli siis muutamia merkittäviä kysymyksiä. Koko prosessin määrittelevä ratkaisu tulisi olemaan, että mikä on pelin ensisijainen kohdealusta, sillä mobiilikäyttöjärjestelmiä on useita, ja jokaiselle niistä voi tehdä pelejä. Nykyään on kuitenkin myös mahdollisuudet kehittää mobiilisovelluksia hyvin alustariippumattomasti, mikä asettaa alustan valinnan lopullisuuden kyseenalaiseen valoon. Onko loppujen lopuksi järkeä tehdä sovellus natiivina, jos samalla vaivalla saisi aikaan usealla eri alustalla toimivan sovelluksen?

Kohdealustaa koskevien päätösten jälkeen on ratkaistava myös työkalujen käyttö. Jos päättää lähteä alustariippumatoman kehityksen linjalle, ei tule kuuloonkaan tehdä peliä natiivina. Valmiiden pelimoottoreiden tai -ohjelmointikehyksien käyttö mahdollistaa sulavan alustariippumattoman kehityksen, mutta myös helpottaa ylipäätään pelin teknistä toteutusta huomattavasti.

3.1 Alusta

Alustan valinnan tärkein kriteeri oli laajalle levinnyt käyttö: mikäli tarkoituksena on tuoda pelillä ilmi referenssinä tekijöidensä osaamisesta, laajassa kohderyhmässä ei ainakaan ole haittaa.

3.1.1 Android

Android on Googlen kehittämä Linux kerneliin pohjautuva mobiilikäyttöjärjestelmä, jota käyttävä ensimmäinen puhelin julkaistiin vuonna 2008 (Cassavoy, 2008). Sitä käyttävien älypuhelimien myyntiosuus vuoden 2013 kolmannella neljänneksellä oli 81,9 % (Gartner, 2013) ja internet-selauksessa käytetty osuus vuoden 2013 viimeisellä neljänneksellä 33,4 % (Netmarketshare). Tällä hetkellä uusin Android-versio on KitKat-lempinimeä käyttävä 4.4.2 (Android Api Guide).

Kehittäjän näkökulmasta Androidilla on yksi merkittävä haittapuoli: laitekannan fragmentaatio. Vuoden 2013 kesäkuussa opensingal.comin laatiman raportin mukaan vielä käytössä olevia eri Android-laitteita on noin 12 000, jotka käyttävät kahdeksaa eri

Androidin versiota. Periaatteessa tämä tarkoittaa sitä, että kehittäjän on varauduttava suureen määrään erikokoisia näyttöjä ja eritehoisia prosessoreita ja niin edelleen. Totuus ei kuitenkaan ole täysin tämä, sillä Opensingal.comin artikkeli mainitsee, että fragmentaatio on hyvinkin kontekstista riippuvaista. Käytännössä tämä tarkoittaa, että pohtiessaan Android-sovelluksensa ominaisuuksia kehittäjän kannattaa ottaa huomioon kohdemarkkinat. (Opensingal, 2013)

Android-sovellusten ensisijainen jakelukanava on ehdottomasti Google Play, jossa on tällä hetkellä 1,1 miljoonaa sovellusta ladattavissa joko maksusta tai ilmaiseksi (Appbrain, 2014). Google Playhyn voi rekisteröityä kehittäjäksi kuka tahansa tuetussa maassa asuva, Google-tilin omistava henkilö 25 dollarin (noin 20 euron) rekisteröintimaksua vastaan (Google Support).

3.1.2 iOS

iOS on Applen mobiililaitteissa ja televisioissa oleva käyttöjärjestelmä, joka julkaistiin vuonna 2007 iPhone-älypuhelimessa. iOS:ää ei lisensoida käytettäväksi muiden valmistajien laitteissa joten se on käytössä vain kahdessakymmenessä eri laitteessa (Blakespot, 2013). Tästä huolimatta iOS:ää käyttävät mobiililaitteet ovat todella suosittuja: niiden myyntiosuus vuoden 2013 kolmannella neljänneksellä oli 12,1 % (Gartner, 2013) ja internet-selauksen osuus vuoden 2013 viimeisellä neljänneksellä 54,9% (Netmarkethsare) iOS on ehtinyt versioon 7.0.5, joka julkaistiin 29. 1. 2014 (Apple, 2014).

iOS-sovellusten ainoa jakelukanava on Applen oma App Store, josta käyttäjät voivat ladata niin ilmaisia kuin maksullisia sovelluksia iTunes-ohjelman kautta. Tammikuussa 2014 App Storessa oli noin 1,1 miljoonaa ladattavaa sovellusta (148Apps, 2014). Kehittäjälisenssi App Storeen maksaa kuitenkin 99 dollaria (noin 70 euroa) vuodessa (Apple Developer).

3.1.3 Muut alustat

Yli prosentin myyntiosuuksiin vuoden 2013 kolmannella neljänneksellä pääsi Androidin ja iOS:än lisäksi vain Microsoftin Windows Phone 3,6 %:n osuudella ja Blackberry Limitedin BlackBerry 1,8 %:n osuudella (Gartner, 2013). Windows Phone

Store ylitti joulukuussa 2013 kahdensadantuhannen sovelluksen rajan (Hachman, 2013), mutta siitä huolimatta sitä pidetään hyvinkin merkityksettömänä tekijänä varsinkin mobiilipelimarkkinoilla.

3.1.4 Valinta

Lukuja tarkkailemalla alustan valinta on selkeä: Android-laitteet hallitsevat markkinoita vahvasti. Kuitenkin on mietittävä mitä käytännössä tarkoittaa nettiselauksen osuuden erot alustojen välillä. Voiko nettiselauksen kallistuminen täpärästi iOS-laitteiden puolelle tarkoittaa sitä, että isolla osalla Android-käyttäjistä ei kuitenkaan ole käytössään nettiyhteyttä puhelimeensa? Tämä heijastelee uutisia, joissa kerrotaan Android-laitteiden markkinakasvun olevan kehittyvien markkinoiden ansiota (Alfreds, 2014). Nettiyhteyden puuttuminen mobiililaitteesta hankaloittaa kuitenkin suuresti minkäänlaisen ulkoisen sovelluksen asentamista kyseiselle laitteelle.

Android valikoitui ensisijaiseksi kohdealustaksi levinneisyytensä takia. Nykyisessä tilanteessa ei kuitenkaan ole mitään mieltä sulkea potentiaalisia käyttäjiä heti kehityksen alkuvaiheessa pois kohdeyleisön joukosta. Tästä johtuen kehityksen ohjenuoraksi valikoitui alustariippumattomuus, mikä heijastui suuresti myös työkalujen valinnassa.

3.2 Työkalu

Alustariippumaton kehitys tuo mukanaan tarpeen pelimoottorille tai ohjelmistokehykselle, joka kääntää kirjoitetun koodin eri alustoille. Koska Royal Crow Squadron yhdistelee 3D- ja 2D-grafiikkaa, työkalun on myös syytä tukea näitä kumpaakin. Lisäksi projektin käytössä olevat resurssit ovat rahallisesti olemattomat, mistä johtuen käytettävän pelimoottorin on oltava käytännössä ilmainen.

3.2.1 Oma pelimoottori

Oman alustariippumattoman pelimoottorin kehittäminen on vaihtoehto. On kuitenkin otettava huomioon, että se on niin teknisesti kuin ajallisestikin hyvin vaativaa. Royal Crow Squadronin kaltaisessa projektissa, jossa on yksi ohjelmoija ja rajallisesti aikaa, ei ole mielekästä kirjoittaa omaa pelimoottoria.

3.2.2 Gameplay3D

Gameplay3D on avoimen lähdekoodin, alustariippumaton 3D-ohjelmistokehys, jolle ohjelmoidaan käyttäen C++-kieltä. Se käyttää grafiikkarajapintanaan OpenGL ja OpenGL ES -rajapintoja. Gameplay3D:hen on rakennettu sisälle merkittävä määrä mobiilikontrollereita, sillä se on pääasiassa suunnattu mobiilikehitykseen, ja tällä hetkellä sillä onkin tuki Android, iOS ja BlackBerry -mobiilikäyttöjärjestelmien lisäksi myös OS X, Microsoft Windows ja Linux -työpöytäkäyttöjärjestelmille. Gameplay3D:n merkittävänä etuna on sen avoin lähdekoodi, jonka ansiosta kehittäjä tietää tasan tarkalleen miten peli teknisesti toimii sekä lisätä siihen kokonaan omia ominaisuuksiaan kuten tuen uusille käyttöjärjestelmille. (Github, 2014)

3.2.3 LibGDX

Suosittu LibGDX on niin ikään avoimen lähdekoodin, alustariippumaton peliohjelmistokehys, jolle voi kehittää sekä 2D- että 3D-pelejä. Se on hyvin suosittu: kotisivuilla listataan tällä hetkellä 825 LibGDX:llä tehtyä peliä. LibGDX:lle kirjoitetaan koodi javalla. Avoimen lähdekoodin projektina siihen pätevät samat hyvät puolet siltä osin kuin Gameplay3D:kin. LibGDX on kuitenkin lähtenyt 2D-peliprojekteista eikä sen 3D-tuki ole yltänyt vielä aivan 2D:n tasolle. (BadLogicGames)

3.2.4 Unity

Todella suosittu Unity on Unity Technologiesin kehittämä pelimoottori, jossa on sisäänrakennettu graafinen kehitysympäristö. Siinä voi natiivisti kehittää sekä 2D- että 3D-pelejä usealle eri kohdealustalle, joihin kuuluu myös Android. Se tukee koodin kirjoittamisessa JavaScriptiä, C#:ia ja Boota. Unity on kuitenkin yksityisen yrityksen yrityksen omistama pelimoottori, mistä johtuen sen lähdekoodi ei ole avointa ja se ei ole aivan kokonaan ilmainen. (Unity3D)

3.2.5 Valinta

Ensinnäkin oman pelimoottorin kirjoittamista ei edes harkittu, koska se vie yksinkertaisesti liikaa aikaa, ja tavoitteena on saada kokemusta peliohjelmoinnista eikä pelimoottorin ohjelmoinnista.

Muista vaihtoehdoista valitseminen on kuitenkin hieman vaikeampaa: kaikilla niistä saisi aikaan halutun pelin, joka toimii halutuilla alustoilla. Tärkeimmäksi kriteeriksi valikoitui helppokäyttöisyys.

Jos oletetaan tilanne, jossa käyttäjä aloittaa peliprojektin Gameplay3D:llä, LibGDX:llä ja Unitylla. Gameplay3D ja LibGDX käyttäytyvät kummatkin hyvin samalla tavalla: edessä on valitsemansa tekstieditorin tai kehitysympäristön tyhjä ruutu, johon kehittäjän tulee alkaa kirjoittaa koodia. Kumpikaan ohjelmistokehys ei tarjoa missään vaiheessa tämän sofistukoituneempia työkaluja itse pelinkehittämiseen. Mikäli kehitystiimille on oleellista, että muutkin kuin ohjelmoijat voivat osallistua kehitykseen esimerkiksi kenttäeditoinnin muodossa, siihen täytyy rakentaa itse työkalut. Pienemmissä projekteissa tämä voidaan kiertää käyttämällä 3D-mallinnusohjelmaa kenttäeditorina, mutta sen rajat tulevat nopeasti vastaan (McShaffry, Graham, 2103, 17).

Unityn suurin etu kilpaileviin vaihtoehtoihin on sen sisäänrakennettu editori ja sen valmiit ominaisuudet, mitkä tekevät oikeastaan kaikesta pelintekemisessä helpompaa. Unityn editori on kuin valmis kenttäeditori, mutta siinä voi kenttien lisäksi rakentaa kaiken muunkin peliin kuuluvan aina materiaalien muokkaamisesta äänien editointiin. Sen helppokäyttöisyyden ansiosta peliä voivat tehdä muutkin projektiin osallistujat kuin pelkästään ohjelmoijat, mutta se helpottaa suuresti myös ohjelmoijan työtä: heidän ei tarvitse esimerkiksi visioida kolmiulotteista maailmaa pelkän tekstin perusteella vaan se on koko ajan näkyvissä editorissa.

Unityssa on kuitenkin myös huonoja puolia verrattuna kilpailijoihin. Sen lähdekoodi on salaista, mistä johtuen kehittäjä ei oikeasti tiedä mitä pelissä tapahtuu ”pinnan alla”. Jos joku Unityn ominaisuuksista toimii huonosti tai on kokonaan rikki, se toimii silloin huonosti tai on rikki niin kauan kunnes Unity Technologies tekee siihen korjauksen. Se ei myöskään ole täysin ilmainen, mutta ilmaisversiota saa käyttää kuitenkin niin kauan kunnes sitä käyttävän yhteisön liikevaihto ylittää tietyn rajan. Royal Crow Squadronin päämääränä ei kuitenkaan ole tehdä kehittäjilleen rahaa joten tämä ei ole ongelma.

4 UNITY

Unity on Unity Technologiesin kehittämä pelimoottori, jossa on sisäänrakennettu kehitysympäristö. Unitya käytetään pääasiassa pelinkehittämiseen yhdelle tai useammalle sen tukemista alustoista. Varsinkin indie-kehittäjät ovat ottaneet sen omakseen, mistä kertoo muun muassa se, että vuoden 2014 Global Game Jameilla tehdyistä peleistä yli kahdessa tuhannessa käytettiin Unitya (Unity Technologies, 2014).

Ensimmäinen versio Unity-pelimoottorista julkaistiin vuonna 2005 puhtaana tarkoituksenaan antaa Applen OS X -käyttöjärjestelmälle natiivi pelimoottori. Tästä huolimatta tuki Microsoftin Windows -käyttöjärjestelmälle ja nettiselaimille seurasi hyvin pian. Yksi merkittävimmistä käännekohdista pelimoottorin historiassa oli Applen App Storen julkaisu, mikä avasi portit Unityn käytössä mobiilikehityksessä. (Brodkin, 2013)

Vuonna 2014 Unity on ehtinyt versioon 4.3.4 ja sillä on tuki julkaisulle Windowsille, OS X:lle, Linuxille, useille selaimille, Wii U:lle, PS3:lle, Xbox 360:lle, Flashille, Windows Phone 8:lle, iOS:lle, Androidille ja BlackBerry 10:lle. Versiosta 4.3 lähtien Unity on mahdollistanut natiivin 2D-pelinkehityksen, mikä tipautti 3D:n pois pelimoottorin nimestä (Goldstone, 2013). Unity perusversio tukee kehitystä kaikille mobiilialustoille sekä on täysin ilmainen käyttäjille, joiden liikevaihto oli alle satatuhatta dollaria edellisenä tilivuonna, mikä tekee siitä erinomaisen työkalun indie-kehittäjille. Pro-lisenssi tuo käyttöön useita uusia ominaisuuksia, mutta se maksaa joko 1500 dollaria kertamaksuna tai 75 dollaria kuukaudessa. Pro-lisenssit mobiilialuistoille maksaa vielä tämän lisäksi saman verran yhtä alustaa kohden lukuun ottamatta Windows Phone 8:aa, jonka pro-lisenssi tulee ilmaiseksi tavallisen pro-lisenssin mukana. Konsolikehityksestä kiinnostuneiden tulee sen sijaan ottaa yhteyttä Unity Technologiesin Sales-tiimiin. (Unity3D)

4.1 Työkalun keskeiset elementit

Unity antaa kehittäjän käyttöön monia AAA-peleistä tuttuja työkaluja. Grafiikkamoottori on kohdealustasta riippuvainen, mutta esimerkkinä ajantasaisesta kehityksestä versiosta 4.x lähtien Unitylla kehitetyissä Windows-peleissä on ollut mahdollista käyttää DirectX 11 -multimediarajapintaa (Unity Manual: DirextX 11, 2013). Fysiikkamoottorina Unity-pelissä on mahdollista käyttää joko NVIDIAN PhysX:ää, jota on käyttänyt muun muassa Batman: Arkham Cityssä (Bayer, Stöwer & Sauter, 2011), tai 2D-peleistä tuttua Box2D:tä, joka sen sijaan tunnetaan muun muassa Angry Birdsistä (Kumparak, 2011). Unityn animaatiotekniikasta vastaa Mecanim, jota kehitettiin itsenäisessä yhtiössä ennen kuin Unity Technologies osti sen vuonna 2011 (Yahoo! Finance, 2011). (Unity3D)

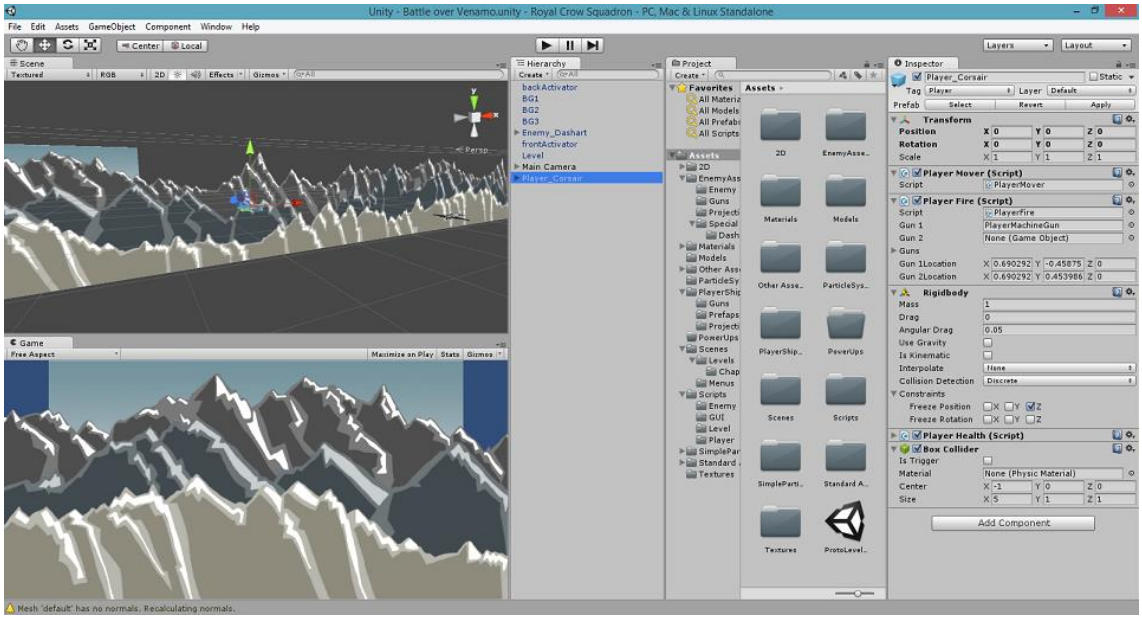
Itse editorina Unity on hyvin samanlainen graafisella käyttöliittymällä varustettu 3D-ympäristö kuin monet mallinnusohjelmat. Unity-pelit rakentuvat scene-tiedostoista, jotka ovat kokoelma toisiinsa liittyneitä objekteja, jotka toimivat yhdessä. Yksinkertaisinta onkin ajatella, että jokainen scene on yksi kenttä (Unity Manual: Creating Scenes, 2013). Kuitenkin tärkein Unity-pelien peruselementti on gameobject. Ohjelmointinäkökulmasta voidaan ajatella, että yksi gameobject scenessä olisi yksi olio, mutta tilanne ei kuitenkaan ole aivan tämä. Unity nimittäin suosii hyvin komponenttikeskeistä ajattelua: jokainen gameobject sisältää komponentteja, jotka määrittävät gameobjectin eri ominaisuuksia. Komponentteja ei ole mahdollista luoda koodillisesti (Unity Script Reference: Component). Gameobjectien suurin etu on niiden mahdollisuus tallentaa prefabeiksi editorin resursseihin, jolloin niitä on mahdollista lisätä sceneen haluttu määrä haluttuihin paikkoihin tai vaikka antaa referenssiksi muille gameobjecteille. (Unity Script Reference: GameObject)

Erilaisia komponentteja on monia. Tärkeimpänä näistä voi pitää transformia jo siinä mielessä, että se on ainoa komponentti, mikä on oletuksena kiinni kaikissa gameobjecteissa. Transform määrittää gameobjectin sijainnin scenessä, suhteellisen koon ja kulman (Unity Documentation: Transform). Jokaisella näistä on x, y ja z -arvot, jotka ovat yhdessä luokka nimeltään vector3. Vector3 on Unityssa ylipäätään monikäyttöinen luokka, sillä se määrittää sijaintipisteen lisäksi myös suuntia. Vector3:n täydellinen ymmärtäminen vaatii myös hieman matemaattista tietoa vektoreista, sillä suuntavektoreilla on aina suunnan lisäksi myös pituus, jota käytetään eri asioihin

riippuen vector3:n käyttötarkoituksesta (Unity Script Reference: Vector3). Muita komponentteja on esimerkiksi renderer, joka määrittää gameobjectin ulkonäön pelissä, collider, joka määrittää gameobjectin törmäyspinnat muiden gameobjectien collidierien kanssa, sekä rigidbody, joka hallitsee gameobjectin liikkeitä fysiikan kautta (Unity Documentation: Components). Jokainen gameobjectiin liitetty komponentti ei kuitenkaan tarvitse olla päällä koko aikaa vaan niiden päällä oloa voidaan säädellä sekä editorissa että koodillisesti.

Gameobjectit voivat olla myös muiden gameobjectien alaisena, jolloin puhutaan parent-child -suhteesta. Tämä on erityisen hyödyllistä esimerkiksi silloin, kun halutaan joidenkin gameobjectien toimivan itsenäisesti, mutta silti osana suurempaa kokonaisuutta. Esimerkiksi ihmishahmo-gameobjectin childeina voisi olla ihmishahmon ruuminosat tai hänen kädessään olevat asiat. On tärkeä muistaa, että childinä olevat gameobjectit liikkuvat, kääntyvät ja kasvavat suhteessa parent-gameobjectiin.

4.2 Käyttöliittymä

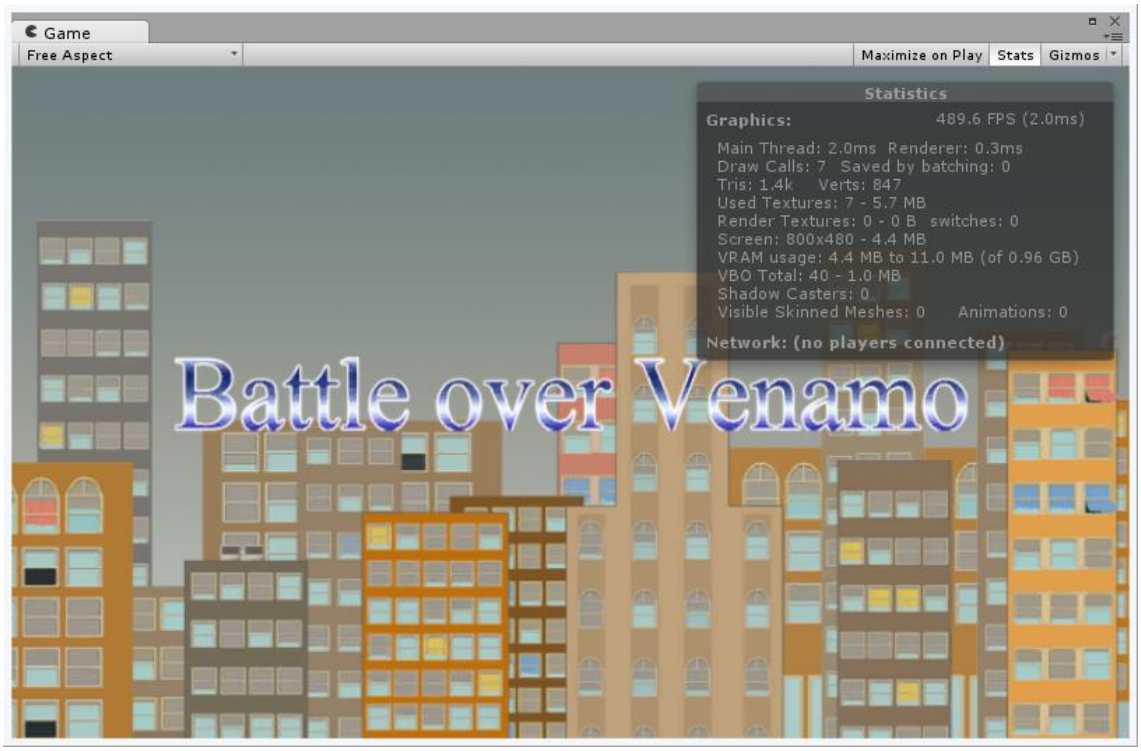


KUVA 1 Näkymä Unityn editorista

Unityssa on käytössä tehokas ja intuitiivinen 3D-ympäristössä toimiva editori, joka mahdollistaa lähes kaikkien pelin osa-alueiden hallinnan graafisen käyttöliittymän kautta. Kuvassa yksi voi nähdä editorin monikäyttöisessä 1-to-3 -asettelussa, jossa kaikki editorin keskeiset elementit ovat helposti saatavilla yhdellä ruudulla. Jokainen elementeistä on halutessaan irrotettavissa ruudusta ja jopa kopioitavissa, mikä on erityisen hyödyllistä useamman näytön kokoonpanossa.

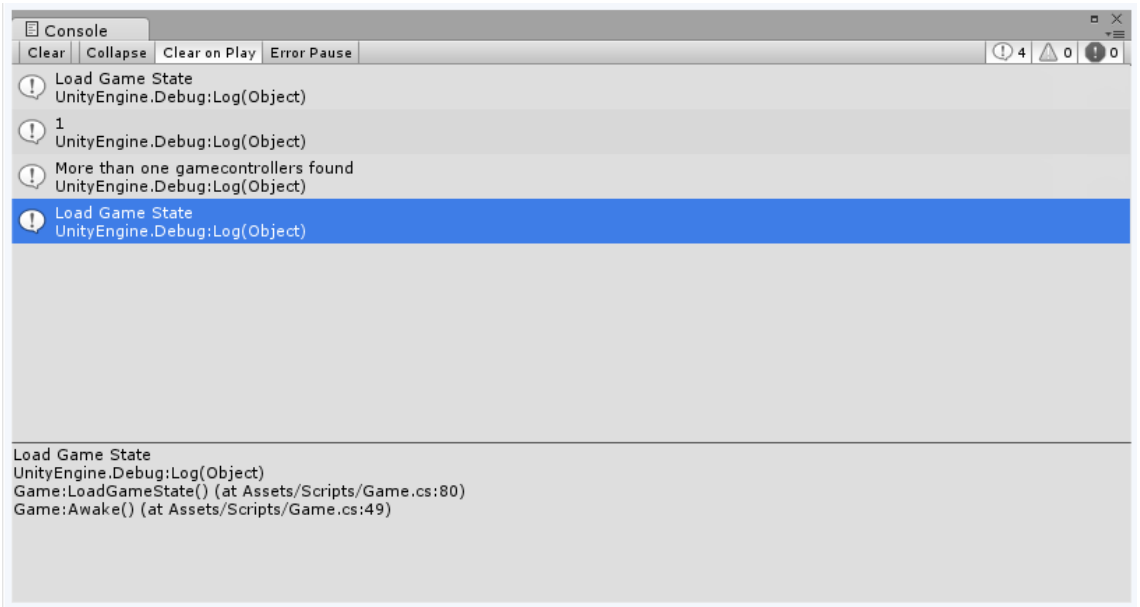
Kuvan yksi ylävasemmalla on Scene-ikkuna, jossa voi vapaasti editorin sisällä tutkia auki olevaa sceneä. Scene-ikkunassa näkyy myös paljon sellaista tietoa scenestä, jota ei näy itse pelissä. Näistä esimerkkinä voi pitää kuvassa yksi Scene-ikkunan keskellä näkyviä kolmea nuolta, jotka näyttävät valitun gameobjectin suunta-akseleita.

Scene-ikkunan alla on Game-ikkuna, jossa näkyy pelinäkymä. Tämä näkymä on käytännössä se, mitä scenessä olevat kamera-gameobjectit kuvaavat. Peliä ajettaessa editorissa Game-ikkuna on myös se, missä peli alkaa tapahtua. Kuvassa yksi tämä näkymä ei vastaa täysin haluttua, sillä se ei ole oikeassa resoluutiossa.



KUVA 2 Tarkempi kuva Game-ikkunasta

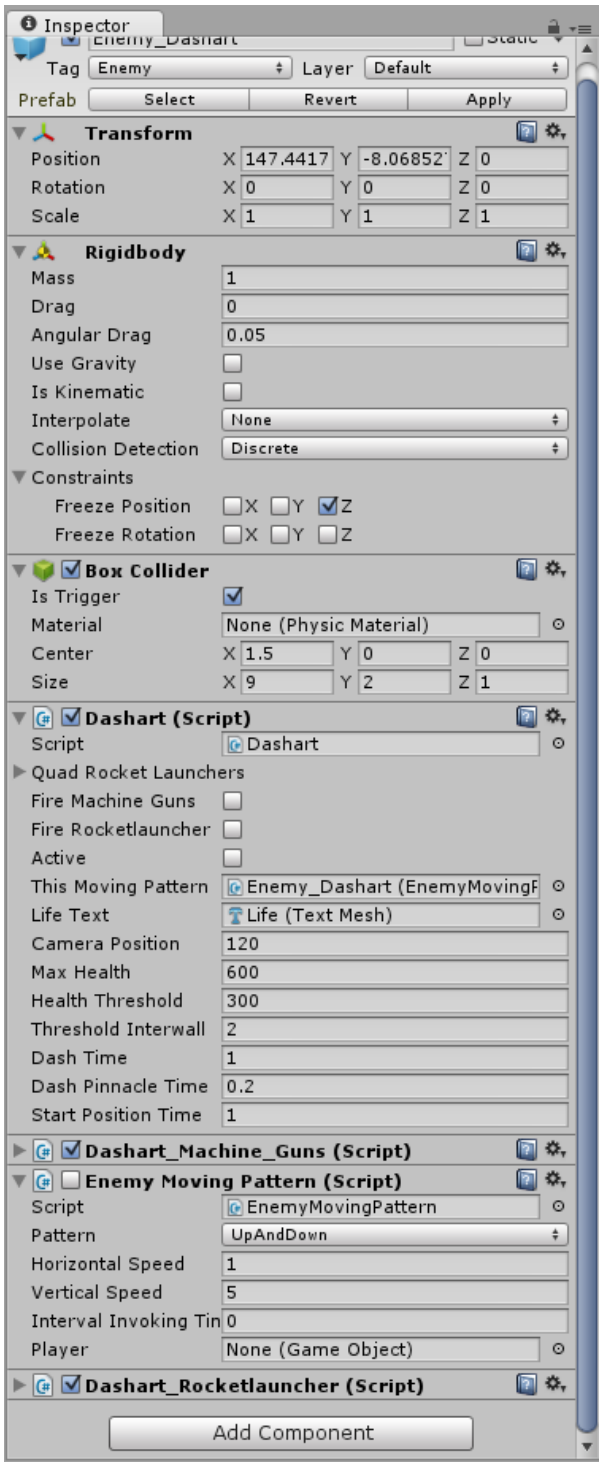
Kuvassa kaksi Game-ikkuna on venytetty haluttuun resoluutioon. Statistics-ikkunan ollessa näkymässä voidaan myös tarkkailla pelin vaatimaa suoritustehoa.



KUVA 3 Editorin Console-ruutu

Editorin alapalkissa on näkyvissä uusin konsoliviesti. Suurennettuna konsoli näyttää kuvan kolme mukaiselta erilliseltä ikkunalta. Unity-kehittäjälle konsoliruutu on paras

ystävä, sillä se näyttää koodin kääntämisessä että pelin ajon aikana tapahtuneet virheet ja varoitukset sekä koodiin kirjoitetut debug-viestit kuten kuvassa kolme on tehty.



KUVA 4 Editorin Inspector-ikkuna

Scene ja Game -ruutujen oikealla puolella on Hierarchy-ikkuna, jossa näkyy kaikki auki olevassa scenessä olevat gameobjectit. Kuvassa yksi voidaan nähdä hierarkianäkymässä sinisellä maalattu gameobjectin nimi, joka merkitsee tällä hetkellä Scene-ikkunassa valittua gameobjectia. Tämän lisäksi osan gameobjectien nimet ovat sinisellä, mikä tarkoittaa, että ne ovat projektin tietoihin tallennettuja valmiita prefabeja.

Kuvassa yksi Hierarchy-ikkunan oikealla puolella on Project-ikkuna, jossa näkyy kaikki auki olevan projektin käytössä olevat assetit ja prefabit, jotka ovat käytettävissä Scene-ikkunaan pelkästään hiirellä klikkaamalla ja raahamalla. Projektin paisuessa pelin assetteja ja prefabeja tulee todella paljon, mikä tekee projekti-ikkunan järjestämisen kansiorakenteeseen lähes välttämättömäksi.

Äärioikealla kuvassa yksi on näkyvissä Inspector-ikkuna, jossa on näkyvissä valitun gameobjectin tiedot ja siinä kiinni olevat komponentit ja niiden tiedot. Inspector-ikkuna on siinä mielessä tärkeä ja hyödyllinen, että kehittäjä voi muuttaa siinä gameobjectin ja sen komponenttien arvoja helposti kajoamatta itse koodiin, mikä on nähtävissä kuvassa neljä. Testaamista helpottamiseksi Inspectorissa voi muuttaa arvoja myös pelin ajon

aikana. Erityisen hyödyllistä on myös huomata, että Inspectorissa voi suoraan muuttaa kehittäjän omien komponenttien, scriptien, public-suojauksella merkittyjä muuttujia. Scripteistä kerrotaan laajemmin seuraavassa osassa.

4.3 Scriptaus

Vaikka gameobjectien toimintaa voikin säädellä hyvin pitkälle valmiilla komponenteilla, tulee hyvin pian vastaan tilanne, jossa ne eivät riitä. Tällöin Unity-kehittäjä haluaa tehdä oman komponenttinsa, jota kutsutaan scriptiksi. Kuten jo aiemmin mainitsin, scriptien kirjoittaminen on hyvin merkittävä osa kaikenkokoisia pelejä.

Unity käyttää scriptien rakentamisessa Mono-sovelluskehystä, joka on avoimen lähdekoodin alustariippumaton vastine .NET-sovelluskehykselle. Scriptejä voi kirjoittaa JavaScriptillä, C#:lla tai Boolla, jotka kaikki pääsevät Monon kautta hyödyntämään .NET-sovelluskehyksestä tuttuja luokkakirjastoja. (Mono)

Unity käyttää JavaScripristä omaa variaatiotaan, joka eroaa itse JavaScriptistä niinkin paljon, että sille on annettu oma pseudonyymi, UnityScript. JavaScriptin tavoin se on kuitenkin dynaamisesti tyypitetty olio-ohjelmointikieli, jonka syntaksi mukailee C-kieltä. UnityScript saattaa tuntua monelle web-kehittäjälle ja aloittelevalle kehittäjälle ensimmäiseltä vaihtoehdolta käytettäväksi ohjelmointikieleksi. (Loewald, 2013)

C# (lausutaan C-sharp) on Microsoftin .NET-sovelluskehykselle kehitetty olio-ohjelmointikieli, jonka syntaksi muistuttaa muita C-kieliä (Microsoft Developer Network). Monelle sovellusohjelmoijataustaiselle kehittäjälle C# on luonnollinen valinta käytetyksi kieleksi, koska esimerkiksi Javan tavoin se on vahvasti tyypitetty kieli. Merkittävänä etuna on myös se, että .NET:in dokumentaatio on kirjoitettu ainoastaan C#:lle.

Boo on Pythonin syntaksin inspiroima olio-ohjelmointikieli, joka on yhteensopiva niin Mono kuin .NET -sovelluskehyksen kanssa (Boo). Se on Unityn käyttämistä ohjelmointikielistä kaikkein vähiten suosittu, sillä keskustelupalstoja selaamalla ei löydy juuri lainkaan kehittäjää, joka käyttäisi Boota. Huhujen mukaan se onkin mukana

vain, koska sen kehittäjä oli aikaisemmin töissä Unity Technologiesilla (Unity Answers).

Mitä ohjelmointikieltä kehittäjän sitten kannattaa käyttää? Jokaisella niistä on käytössään Monon avulla .NET:in luokkakirjastot ja ainakin Unityn oma dokumentaatio. Tärkein puoli valinnassa lieneekin oma mieltymys. Valintaa ei kuitenkaan välttämättä tarvitse tehdä kerralla, sillä joissakin määrin eri kielillä kirjoitetut scriptit voivat keskustella keskenään Unityn sisällä, mikä on hyödyllistä etenkin silloin, kun otetaan käyttöön jonkun ulkoisen osapuolen kirjoittamia scriptejä, joita ei välttämättä ole mahdollista saada haluamallaan kielellä kirjoitettuna. Tästäkin huolimatta on perusteltua, että kaikki samaan projektiin osallistuvat ohjelmoijat käyttäisivät samaa kieltä.

Projektissa valinta osuu C#:iin jo pelkästään sen takia, että .NET:in dokumentaatio on olemassa sille. Lisäksi C#:ssa kirjoitetuissa scripteissä on käytössä paljon ominaisuuksia, joita ei löydy muista vaihtoehdoista. Käyttämällä esimerkiksi lambda expressioneja ja delegaatteja voi kirjoittamastaan koodista saada tehokkaampaa.

```
using UnityEngine;
using System.Collections;

public class Game : MonoBehaviour {

    public enum LevelState {
        fresh,
        tried,
        made,
        locked
    }

    public static Game Singleton = null;

    public LevelState[] levelState = new LevelState[5];

    public float UniversalSpeed = 2.0f;

    public bool SkipIntro = false;

    private bool gameOn;

    public int MainMenuState { get; set; }

    void Awake () {

        if (GameObject.FindGameObjectsWithTag("GameController").Length > 1) {

            Debug.Log("More than one gamecontrollers found");

            Destroy(gameObject);

        }

        else {

            DontDestroyOnLoad(gameObject);

            Singleton = this;

        }

        MainMenuState = -1;

        Screen.SetResolution(800, 480, true);

        LoadGameState();

    }
```

KUVA 5 Esimerkki MonoBehaviour-scriptistä ja Awake-metodista Royal Crow Squadron -pelistä

Huolimatta valitusta ohjelmointikielestä jokainen luotu scripti periytyy kuvan viisi mukaisesti Unityn sisäisestä MonoBehaviour-luokasta, mikä tarkoittaa, että se on yhteydessä Unityn toimintaan (Unity Manual: Scripts). Se mahdollistaa muiden Unityn sisäisten luokkien – kuten jo mainitun vector3:n – sekä metodien toimimisen scriptissä. MonoBehaviourin metodit ovatkin merkittävässä osassa pelin toiminnan kannalta.

Jokainen scenessä oleva gameobject kutsuu scriptiensä Start-metodit sillä hetkellä, kun scriptit laitetaan päälle eli enabloidaan. Tämä tapahtuu riippumatta siitä ovatko gameobjectit valmiiksi scenessä vai luodaanko ne sinne dynaamisesti pelin ajon aikana. Unityssa ei alusteta scriptejä olio-ohjelmoinnin tavoin kutsumalla niiden alustajaa vaan alustus tulisi tarvittaessa suorittaa scriptin Start-metodissa. (Unity Script Reference: MonoBehaviour)

Scenen alussa scenessä olevien gameobjectien scriptien Start-metodit kutsutaan satunnaisessa järjestyksessä. Mikäli jossakin scripitissä halutaan viitata samassa scriptissä olevaan ominaisuuteen, joka on siinä vasta Start-metodin kutsumisen jälkeen, kysymykseen tulee Awake-metodin käyttö. Awake-metodi toimii lähes identtisesti Start-metodien kanssa sillä erotuksella, että se kutsutaan kaikista scripteitä ennen yhdenkään Start-metodin ajamista. Tämä tarkoittaa siis sitä, että uuden scenen alkaessa kaikista siinä olevien gameobjectien scripteistä kutsutaan ensin Awake-metodit satunnaisessa järjestyksessä, minkä jälkeen ajetaan Start-metodit niin ikään satunnaisessa järjestyksessä. On kuitenkin syytä huomioida, että Awake ajetaan scriptin elikaaren alussa siitä huolimatta onko scripti päällä vai ei. (Unity Script Reference: Monobehaviour.Start)

Scriptien tärkein metodi on kuitenkin Update, joka ajetaan päällä olevasta scriptistä jokaisen ruudunpäivityksen yhteydessä. Updatea käytetään scripteissä luomaan niihin kiinnitetyn gameobjectin toiminta pelin aikana. Kaikki koodillisesti säädeltävä liike ja tapahtumat tapahtuvat scriptien Update-metodissa. (Unity Script Reference: Monobehaviour.Update)

FixedUpdate toimii samalla tavalla kuin Update, mutta se kutsutaan ennaltamäärätyin välein jokaisen ruudunpäivityksen sijaan. FixedUpdatea tulisi käyttää Updaten sijaan, kun halutaan vaikuttaa gameobejctien rigidbody-komponentteihin eli tehdä fysiikkaan liittyviä toimintoja pelissä sen ajon aikana. (Unity Script Reference: Monobehaviour.FixedUpdate)

OnGUI-metodia käytetään Unityn oman käyttöliittymäjärjestelmän käyttöliittymäelementtien ja –toimintojen hallitsemiseen. OnGUI:ta kutsutaan jokaisen ruudunpäivityksen yhteydessä tai silloin, kun käyttäjä käsittelee jotain käyttöliittymäelementtiä. Tätäkään metodia ei ajeta mikäli script ei ole päällä. (Unity Script Reference: Monobehaviour.OnGUI)

OnDestroy-metodi on yksi monista MonoBehaviourin On-alkuisista metodeista. Sitä kutsutaan aktiivisesta scriptistä sillä hetkellä, kun gameobject, jossa scripti on kiinni, tullaan tuhoamaan. (Unity Script Reference: Monobehaviour.OnDestroy)

Nämä ovat kuitenkin vain esimerkkejä MonoBehaviour-scriptien käyttömahdollisuuksista. Sillä voi hallita lähes kaikkia pelimoottorin elementtejä ja muodostaa ihan niin monimutkaisia rakenteita kuin olio-ohjelmoinnissa yleensä. Eli vaikka mainitsinkin, ettei scriptit itsessään toimi täysin luokkien tavalla, niiden sisällä voi määritellä uusia luokkia, jotka toimivat ihan olio-ohjelmoinnin oppien mukaisesti.

4.4 Android

Unity tarjoaa sisäänrakennettussa Input-rajapinnossa nimensä veroisesti mahdollisuuden monenlaisten syötteiden antamiseen pelille ajon aikana. Se tarjoaa perinteisten näppäimistö- ja hiirisyötteiden lisäksi myös useita eri keinoja tutkia mobiililaitteen kosketusnäytöllä tapahtuvia kosketuksia sekä useissa mobiililaitteissa olevan sisäisen kiihtyvyysmittarin liikkeitä. Kosketusten tutkiminen antaa tietoja näytölle tapahtuvien kosketusten määrästä, sijainnista näytöllä sekä sen vaiheista. Kiihtyvyysmittari antaa nimensä veroisesti puhelimen kiihtyvyyden kolmen erin ulottuvuuden suuntaisen G-voimissa, joita voidaan käyttää esimerkiksi ohjaustapana peleissä.

Kosketusten ja kiihtyvyysmittarin lisäksi Unityn mobiilitoiminnoissa on mukana mobiililaitteen virtuaalinäppäimistö sekä hiiri-simulaatio. Virtuaalinäppäimistö on varmasti kaikille kosketusnäytöllisiä laitteita joskus käyttäneille tuttu, ja Unityssa se toimii hyvinkin automaattisesti käyttäjän valitessa pelistä elementin, joka vaati tekstisyötteen. Hiiri-simulaatio sen sijaan on hyvin hyödyllinen ominaisuus, joka tulkitsee mobiililaitteen kosketusnäyttöön tapahtuvia kosketuksia hiiren liikkeiksi ja sen nappien painalluksiksi. Teoriassa tämä tarkoittaa, että mobiiliympäristöön voi kääntää lähes ilman muutostöitä hiiriohjauksella toimivia pelejä, mutta käytännössä se ei ole täysin ongelmaton ratkaisu. (Unity Manual: Input)

Ylipäätään kehittäjä voi luottaa, että ylläolevat ja lähes kaikki muutkin mobiililaitteiden ominaisuudet toimivat samalla koodilla sekä iOS- että Android-käännöksissä peleissä. Joitakin poikkeuksia tietysti on, mutta niitä ei kannattane alkaa syvällisemmin eritellä.

Android-peliä teknisesti toteuttaessa kehittäjän tulee kuitenkin erityisesti ottaa huomioon tiettyjä seikkoja. Näistä merkittävässä osassa on tietysti Androidin oma kehitysympäristö, sillä Unity ei pysty itse pakkaamaan peliä Androidin tukemaan apk-formaattiin. Kyseessä ei kuitenkaan ole mitenkään monimutkainen operaatio, sillä

Android SDK -kehitysympäristön lisäksi tarvitaan vain polunmäärittely Unityssa, minkä jälkeen Unity huolehtii itse lopusta. (Unity Manual: Android)

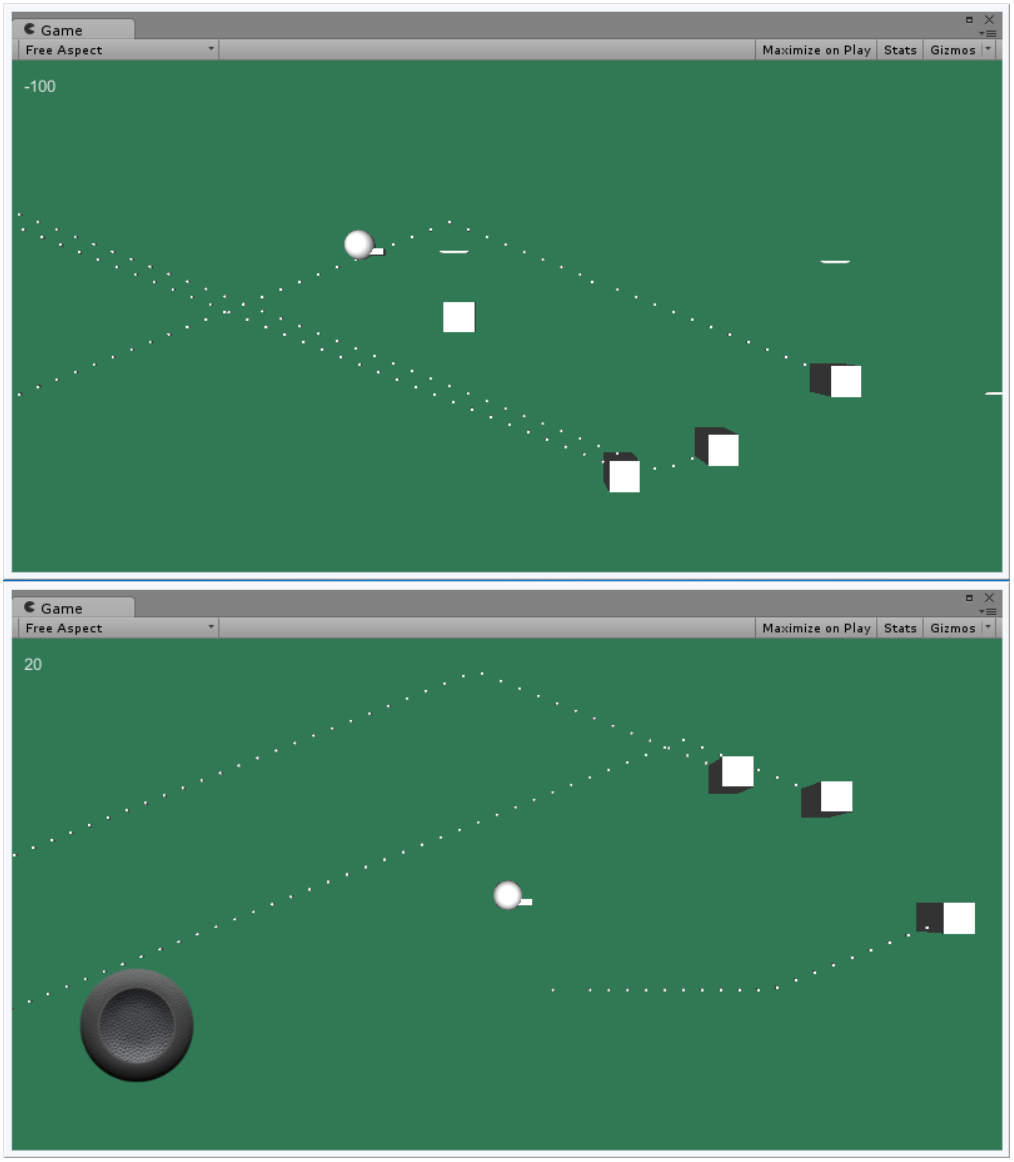
Niin Unity-peleissä kuin missä tahansa muussakin Android-sovelluksessa on tietysti otettava huomioon alustan merkittävä fragmentaatio, mistä on mainittu jo tässäkin opinnäytetyössä. Unityn versiolla 4.3.4 kehitetyt pelit pitäisi periaatteessa toimia Androidin versiossa 2.3.1 ja sitä uudemmissa, mikä onkin aika pitkälle ainoa asia mikä on luvattavissa. Kyseessä on kuitenkin hyvin jopa 3D-grafiikkaan pystyvä pelimoottori ja on mahdotonta luvata sen toimivan kovinkaan heikkotasoisissa puhelimissa, joita löytyy Android-markkinoilta runsaasti. Parasta mitä Unityn Android-kehittäjälle voikin sanoa on, että pohtii alusta lähtien pelinsä kohderyhmää ja miettii sen perusteella mitä ominaisuuksia ottaa pelissään käyttöön. (Unity Manual: Mobile Advanced)

5 SUUNNITTELU

KUVA 6 Kuvankaappaus NES-konsolia emuloivasta John NES Lite -sovelluksesta ajamassa peliä Gradius (1985)

Royal Crow Squadronin konseptin suunnittelussa lähdettiin hyvin yksinkertaisista ajatuksista liikkeelle. Tiesimme millaisen pelin halusimme tehdä, miltä se tulisi näyttämään ja teeman. Testasimme konseptin toimivuutta mobiilipelinä pelaamalla esikuvina toimivia NES-pelejä Android-älypuhelimella ajetulla emulaattori-sovelluksella, mistä on esimerkkinä kuva kuusi. Konseptin ollessa moneen kertaan toimivaksi todistettu ja Unityn tarjotessa valmiin pelimoottorin totesin, että tekninen suunnittelu toimisi parhaiten prototyypin pohjalta. Samalla se antaisi elävän kuvan konseptin pelattavuudesta virtuaali-kontrolleilla.

5.1 Ensimmäinen prototyyppi



KUVA 7 Ensimmäinen prototyyppi PC ja Android -versioina

Royal Crow Squadron on niin sanottu 2.5D-peli, joka toimii kahdessa ulottuvuudessa silti käyttäen kolmiulotteisia elementtejä pelissä. Tämän pelin kohdalla se tarkoittaa, että pelaajan hahmo (kuvassa seitsemän pallo) ja viholliset (kuvassa seitsemän kuutiot) ovat 3D-objekteja taustan jäädessä kaksiulotteiseksi. Pelimekaanisesti tämä tarkoittaa, että kolmiulotteisessa ympäristössä kolmeulotteisten hahmojen liikkuminen rajoitetaan kahteen akseliin, jotka ovat tässä tapauksessa pysty- ja vaaka-akselit.

Tärkeänä elementtinä pelin pääkamera liikkuu koko ajan tietyllä nopeudella vaaka-akselilla positiiviseen suuntaan, mikä on Royal Crow Squadronin tapauksessa oikealle. Tästä huolimatta pelaajahahmon on pysyttävä ruudun suhteen paikallaan, vaikka pelaaja

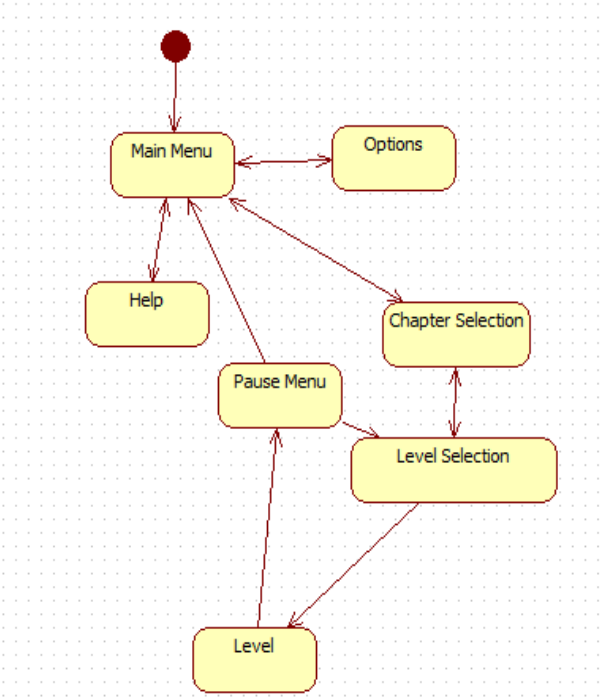
ei liikuttaisikaan sitä eteenpäin samalla nopeudella. Käytännössä tämä tarkoittaa, että pelaajahahmolla on yhtä nopea jatkuva liike eteenpäin kuin kameralla, mutta myös, että pelaajahahmon liike tulee olla rajoitettu kameran näyttämälle alueelle.

Sisällöllisesti ensimmäisessä prototyypissä on olemassa jo pelin keskeiset elementit: pelaajahahmo voi liikkua ja ampua ja viholliset voivat liikkua ja ampua. Lisäksi kummatkin ottavat toisen puolen ampumista projektiileista vahinkoa, mutta pelaajahahmo ei kuole näistä vielä.

5.2 Pelin runko

Peleissä yleisesti tarvitaan tapaa säilyttää informaatiota kentästä toiseen ihan pelin alkuvalikoista asti. Tällaista informaatiota on esimerkiksi pistemäärät, vaikeusasteet ja pelaajien tekemät asetukset. Unityssa tähän käytetään GameController-gameobjectia, joka asetetaan olemaan tuhoutumatta scenen vaihtuessa. Royal Crow Squadron tulee käyttämään GameControlleria tähän samaan tarkoitukseen.

5.3 Navigointi



KUVA 8 Tilakaavio pelistä Royal Crow Squadron

Peli vaatii muutakin kuin pelkästään pelattavan osion ollakseen eheä kokonaisuus. Käynnistettäessä ensimmäisenä tulee vastaan alkuvalikko, jossa on vaihtoehtoina mennä kentänvalintaan, asetuksiin tai peliohjeisiin. Kenttävalinnasta voi siirtyä mihin tahansa auki olevista kentistä, joista voi palata takaisin kenttävalittaan tai alkuvalikkoon joko pause-valikon kautta tai sitten pelaamalla kentän läpi. Navigointi näiden tilojen välillä tulee tapahtumaan valataosin Unityn omalla UI-järjestelmällä.

5.4 Pelimekaniikka

Pelimekaanisesti Royal Crow Squadron on hyvin yksinkertainen. Pelaaja ohjaa sivusta päin kuvattua lentokonetta vaaka- ja pystyakseleilla kokoajan eteenpäin menevällä pelikentällä. Vastaan tulee erilaisia vihollisia, jotka ovat joko lentoaluksia tai paikallaan pysyviä kiinteitä kohteita. Pelaaja voi ampua näitä vihollisia kohti, mikä osuessaan tekee viholliseen tietyn verran pysyvää vahinkoa. Ottaessaan tarpeeksi vahinkoa vihollinen tuhoutuu, jolloin pelaaja ei saa pisteitä. Vihollisten tuhoaminen ei kuitenkaan ole kenttien läpäisyn kannalta oleellista tiettyjä poikkeuksia lukuun ottamatta.

Viholliset liikkuvat ja ampuvat tietyn kaavan mukaisesti. Tämä tarkoittaa sitä, että ne liikkuvat kentällä ennalta määrätyn kuvion ja ampuvat tietyn väliajan mukaan. Joillakin vihollisilla on kuitenkin erikoiskäyttäytämisiä: ne voivat hakeutua pelaajan lentokonetta kohti tai vaikka jäädä ruutuun taistelemaan pelaajaa vastaan tuhoutumiseensa asti.

Osuessaan pelaajan koneeseen vihollisen panokset vähentävät pelaajan elämälukua. Sen laskiessa nollaan pelaajan lentokone tuhoutuu. Suunnitelman mukaan tämä kuitenkaan ei lopeta peliä, vaan pelaajan lentokone tulee kentälle uudelleen ja peli jatkuu uudestaan. Pelaajalle kertynyt pistemäärä kuitenkin vähenee jokaisesta kuolemasta.

5.5 Käyttöliittymä mobiililaitteella

Kuvassa kuusi on näkyvissä esimerkki pelin käyttöliittymästä mobiililaitteen kosketusnäytöllä. Voimme varmasti olla yhtä mieltä siitä, että NES:in D-pad ja muut napit vievät siitä kohtuuttomasti tilaa saaden ruudun näyttämään ahtaalta ja peittäen oleellisia osia pelinäkymästä. Tämä oli tilanne, jota en halunnut Royal Crow Squadroniin.

Niinpä päädyimme pelissä vain yhteen näkymään UI-elementtiin, joka on virtuaali-joystick. Sitä käytetään pelaajan aluksen ohjailuun. Ampuminen taas suoritetaan koskettamalla ruudun oikeaa puoliskoa. Pause-valikkoon siirtyminen hoidetaan painamalla Androidin Back-näppäintä, jonka voimme ainakin toistaiseksi olettaa olevan vakiona jokaisessa Android-laitteessa.

6 TOTEUTUS

Todettuamme ensimmäisen prototyypin olevan pelikelpoinen lähdin kehittämään peliä pidemmälle. Prototyyppi-lähtöinen kehitys kuitenkin oli jatkuvasti läsnä pelin etenemisessä: lisäsin peliin ominaisuuksia yksi kerrallaan ja testasin niiden toimivuutta käytännössä suoraan mobiililaitteella.

6.1 Keskeiset scriptit

Teknisen toteutuksen ensisijainen tehtävä oli pelin käyttämien scriptien kirjoittaminen. Kuten jo aiemmin mainitsin, päätin jo alun perin käyttää tähän Unityn tarjoamista vaihtoehdoista C#-ohjelmointikieltä muun muassa sen vahvan tyypityksen, .NET:in dokumentaation sekä oman kokemuspohjani perusteella. Tällä hetkellä pelin beta-versiossa on noin 30 eri scripti-tiedostoa. En kuitenkaan käy näistä kaikkia läpi vaan keskityn pelin toiminnan kannalta keskeisimpiin.

6.1.1 Game.cs

Game-scripti on GameController-gameobjectissa kiinni oleva scripti, joka pitää huolen pelin keskeisimmistä elementeistä. Tämän takia on olennaista, ettei se tai gameobject, johon se on kiinnitetty, ei tuhoudu pelin scenen vaihtuessa kuten normaalisti tapahtuu siinä tilanteessa. Game-scriptissä säilytetään myös olennaisia tietoja koko pelin toiminnon kannalta, mistä syystä muiden scriptien kirjoittamisen helpottamiseksi annoin sille hieman rikkonaisen Singleton-patternin jotta pelissä läsnä olevaan Game-scriptiin olisi olemassa jatkuva viittaus. (Microsoft Develepor Network: Singleton)

```
public static Game Singleton = null;
void Awake () {
    if (GameObject.FindGameObjectsWithTag("GameController").Length > 1) {
        Debug.Log("More than one gamecontrollers found");
        Destroy(gameObject);
    }
    else {
        DontDestroyOnLoad(gameObject);
        Singleton = this;
    }
```

KUVA 9 Singleton-patternin implementaatio Game-scriptissä

Kuvassa yhdeksän näemme Game-scriptin luonnin yhteydessä kutsuttavan Awake-metodin, joka aivan ensimmäisenä tarkistaa onko se kiinni ainoassa kentältä löytyvästä GameController-gameobjectissa. Tämä on ensiarvoista sen takia, että GameController on asetettu pelin aloitus-sceneen, ja mikäli pelaaja palaa uudestaan pelin ajon aikana nimenomaiseen sceneen, tuhoutumattomaksi asetettu GameController seuraa mukana, mutta aloitus-scenessä on jo valmiiksi uusi GameController. Tällöin uusi Game-scripti luontinsa yhteydessä huomaa, että GameControllereita on useampi ja tuhoaa itsensä ennen kuin tekee mitään muuta. Jos taas Game-scripti huomaa olevansa kiinni pelin ainoassa GameControllerissa, se asettaa itsensä tuhoutumattomaksi ja merkitsee itsensä Singleton-muuttujaan, minkä jälkeen sitä voi kutsua missä tahansa muussa Scriptissä pelkästään Game.Singleton-viittauksella.

Tärkeimpänä Game-scriptin huolehtimana asiana on pelin tietojen tallennus, mikä on myös keskeinen osa miltein jokaista isoa ja pientä peliä. Royal Crow Squadronissa tarvitsee kuitenkin tallentaa vain tietoja, jotka kertovat siitä mitkä pelin kentistä on läpäisty ja näissä kentissä saavutetut pisteet. Käytännössä tämä tehdään siten, että kentille annetaan enum LevelState-muuttuja, jossa on vaihtoehtoina *fresh*, joka tarkoittaa kentän olevan juuri auennut pelattavaksi; *tried*, joka tarkoittaa kentän olevan ainakin kerran kokeiltu; *made*, joka tarkoittaa kentän olevan läpäisty ja *locked*, joka tarkoittaa ettei kenttä ole vielä pelattavissa. Pelin tässä vaiheessa pisteytyksiä ei ole vielä implementoitu, mutta todennäköisesti jokaiselle kentälle tullaan vain antamaan long-tyyppinen muuttuja, johon tallennetaan sen kentän korkein pistemäärä.

```
private void LoadGameState () {

    if (PlayerPrefs.HasKey("SaveCheck")) {

        for (int i = 0; i < levelState.Length; i++) {

            switch (PlayerPrefs.GetInt("Level " + i + 1 + " state")) {
                case 0:
                levelState[i] = LevelState.fresh;
                break;
                case 1:
                levelState[i] = LevelState.tried;
                break;
                case 2:
                levelState[i] = LevelState.made;
                break;
                case 3:
                levelState[i] = LevelState.locked;
                break;
            }

        }

    }

    else {

        Debug.Log("Load Game State");

        for (int i = 0; i < levelState.Length; i++) {

            if (i == 0) {

                levelState[i] = LevelState.fresh;

            }

            else {

                levelState[i] = LevelState.locked;

            }

        }

    }
```

KUVA 10 Tallennuksen lataaminen Game-scriptissä

Kuvassa kymmenen näemme miten Royal Crow Squadronissa ladataan kenttien tilat. PlayerPrefs on Unityn ominaisuus, joka mahdollistaa pelin tietojen tallentamisen pelilaitteen muistiin avain-arvo-pareina. Sitä käyttäessä on otettava huomioon, että tallennettavat muuttujat voivat olla vain float, int ja string -tyyppisiä, mikä tekee sen käyttämisestä hieman hankalampaa monimutkaisemmilla tietorakenteilla. (Unity Script Reference: PlayerPrefs)

Kuvassa kymmenen pelin tietojen lataaminen aloitetaan tarkastamalla onko pelistä olemassa tallennusta tarkastamalla SaveCheck-avaimen olemassaolon PlayerPrefistä. SaveCheckillä ei ole mitään muuta käyttötarkoitusta joten sen arvolla ei ole mitään merkitystä. Mikäli SaveCheck löytyy, ladataan kenttien tilat, jotka ovat tallennettuna int-tyypin muuttujina jolloin ne täytyy muuttaa enum LevelState -muuttujuksi. Mikäli SaveCheck-avainta ei kuitenkaan löydy PlayerPrefistä eli pelistä ei ole tehty tallennusta,

scripti alustaa kenttien tilat oletusarvoilla, jotka antavat pelattavaksi aluksi vain ensimäisen kentän.

6.1.2 PlayerController.cs

Game-scriptin tavoin PlayerController-scripti on GameController-gameobjectin komponentti, koska sitä tarvitaan pelin jokaisessa scenessä alkuvalikkoa lukuun ottamatta. Yhtäläisyydet eivät kuitenkaan lopu tähän, sillä niin ikään PlayerControllerissa käytetään rikkinäistä Singleton-patternia, jolloin siihenkin voidaan tehdä viittauksia pelkästään käyttämällä PlayerController.Singletonia. Ollessaan Game-scriptin kanssa saman gameobjectin komponentti PlayerController ei myöskään tuhoudu scenen vaihtuessa, sillä jo Game-scriptissä kyseessä oleva gameobject on merkitty tuhoutumattomaksi scenen vaihtuessa.

Nimensä veroisesti PlayerController kontrolloi pelaajahahmon toimintoja. Tämän se saa aikaan tarkkailemalla pelaajan antamia syötteitä ja muuntamalla ne arvoiksi, jotka muut scriptit osaa tulkita pelihahmon liikkeiksi. PlayerController ei siis itse liikuta pelihahmoa vaan ottaa vain vastaan pelaajan syötteitä ja tulkkaa ne peliympäristön ymmärtämiksi arvoiksi.

```
#if UNITY_STANDALONE || UNITY_WEBPLAYER

        MovementX = Input.GetAxis("Horizontal");
        MovementY = Input.GetAxis("Vertical");

        if (Input.GetButton("Fire1")) {
            PlayerFire.Singleton.PrimaryFire();
        }

        if (Input.GetButton("Fire2")) {
            PlayerFire.Singleton.SecondaryFire();
        }

#endif
```

```
#if UNITY_ANDROID

        MovementX = joystick.position.x;
        MovementY = joystick.position.y;

        int i = 0;
        while (i < Input.touchCount) {

            if (Input.GetTouch(i).position.x >= Screen.width / 2)
            {
                PlayerFire.Singleton.PrimaryFire();
            }

            i++;

        }

#endif
```

KUVA 11 Pelaajan syötteiden vastaanotto PlayerController-scriptissä

Vaikka syötteiden tulkkaaminen ei varsinaisesti monimutkaista olekaan, se on tarpeellinen, koska pelin haluttiin olla pelattavissa myös tietokoneella jo pelkästään debuggaus-tarkoituksessa. PlayerController siis tarkastaa onko kyseessä pelin Android- vai standalone-käännös ja tekee sen mukaan päätöksen mitä syötteitä tarkkaillaan. Standalonella se tarkkailee pelimoottorin ohjausakseleina toimivien näppäimien

painalluksia, jotka useimmiten ovat näppäimistössä tai hiiressä. Peliä ajettaessa Androidilla kuitenkin tarkkaillaan virtuaali-joystickissä kiinni olevaa Joystick.js-scriptiä, johon on PlayerControllerissa tallennettu viittaus joystick-muuttujaan. Androidilla tarkkaillaan myös kosketusta näytön oikeaan puoliskoon, mikä laukaisee ampumisen.

Kumpikin kuvan 11 koodi pyörii Update-metodissa, mutta vain silloin, kun Game-scriptin GameOn-muuttuja on true, millä estetään käyttäjän peliin liittyvien syötteiden tarkkailu esimerkiksi pelin ollessa keskeytetty tai alkuvalikossa. #if ja #endif -merkinnät tarkkailevat pelin alustaa joten alustakohtaiset koodit ovat poissa päältä, kun peli ei ole käännetty kyseiselle alustalle.

6.1.3 Pelihahmossa kiinni olevat scriptit

PlayerControllerin ottaessa vastaan pelaajan syötteitä pelihahmon toimintaa ohjailee pari eri scriptiä, jotka ovat kaikki kiinni scenessä pelaajahahmossa eli pelaajan ohjaamassa lentokoneessa. PlayerMover.cs huolehtii pelihahmon liikuttamisesta, PlayerFire.cs ampumisesta ja PlayerHealth.cs pelihahmon elämäpisteiden kulumisesta ja kuolemisesta. Kaikissa näissäkin sovelletaan hieman rikkinäistä singleton-patternia.

PlayerMover.cs tarkkailee Update-metodissaan PlayerController-scriptin MovementX ja MovementY -muuttujia, joiden mukaan se liikuttaa pelaajahahmoa y ja x -akseleilla. Lisäksi PlayerMoverissa tehdään pieniä säädöksiä pelaajahahmon liikkumiskäyttäytymiseen kuten liu’unhallintaa sekä liikutetaan koko ajan pelaajahahmoa eteenpäin samalla nopeudella kuin kameraa.

PlayerFire.cs saa kutsuja PlayerController-scriptistä pelaajan antaessa ampumiskomentoja kontrollien kautta. PlayerFire pitää varsinaisesti vain listaa pelaajahahmolla olevista aseista ja kutsuu tässä listassa olevia aseita aina itse saadessaan metodikutsun. Se myös huolehtii aseiden asettamisesta tähän listaan sekä pelihahmon näkyvään avatariin sekä kentän alussa että pelaajan poimiessa mahdollisen uuden aseen.

PlayerHealth.cs tarkkailee osumia pelaajahahmon collideriin. Osuman ollessa vihollisen ampuva projektiili pelihahmon elämäpisteistä vähennetään projektiilissa tietona ollut

vahinko. Mikäli pelihahmon elämäpisteet vähenevät näin nollaan, PlayerHealth ilmoittaa pelaajahahmon kuolemasta kentän Level-scriptille, joka tekee tilanteessa vaadittavat toimenpiteet.

6.1.4 Level.cs

Level-scripti on jokaisessa pelikentässä oleva scripti, joka hoitaa suuren määrän kentän kannalta oleellisia asioita. Näihin kuuluu muun muassa pausen painaminen, pelaajahahmon kuoleminen ja uudelleen pelikentälle tekeminen, erikoistilanteet kentissä ja kentän läpäiseminen tai keskeyttäminen. Level-scripti hallinnoi myös muutamia muita kenttään liittyviä scriptejä, jotka liittyvät kenttien introon ja läpäisemiseen.

6.1.5 Vihollisissa kiinni olevat scriptit

Ainoa scenen alussa vihollisissa kiinni oleva aktiivinen scripti on Enemy.cs. Itsessään se pelkästään huolehtii vihollisaluksen vahingonottamisesta ja tuhoutumisesta hyvin samalla tavalla kuin PlayerHealth tekee pelaajahahmolle. Enemy kuitenkin myös tarkkailee milloin vihollinen on näkyvissä ruudulla. Tämän tapahtuessa – tai oikeastaan jo hetki ennen – Enemy aktivoi kaksi vihollisten käyttäytymistä hallitsevaa scriptiä, EnemyFiringPatternin ja EnemyMovingPatternin. Niistä kerrotaan yksityiskohtaisemmin seuraavassa, tekoälyä käsittelevässä, luvussa.

6.2 Tekoäly

Luvun otsikosta huolimatta Royal Crow Squadronin kohdalla ei voida varsinaisesti puhua tekoälystä, sillä esikuviensa tavoin vihollisten käyttäytyminen perustuu kaavoihin ja vaiheisiin. Aktivoimisestaan lähtien jokainen vihollinen toteuttaa tiettyä kaavaa, joka tarkoittaa esimerkiksi, että se liikkuu tietyn verran eteenpäin, sitten liikkuu ylös ja sitten alas tiettyjen aika-intervallien mukaisesti.

Kahdesta perusvihollisten käyttäytymistä hallitsevista scripteistä EnemyMovingPattern käsittelee nimensä mukaisesti liikkumista. Liikkumiskaavasta riippumatta jokainen vihollinen käyttää tätä samaa scriptiä. Kaava itsessään valitaan enum MovingPattern -tyyppinen muuttuja, jossa vaihtoehtoina on tällä hetkellä kuusi eri kaavaa. Sen lisäksi käytössä on intervalli-muuttuja, joka käytetään eri liikkumisvaiheiden rytmittämiseen.

Intervalli-muuttujaa muuttamalla saadaan aikaan vaihtelua myös saman kaavan sisällä. Kummatkin näistä muuttujista on merkitty public-suojauksella, jotta niitä olisi mahdollista muuttaa suoraan editorissa kajoamatta itse koodiin.

```
case MovingPattern.CenterHoming:
    transform.Translate(Vector3.left * Time.deltaTime * horizontalSpeed);
    if (intervalTimer >= intervalInvokingTime) {

        if (movingVector == Vector3.zero && Mathf.Abs(transform.position.y) >= 0.03) {

            if (transform.position.y > 0)
                movingVector = Vector3.down;

            else
                movingVector = Vector3.up;

        }

        if (Mathf.Abs(transform.position.y) >= 0.03) {

            transform.Translate(movingVector * Time.deltaTime * verticalSpeed);

        }

        else if (movingVector != Vector3.zero) {

            movingVector = Vector3.zero;

        }


    }
    break;
```

KUVA 12 Esimerkki EnemyMovingPattern-scriptissä olevasta kaavan toiminnasta

Kuvassa 12 näemme esimerkin kaavasta, joka on CenterHoming eli keskustaa päin hakeutuva. Käytännössä se tarkoittaa, että viholliskone menee ensin horistontaalisesti eteenpäin intervallin verran aikaa sekunneissa, minkä jälkeen se lähtee liikkumaan vertikaalisesti kohti ruudun vertikaalista keskustaa. Saavutettuaan vertikaalisen keskustan kone jatkaa horisontaalisesti eteenpäin.

EnemyShootingPattern tekee periaatteessa samaa vihollisen ampumiselle kuin EnemyMovingPattern liikkumiselle. Suunnitteluvaiheessa tosin huomasimme, ettei kovin monelle eri ampumiskaavalle ole käyttöä joten niitä on tällä hetkellä käytössä vain yksi, joka on BurstShot eli purskeammunta. Käytännössä tämä tarkoittaa, että vihollinen odottaa intervallin, sitten ampuu tietyn määrän projektiileja oman tulinopeutensa mukaisesti, odottaa taas intervallin, ampuu tietyn määrän ja niin edelleen. Scripti on kuitenkin rakennettu helposti laajennettavaksi silmällä pitäen tarvetta uusille ja erilaisille ampumakaavoille.

6.3 Mobiili-ominaisuudet

Tärkeimpänä mobiililaitteita koskevana ominaisuutena Royal Crow Squadronissa on virtuaali-joystick, jolla ohjataan pelaajahahmon liikkumista. Pelissä käytimme suoraan Unityn standard assettina tarjoamaa joystickiä, jonka scripti on kirjoitettu UnityScriptillä. Se keskustelee kuitenkin C#:lla kirjoitetun PlayerControllerin kanssa ongelmitta, koska tarvitsemme joystickistä vain kahta float-arvoa, jotka määrittelevät joystickin asentoa horisontaalisesti ja vertikaalisesti välillä -1.0…1.0.

Itse joystick toimii hieman tahmeasti, mikä saattaa johtua yleisesti kosketusnäytöstä, sillä virtuaali-ohjaimeen on melkein mahdoton saada analogisen ohjaimen tuntumaa. Myös itse joystickissä saattaa olla vikaa, mutta muut Unityn Asset Storesta saatavat vaihtoehdot maksavat rahaa ja itse tehtynä on mahdoton arvioida tuleeko lopputuloksesta vaivan ja ajankäytön arvoinen.

Lähes kaikissa muissa mobiilikäyttöliittymän osissa pystyin luottamaan Unity hiiri-simulaatio -ominaisuuteen, joka sai pelin UI:n toimimaan saumattomasti samalla koodilla sekä tietokoneella että mobiililaitteilla.

Koska pelissä kentän rajat määräytyvät sen mukaan, mihin näytön rajat yltävät, Android-puhelimien vaihtelevat näyttökoot aiheuttavat jonkinlaista päänvaivaa kehittäjälle. Suhtauduin tähän alusta asti pakottamalla pelin resoluutioksi 800x480, mistä ei ainakaan toistaiseksi ole aiheutunut laajamittaisia ongelmia. Seuraavassa luvussa kuitenkin käy selville testauksessa käytettyjen laitteiden tietty samankaltaisuus, minkä johdosta meillä ei ole kovinkaan laajaa kuvaa pelinäkymästä erilaisilla laitteilla.

6.4 Testaaminen

Royal Crow Squadronin ominaisuuksien testaamisessa käytettiin vaihtelevaa toimintatapaa riippuen ominaisuuden luonteesta. Mikäli ominaisuus ei koskenut mobiilikäyttöliittymää vaan pelin yleistä toimivuutta, se testattiin suoraan editorissa tietokoneella. Mobiilikäyttöä koskevat ominaisuudet testattiin sen sijaan mobiililaitteella. Ominaisuuden luonteesta huolimatta pelistä tehtiin tasaisin väliajoin versio mobiililaitteelle jotta olisimme voineet arvioida pelaamisen luonnetta, ja suuntaa, johon olimme peliä viemässä. Näitä versioita kutsuttiin alfa ja beta -buildeiksi.

Testilaitteiden valinnassa otettiin huomioon ensisijaisesti laitteiden suosio. Samsungin ollessa markkinajohtaja 65% osuudella Android-mobiililaitteiden markkinoista testilaitteiden sarjassa päätettiin keskittyä Samsungin Galaxy -sarjaan (Hoch, 2014). Mukaan valittiin laitteita niiden näyttökoon ja iän perusteella. Koettiin kuitenkin tärkeäksi valita laite myös tietyn valmistajan ulkopuolelta, jotta olisi mahdollista havaita viat, jotka ilmenevät vain tietyllä valmistajalla.

TAULUKKO 1 Eri testilaitteiden näyttöjen vertailu

Nimi:	Koko (tuumia):	Resoluutio (pikseleitä):	Kuvasuhde:	Valmistusvuosi:
Sony Xperia Z	5,0	1920x1080	16:9	2013
Samsung Galaxy Note II	5,5	1280x720	16:9	2012
Samsung Galaxy S4	5,0	1920x1080	16:9	2013
Samsung Galaxy S2	4,3	800x480	15:9	2011
Samsung Galaxy S	4,0	800x480	15:9	2010
Samsung Galaxy Tab 10.1	10,1	1280x800	16:10	2011

Taulukosta yksi käy ilmi testilaitteiden näytöt ja valmistusvuodet. Ensisijainen päämäärä laitteiden testaamisessa oli testata toimiiko peli hyväksyttävästi kyseisellä laitteella. Tämän jälkeen huolta aiheutti peliin koodillisesti pakotetun resoluution (800x480) toimivuus erikokoisilla näytöillä. Testilaitteista vain kahden toimiessa natiivina vastaavalla kuvasuhteella. Kuitenkaan pelin toimivuudessa ei havaittu ongelmia kahdella muullakaan kuvasuhteella. Vaikka onkin mahdollista olettaa, että valtaosa Android-laitteista toimisi näillä kuvasuhteilla tai ainakin joillakin vastaavista, on kattava lista kaikista kuvasuhteista vaikea löytää. Testilaitteet kuitenkin valittiin hyvin pitkälle niiden suosion perusteella, mistä johtuen voidaan olettaa pelin resoluution toimivan merkittävässä osassa Android-laitteista.

6.5 Ongelmakohdat

Varsinaisessa teknisessä suunnittelussa ja toteutuksessa ongelmat olivat hyvin harvinaisia. Ainoa asia, jota en alustavasti osannut tehdä oli koodillisesti näytön rajojen etsiminen, jotta pelaajahahmon liikkeet voitaisiin rajoittaa näkyvän alueen sisälle. Tähänkin asiaan tuli ratkaisu hyvin nopeasti kahlaamalla läpi Unityn dokumentaatiota.

Todelliset aikaa vievät asiat liittyivät matematiikkaan. Itselläni on jonkinlaista taustaa teknisissä opinnoissa, mutta niistä on jo hetken verran aikaa joten radiaanit ja geometria ei ollut kauhean tuoreessa muistissa. Esimerkkinä tästä voidaan pitää scriptiä, joka ohjasi tykkitornin liikettä. Tykkitornilla oli eri vaiheita, joiden aikana se kääntyi tietyn ajan puitteissa tiettyihin kulmiin.

Silti ei kannata olettaa, että projektin valmistumisessa ei ollut ongelmia. Ne koskivat minua, mutta ei roolissani teknisenä toteuttajana eli ohjelmoijana. Projektin roolien rajojen hämärtyminen onkin asia, jota pohdin tämän opinnäytetyön viimeisessä luvussa.

7 VALMIS SOVELLUS

KUVA 13 Sony Xperia Z -älypuhelimella otettuja kuvankaappauksia Royal Crow Squadronin beta-vaiheesta.

Kuvassa 13 näkyy, että Royal Crow Squadron on tällä hetkellä luvatusti beta-vaiheessa, mutta silti kiusallisen kaukana julkaisuvalmiista. Teoriassa pelistä puuttuu muutamien valikoiden lopullisia versioita, kenttäeditointia, muutamia loppuvastuksia, 2D-grafiikkaa sekä animaatioita. Käytännössä näiden lisäksi puuttuu myös useita kymmeniä tunteja käytettäväksi testaamiseen ja tasapainottamiseen.

Pelin valmiiksi saattaminen on tällä hetkellä kiinni muutamista tekijöistä, joista tärkein on 2D-artistin löytyminen. Alkuperäinen artisti aloitti vuodenvaihteessa koulun, mistä johtuen hänen kiinnostuksensa tätä projektia kohtaan laantui hieman. Toisella projektin päävetäjällä, Heikki Mäkelällä, on myös kova kiire palkkatöihin, joihin itsekin menisin tässä vaiheessa uraani kovin mielelläni. Tästäkään huolimatta ei ole täysin mahdotonta, että peli julkaistaisi vielä muutaman kuukauden sisällä, sillä siitä ei oikeasti puutu paljoa.

Edellä mainittujen puutteiden jälkeenkin Royal Crow Squadron ei kuitenkaan ole valmis, sillä alkuperäinen ajatus oli tehdä pelistä kolmen chapterin mittainen. Näistä ensimmäiset, nyt valmistuvat viisi kenttää ovat vain ensimmäinen chapteri. Loppujen kahden chapterin kohtalo on todellisen hämärän peitossa. Toisaalta on syytä muistaa, että Royal Crow Squadronin pohja on vahvasti olemassa, eikä uusien kenttien

tekeminen ole kovinkaan suuri haaste enää tässä vaiheessa. Kahden viimeisen chapterin kohtalo tulee siis ratkaistavaksi siinä vaiheessa, kun ensimmäinen chapteri on valmis.

Mitä tulee pelin julkaisuun, ja siitä mahdollisen rahan saantiin, alun perin tehtiin päätös, ettemme pyri tähän projektiin sisäistämään mitään muuta tienaamismenetelmää kuin mainokset. Pelin käyttöliittymää suunniteltaessa on koko ajan pidetty mielessä, että jossain vaiheessa pelinäkymän ylälaidassa kummittelee jatkuvasti mainosbanneri. Monet mobiilimainontapalveluiden tarjoajat ovat ottaneet Unityn niin avosylin vastaan, että ovat tehneet pelimoottorille natiivin pluginin oman palvelunsa käyttöön, mikä tekee mainosten toteuttamisen hyvin marginaaliseksi tehtäväksi Unity-kehittäjälle. Joka tapauksessa myös mainokset saavat odottaa julkaisuversion valmistumista.

8 POHDINNAT

Opinnäytetyön tarkoituksena oli teknisesti suunnitella ja toteuttaa mobiilipeli Unity-pelimoottorilla. Tällä hetkellä Royal Crow Squadron on olemassa ja pelattavissa vaikkakaan ei ihan niin valmiina kuin itse henkilökohtaisesti toivoin. Se on jopa hyvin pitkälle visiomme mukainen eli peli, jota suunniteltiin, on myös valmistunut.

Tavoitteena sen sijaan oli kokemuksen ja tiedon hankkiminen pelin teknisestä suunnittelusta ja toteutuksesta Unity-pelimoottorilla. Tässäkin tapauksessa voidaan sanoa, että tavoitteeseen on päästy. Vaikka mainitsinkin, etten kohdannut varsinaisesti ongelmia oman tonttini hoitamisessa, jokaisen uuden pelikokonaisuuden luominen on kokemusta ja rutiinia rakentava tehtävä. Loppujen lopuksi ongelmien kohtaaminen ja ratkaiseminen on elävä osa jokaista ohjelmistoprojektia, oli kyseessä sitten laaja peliprojekti tai pieni internet-sovellus. Siihenkin voi rakentaa rutiinia, ja pitkällä kokemuspohjalla ongelmanratkaisusta on eittämättä positiivisia vaikutuksia ongelmien kohtaamiseen tulevaisuudessa.

Kuten jo mainitsin, tällä hetkellä Royal Crow Squadron ei ole vielä julkaisukunnossa. Asioiden loppuun vieminen on tärkeä taito, jonka omaksuminen vaatii ihmiseltä vahvaa päättäväisyyttä ja integriteettiä. Näitä asioita mitataan mahdollisesti vielä tulevina kuukausina.

On kuitenkin syytä huomata, että pelin valmistumisvaihe ei ole läheskään vaan päättäväisyyden puutetta. Haukkasimme peliä suunnitellessa liian ison palan, mikä selvisi meille vasta projektin ollessa käynnissä. Joihinkin peliprojekteihin osallistuneena pitäisi jo tietää, että hyvin harva asia on niin yksinkertainen kuin alun perin sen ajattelee olevan. Tämä lienee totuus ylipäätään kaikessa ohjelmistotuotannossa, mutta hyvin paljon etenkin pelituotannossa. Voin siis tässä vaiheessa tunnustaa aivan suoraan, että käytettävissämme oleviin miestyötunteihin nähden Royal Crow Squadron oli aivan liian iso projekti.

Valtaosa projektia tehdessä vastaan tulleet ongelmat liittyivät tavalla tai toisella tähän. Alkuperäinen idea oli, että minä hoidan ohjelmoinnin ja Heikki Mäkelä 3D-mallintamisen, ja kummatkin osallistuvat pelisuunnitteluun. Käytännössä projektin

missään vaiheessa asiat eivät olleet näin yksinkertaisia. Kummankin roolien rajat hämärtyivät vahvasti Heikin joutuessa osallistumaan lähes kaikkeen grafiikan tekemiseen 2D-grafiikan piirtämistä myöden, jolloin minulle jäi melkein kaikki muu testaamisesta pelimekaanikan suunnitteluun. Loppujen lopuksi olisimme varmaankin tarvinneet kaksi täyspäiväistä jäsentä lisää projektin toteutukseen: pelisuunnittelijan ja 2D-artistin. Toinen vaihtoehto olisi voinut olla paljon kevyempi projekti heti alusta asti.

Johdannossa vihjasin myös Royal Crow Squadronin käytöstä referenssinä tekijöidensä taidoista pelien tekemisessä ja mahdollisesti alalle töihin pääsystä. Onko siitä apua näillä päämäärillä? Uskon siihen vahvasti varsinkin sitten, kun olemme saaneet pelin julkaistua.

LÄHTEET

Sandstrom, G. 2010. The Source. Angry Birds Smartphone App Takes Off For Rovio. Luettu 19. 3. 2014. http://blogs.wsj.com/source/2010/05/12/angry-birds-smartphone-app-takes-off-for-rovio/

Long, N. 2014 Edge. Two billion downloads? We're just getting started, says Angry Birds creator Rovio. Luettu 19. 3. 2014. http://www.edge-online.com/features/two-billion-downloads-were-just-getting-started-says-angry-birds-creator-rovio/

Bond, N, 2012. Daily Mail. Now they're taking off: First official Angry Birds theme park opens as game's creators look to become as big as Disney. Luettu 19. 3. 2014. http://www.dailymail.co.uk/sciencetech/article-2148493/First-official-Angry-Birds-theme-park-opens-games-creators-look-big-Disney.html

Neogames. Luettu 19. 3. 2014. http://www.neogames.fi/

Nystrom, R. Game Loop. Luettu 5. 5. 2014. http://gameprogrammingpatterns.com/game-loop.html

McShaffry, M., Graham, D. 2013. Game Coding Complete - 4th Edition. Boston: Course Technology

Nokia, 2009. Story of Nokia. Luettu 15. 4. 2014. http://wayback.archive.org/web/20090209232201/http://www.nokia.com/A4303014

Fahey, M. 2013. The Best-Looking Mobile Game Rises Again, More Powerful Than Ever. Luettu 15. 4. 2014. http://kotaku.com/the-best-looking-mobile-game-rises-again-more-powerful-1339631128

Thinkgaming, 2014. Clash of Clans. Luettu 15. 4. 2014. http://thinkgaming.com/app-sales-data/1/clash-of-clans/

Sinha, R. 2014. Clash of Clans Revenue at $654,000 Per Day, Third Best Peforming Freemium Title Worldwide. Luettu 15. 4. 2014. http://gamingbolt.com/clash-of-clans-revenue-at-654000-per-day-third-best-performing-freemium-title-worldwide

Valadares, J. 2011. Free-to-play Revenue Overtakes Premium Revenue in the App Store. Luettu 15. 4. 2014. http://blog.flurry.com/bid/65656/Free-to-play-Revenue-Overtakes-Premium-Revenue-in-the-App-Store

Gartner, 2013. Gartner Says Worldwide Video Game Market to Total 93$ Billion in 2013. Luettu 15. 4. 2014. http://www.gartner.com/newsroom/id/2614915

Brightman, J. 2014. Mobile gaming to push industry above $100 billion by 2017. Luettu 15. 4. 2014. http://www.gamesindustry.biz/articles/2014-01-14-mobile-gaming-to-push-industry-above-USD100-billion-by-2017

Eduskunta, 2013. Eduskunta hyväksyi vuoden 2014 budjetin. Luettu 15. 4. 2014. http://web.eduskunta.fi/Resource.phx/pubman/templates/1.htx?id=6280

Cassavoy, L. 2008. About.com. Meet the T-Mobile G1: The First Android Phone. Luettu 19. 3. 2014. http://cellphones.about.com/od/smartphonebasics/p/meet_tmobile_g1.htm

Gartner, 2013. Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Sales in Third Quarter of 2013. Luettu 19. 3. 2014. http://www.gartner.com/newsroom/id/2623415

Netmarketshare. Mobile/Tablet Operating System Share. Luettu 19. 3. 2014. http://www.netmarketshare.com/

Android Api Guide. What Is API Level? Luettu 19. 3. 2014. http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels

Opensingal, 2013. Android Fragmentation Visualized. Luettu 19. 3. 2014. http://opensignal.com/reports/fragmentation-2013/

Appbrain, 2014. Number of Android applications. Luettu 19. 3. 2014. http://www.appbrain.com/stats/number-of-android-apps

Google Support. Kehittäjäksi rekisteröityminen. Luettu 19. 3. 2014. https://support.google.com/googleplay/android-developer/answer/113468?hl=fi

Blakespot, 2013. Blake's iOS Device Specifications Grid. Luettu 19. 3. 2014. http://blakespot.com/ios_device_specifications_grid.html

Apple, 2014. iOS 7.0.5. Luettu 19. 3. 2014. http://support.apple.com/kb/DL1718/

148Apps, 2014. Count of Active Applications in the App Store. Luettu 19. 3. 2014. http://148apps.biz/app-store-metrics/?mpage=appcount

Apple Developer. Choosing an iOS Developer Program. Luettu 19. 3. 2014. https://developer.apple.com/programs/start/ios/

Hachman, M. 2013. PCWorld. Windows Phone Store passes 200,00 apps, so let's make them better, Microsoft!. Luettu 19. 3. 2014. http://www.pcworld.com/article/2080804/windows-phone-store-passes-200-000-apps-so-lets-make-them-better-microsoft.html

Alfreds, M. 2014. News24. Developing countries to boost Android growth. Luettu 29. 4. 2014. http://www.news24.com/Technology/News/Developing-countries-to-boost-Android-growth-20140108

Github, 2014. Gameplay – 3D Game Framework. Luettu 15. 4. 2014. http://blackberry.github.io/GamePlay/api/index.html

badLogickGames. Luettu 15. 4. 2014. http://libgdx.badlogicgames.com/

Unity3d, 27. 1. 2014. "More than 2000 games #madewithunity in 48 hours at this year's Global Game Jam!! Check out the whole list: http://bit.ly/unityatGGJ14 #ggj14" Luettu 19. 3. 2014. https://twitter.com/

Brodkin, J. 2013. How Unity3D Became a Game-Development Beast. Luettu 19. 3. 2014. http://slashdot.org/topic/cloud/how-unity3d-become-a-game-development-beast/

Unity3D. Luettu 19. 3. 2014. http://unity3d.com/

Goldstone, W. 2013. Unity Blogs. Unity Native 2D Tools. Luettu 19. 3. 2014. http://blogs.unity3d.com/2013/08/28/unity-native-2d-tools/

Unity Manual: DirectX 11, 2013. Using Direct X 11 in Unity 4. Luettu 19. 3. 2014. http://docs.unity3d.com/Documentation/Manual/DirectX11.html

Bayer, Stöwer, Sauter, 2011. PC-Games Hardware. Batman Arkham City: Technical advantages of the PC version - DX11 tessellation and more. Luettu 19. 3. 2014. http://www.pcgameshardware.com/aid,846939/Batman-Arkham-City-Technical-advantages-of-the-PC-version-DX11-tessellation-and-more/News/

Kumparak, G. 2011. Techrunch. Creator Of Angry Birds' Physics Engine Calls Out Rovio For Not Giving Him Credit. Luettu 19. 3. 2014. http://techcrunch.com/2011/02/28/creator-of-angry-birds-physics-engine-calls-out-rovio-for-not-giving-him-credit/

Yahoo! Finance, 2011. Unite 11: Unity Technologies Opens Canadian Office. Luettu 19. 3. 2014. http://finance.yahoo.com/news/Unite-11-Unity-Technologies-iw-2695539341.html?x=0

Unity Manual: Creating Scenes, 2013. Creating Scenes. Luettu 19. 3. 2014. http://docs.unity3d.com/Documentation/Manual/CreatingScenes.html

Unity Script Reference: Component. Luettu 19. 3. 2014. http://docs.unity3d.com/Documentation/ScriptReference/Component.html

Unity Script Reference: GameObject. Luettu 19. 3. 2014. http://docs.unity3d.com/Documentation/ScriptReference/GameObject.html

Unity Documentation: Transform. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/Components/class-Transform.html

Unity Script Reference: Vector3. Luettu 19. 3. 2014. http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html

Unity Documentation: Components. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/Components/Components.html

Mono. Luettu 19. 3. 2014. http://www.mono-project.com/Main_Page

Loewald, T. 2013. Head First into Unity with UnityScript. Luettu 19. 3. 2014. http://wiki.unity3d.com/index.php?title=Head_First_into_Unity_with_UnityScript

Microsoft Developer Network. Visual C#. Luettu 19. 3. 2014. http://msdn.microsoft.com/en-us/library/kx37x362.aspx

Unity Answers. How is the realtionship between Rodrigo Barreto de Oliveira and Unity? Luettu 19. 3. 2014. http://answers.unity3d.com/questions/39271/how-is-the-relationship-between-rodrigo-barreto-de.html

Unity Manual: Scripts. Creating and Using Scripts. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/Manual/CreatingAndUsingScripts.html
Unity Script Reference: MonoBehaviour. Luettu 29. 4. 2014. https://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html

Unity Script Reference: MonoBehaviour.Start. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Start.html

Unity Script Reference: MonoBehaviour.Update. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Update.html

Unity Script Reference: MonoBehaviour.FixedUpdate. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.FixedUpdate.html

Unity Script Reference: MonoBehaviour.OnGUI. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.OnGUI.html

Unity Script Reference: MonoBehaviour.OnDestroy. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.OnDestroy.html

Unity Manual: Input. Luettu 19. 3. 2014. http://docs.unity3d.com/Documentation/Manual/Input.html

Unity Manual: Android. Getting Started with Android Development. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/Manual/android-GettingStarted.html

Unity Manual: Mobile Advanced. Advanced Unity Mobile Scripting. Luettu 19. 3. 2014. https://docs.unity3d.com/Documentation/Manual/MobileAdvanced.html

Microsoft Developer Network. Implementing Singleton in C#. Luettu 19. 3. 2014. http://msdn.microsoft.com/en-us/library/ff650316.aspx

Unity Script Reference: PlayerPrefs. Luettu 19. 3. 2013. https://docs.unity3d.com/Documentation/ScriptReference/PlayerPrefs.html

Hoch, D., 2014. Samsung Remains King of the Android Market with 65% Share of All Android Devices. Luettu 5. 5. 2014. http://www.localytics.com/blog/2014/samsung-remains-king-of-the-android-market/