jamk.fi

IOS-PELIN LUOMINEN UNITY-PELIMOOTTORILLA

Tuomo Stamblewski

Opinnäytetyö
Huhtikuu 2014

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t) Stamblewski, Tuomo	Julkaisun laji Opinnäytetyö	Päivämäärä 27.04.2014
	Sivumäärä 57	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
IOS-PELIN LUOMINEN UNITY-PELIMOOTTORILLA

Koulutusohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
Mieskolainen, Matti

Toimeksiantaja(t)
Tuotantoyhtiö Taso Oy

Tiivistelmä
Opinnäytetyön tarkoituksena oli toteuttaa mobiilipeliprototyyppi Tuotantoyhtiö Taso Oy:lle käyttäen Unity-pelimoottoria. Pelin suunnittelu ja ideointi tapahtui yhdessä yrityksen kanssa. Tarkoituksena oli tehdä prototyyppi yrityksen suunnittelemasta mobiilipelistä sekä tutkia pelien tekemistä iOS-alustalle käyttäen Unity-pelimoottoria. Prototyypin peli-idea oli reaaliaikainen 3D-verkkopeli, jossa pelaajat taistelevat tankeilla toisiaan vastaan yksi kerrallaan.

Opinnäytetyössä käsitellään mobiilipelien ja -laitteiden historiaa, iOS-laitteita, Unity-pelimoottoria, yleisiä mobiilipeliohjelmoinnin haasteita sekä käydään läpi prototyypin toteuttamisen eri vaiheet. Opinnäytetyössä tulee ilmi Unity–pelimoottorin rakenne, toimintalogiikka ja sen käyttämä komponenttimalli. Opinnäytetyö sisältää paljon sovellettavia ohjelmointiesimerkkejä ja optimointiratkaisuja.

Työn teknisen toteutuksen osiossa käydään läpi pelin valmistusprosessi aina suunnittelusta toteuttamiseen asti. Lisäksi perehdytään yksityiskohtaisesti Unity pelimoottorin ominaisuuksiin ja miten eri ominaisuuksia voidaan käyttää eri tilanteissa. Tämän lisäksi kerrotaan mitä kannattaa ottaa huomioon mobiilipeliä luodessa, kuten mobiililaitteiden rajattu teho ja –muistimäärä tai erilaiset käskytystavat näppäinten puutteen vuoksi.

Avainsanat (asiasanat)
iOS, Unity, Optimointi, Mobiilipeli, Pelisuunnittelu, Pelimoottori, C#

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
Stamblewski, Tuomo	Bachelor´s / Master's Thesis	27.04.2014
	Pages 57	Language Finnish
		Permission for web publication (X)

Title
IOS-GAME DEVELOPMENT WITH UNITY GAME ENGINE

Degree Programme
Software Engineer

Tutor(s)
Mieskolainen, Matti

Assigned by
Tuotantoyhtiö Taso Oy

Abstract
The purpose of the thesis was to create a mobile game prototype for Tuotantoyhtiö Taso Oy using the Unity game engine. Game design and ideation were carried out with the company. The aim was to make a prototype of a mobile game according to the plans the company had made and also explore ways to produce games with Unity game engine for iOS platform. The game idea of the prototype was a real time 3D multiplayer game, where players fight each other with tanks.

This thesis deals with the history of mobile games and devices, iOS devices and Unity game engine as well as general mobile game programming challenges and different stages of implementation of prototype are discussed in the text. The thesis explains the structure, logic and component model of Unity game engine. The thesis also includes applicable programming examples and optimization solutions.

In this thesis game production process is discussed from design to technical implementation. Details in game engine features are explained in more detail and how they can be used in different situations is discovered. In addition, the thesis looks at matters that have to be taken into account when designing and implementing a mobile game.

Keywords
iOS, Unity, Optimization, Game Design, Game engine, C#

Miscellaneous

SISÄLTÖ

Kuvioluettelo

Taulukkoluettelo

KÄSITTEET

3D-MALLI

3D-malli on tietokonegrafiikalla esitetty kolmiulotteinen kappale. Yksinkertaisimmillaan 3D-malli esitellään rautalankamallina.

3G

3G-termillä viitataan niin sanottuun kolmannen sukupolven mobiiliverkkoon. 3G mahdollistaa suuret tiedonsiirtonopeudet myös mobiililaitteilla.

ASSET

Assetteja eli käyttömateriaaleja ovat Unity-pelimoottorissa kaikki äänet, tekstuurit, materiaalit, 3D-mallit sekä myös skriptit.

C#

C# on Microsoft-yhtiön kehittämä ohjelmointikieli .NET-ympäristöön. C# on vahvasti tyypitetty kieli. C# on yksi kolmesta ohjelmointikielestä, jolla voidaan ohjelmoida Unity-pelimoottorissa.

FPS

Frames Per Second eli kuvataajuus. Kuvataajuudella tarkoitetaan sekunnissa ruudulle piirrettävien kuvien määrää.

PELIMOOTTORI

Pelimoottorit ovat videopelien runko, joiden päälle pelikehittäjät rakentavat pelinsä. Ne yleensä sisältävät renderointimoottorin, fysiikkamoottorin sekä muita pelien tekemiseen liittyviä työkaluja.

POLYGON

Polygon eli monitahokas koostuu useasta reunaviivasta muodostaen kaksiulotteisen pinnan 3D-avaruuteen. Polygoneja toisiinsa yhdistämällä voidaan luoda 3D-malleja.

PPI

Pixels Per Inch tarkoittaa pikseleiden määrää tuumaa kohti.

PROFILER

Profiler on Unity Pro -pelimoottorin mukana tuleva profilointityökalu, jolla saadaan tarkkaa tietoa pelin suorituskyvystä eri osa-alueilla kuten prosessorilaskennan tai fysiikoiden suorittamisesta.

PROTOTYYPPI

Prototyyppejä ovat ensimmäiset testiversiot kehitettävästä pelistä. Ne voivat olla myös pidemmälle vietyjä laajempia testauksia peli-ideasta.

RPC

Remote Procedure Call mahdollistaa metodien kutsumisen verkon välityksellä toiselta clientiltä. Metodi pitää erikseen merkitä mahdollistamaan RPC-kutsut.

SKYPE

Skype on Microsoft-yhtiön omistama VoIP-palvelu, joka mahdollistaa videopuheluiden soittamisen internetin välityksellä.

TEKSTUURI

Tekstuuri on bittikarttakuvasta koostuva "tapetti", joka toimii 3D-mallin päälle levitettynä. Tekstuureilla saadaan todenmukaisuutta, monimutkaisuutta ja näyttävyyttä ilman, että tarvitsee kasvattaa polygonimääriä.

1 TYÖN LÄHTÖKOHDAT

Mobiilipelit ovat olleet jo vuosien ajan tärkeää liiketoimintaa. Mobiilipelejä pelaavat lähes kaikenikäiset ihmiset, minkä vuoksi siitä on tullut yksi tärkeimmistä pelialan bisnesalueista. Opinnäytetyön tavoitteena oli kartoittaa mobiilipelien luomisprosessi Unity-pelimoottorilla sekä yleiset iOS-pelien kehittämisen ongelmat ja tekniset rajoitukset. Toteutuksessa on pyritty käyttämään ilmaisia, avoimen lähdekoodin sekä pilvipalveluiden ratkaisuja, joiden avulla saadaan minimoitua taloudelliset riskit ja kustannukset.

Tietoperustana työn tekemiseen toimi aikaisempi kokemus Unity-pelikehittämisestä sekä aikaisempi kokemus projektityöskentelystä IT-projekteissa. Sen lisäksi löytyi laaja kokemus peleistä niin konsoleilla, PC:llä kuin mobiililaitteilla.

Työn teoriaosuudessa käsitellään mobiilipelien ja -laitteiden historiaa, perehdytään iOS-käyttöjärjestelmään ja -laitteisiin sekä käydään läpi mobiilipelin nykyinen markkinatilanne. Näiden lisäksi tutustutaan Unity-pelimoottorin toimintaan, rakenteeseen ja käytön perusteisiin.

Lopuksi työn toteutusosassa käydään läpi luodun pelin idea, työskentelymenetelmät, -laitteet ja -välineet sekä yksityiskohtaisesti pelin toteutus ja perehdytään pelissä käytettyihin ratkaisuihin ja mitä optimointeja kannattaa tehdä eri osa-alueilla.

2 MOBIILIPELIT JA -LAITTEET

2.1 Mobiililaitteiden määritelmä

Mobiililaitteiksi määritellään yleisesti älypuhelimet, taulutietokoneet ja kämmentietokoneet. Mobiililaitteille yhteistä on useasti niiden pieni koko, jonka vuoksi niitä voi kuljettaa mukana. Nykypäivän älypuhelimissa on kosketusnäyttö, jonka avulla laitetta käskytetään. Laitteissa on myös gyroskooppi, joka tunnistaa laitteiden asennon ja kääntelyn. Laitteiden kasvanut prosessoriteho on myös mahdollistanut paremman äänikomentojen

luomisen. Myös GPS ja korkealaatuiset kamerat ovat vakio-ominaisuuksia. (Mobile device 2014.)

2.2 Mobiililaitteiden historia

Mobiililaitteiden historia sai alkunsa vuonna 1983, jolloin julkaistiin maailman ensimmäinen matkapuhelin Motorola DynaTAC 8000x. Eurooppaan vasta vuonna 1987 virallistettiin GSM-mobiiliteknologiastandardi, joka mahdollisti laadukkaat puhelut ja tekstiviestien lähettämisen. Suomalainen Nokia julkaisi ensimmäisen GSM-matkapuhelimensa vuonna 1992. (Mobile Phone 2014; GSM 2014.)

Matkapuhelimet yleistyivät 1990-luvulla räjähdysmäisesti. Tämä mahdollisti mobiililaitteiden nopean kehityksen, ja jo vuonna 1994 IBM julkaisi ensimmäisen kämmentietokoneen, jossa oli täydelliset puhelinominaisuudet. Myös Nokia julkaisi oman näkemyksensä kämmentietokoneesta kaksi vuotta myöhemmin. Nokia 9000 Communicator oli puhelin, jossa oli verkkoselain, faksi ja sähköposti sekä iso kunnollinen näyttö ja näppäimistö. (PDA 2014.)

2000-luvulla matkapuhelimet kehittyivät entistä nopeammin. Puhelimiin tulivat värinäytöt, paremmat soittoäänet, MP3-äänitiedostojen tuki sekä enemmän tehoa ja muistia. Matkapuhelimien muotoilu kehittyi ja puhelimista tehtiin pienempiä. Ensimmäinen kamerapuhelin julkaistiin vuosituhannen alussa Japanissa (Sharp J-SH04: World's First Ever Phone With Integrated Camera 2010).

Vuonna 2003 Nokia julkaisi N-Gagen, joka oli ensimmäinen pelaamiseen tarkoitettu puhelin. Puhelin ei ollut myyntimenestys Nokialle, koska käyttäjät pitivät puhelinta suurikokoisena ja jähmeänä. Vuotta myöhemmin Nokia julkaisi puhelimestaan uudistetun mallin Nokia N-Gage QD:n, joka oli huomattavasti pienempi, mutta samalla menetti ominaisuuksia kuten radion ja musiikkisoittimen. (Nokia N-Gage QD - The Same, But Different 2004.)

Vuonna 2007 Apple julkaisi ensimmäisen kosketusnäytöllisen puhelimen, iPhonen, joka käytti heidän itse kehittämäänsä iOS-käyttöjärjestelmää. Tämä oli edistyksellinen puhelin erityisesti tehon, käyttöliittymän ja uudenlaisen kommunikointitavan takia. iPhone oli

todella suuri edistysaskel matkapuhelimissa ja oli suunnannäyttäjä nykypäivän älypuhelimille. Googlen vastaus tähän oli Android-käyttöjärjestelmä, joka julkaistiin vuonna 2008 HTC Dream -puhelimessa. Windows Phone 7 oli Microsoftin vastaus, joka julkaistiin vasta vuonna 2010. (Mobile operating system 2014.)

Vuonna 2011 Nokia ja Microsoft ilmoittivat tekevänsä yhteistyötä, ja samalla Nokian älypuhelinstrategiaksi tulisi Windows Phone. Tämä käytännössä tarkoitti sitä, että Nokia luopui oman käyttöjärjestelmänsä kehittämisestä ja yhtiön uudet puhelimet käyttäisivät Windows Phone -käyttöjärjestelmää. (Nokia Strategy 2011.)

Vuonna 2010 julkaistiin ensimmäiset taulutietokoneet, kuten Applen iPad. Taulutietokoneista ilmestyi useita versiota. Taulutietokoneet ovat horjuttaneet perinteisen tietokoneen markkinoita, ja monet tietokoneyritykset ovat joutuneet tekemään radikaaleja muutoksia yhtiössään. Apple myi vuoden 2014 ensimmäisellä neljänneksellä 26 miljoonaa iPadia. (First Quarter Results 2014.)

2.3 Mobiililaitteet nykypäivänä

Mobiililaitteet ovat horjuttaneet jo pidemmän aikaa perinteistä tietokoneteollisuutta. Yhä useampi käyttäjä ostaa ennemmin taulutietokoneen kuin perinteisen tietokoneen. Suomessa myytiin vuonna 2013 lähes 700 000 taulutietokonetta. Tämän lisäksi samana vuonna myytiin yli 1.8 miljoonaa kappaletta älypuhelimia. (Älypuhelimia ja tabletteja myytiin vuonna 2013 yhteensä lähes miljardilla eurolla 2014.)

2.4 Mobiilipelin määritelmä

Mobiilipeli on digitaalinen peli, joka on erityisesti suunniteltu mobiililaitteille ottaen huomioon alustan asettamat tekniset vaatimukset ja rajoitteet, kuten näytön koon, muistin määrän ja prosessorin tehon. Mobiilipeleihin eivät sisälly käsikonsolien kuten Nintendo DS:n pelit. Kuitenkin on olemassa monia pelejä, jotka on alunperin suunniteltu jollekin muulle alustalle, mutta joita voi silti pelata mobiililaitteilla. Myös uudet webteknologiat hämärtävät muiden pelien ja mobiilipelien rajaa. (Mobiilipeli 2013.)

2.5 Mobiilipelien historia

Ensimmäisiä julkaistuja mobiilipelejä oli Snake, joka näki päivänvalon vuonna 1997. Snake on Nokian julkaisema peli, ja se on siitä lähtien ollut saatavilla aina tähän päivään asti erilaisina versioina. Kyseistä Snake-peliä väitetään maailman ensimmäiseksi mobiilipeliksi, vaikka jo vuonna 1994 Tetris-peli oli julkaistu Hagenuk MT-2000 -laitteessa. (The Evolution of Mobile Games 2013; Mobile Game 2014.)

Vuosituhannen vaihteessa mobiilipelit alkoivat kiinnostaa isompaa yleisöä, koska matkapuhelimiin tuli värinäytöt, enemmän muistia ja nopeammat prosessorit. Tämän vuoksi mobiilipeleihin alettiin sijoittamaan enemmän rahaa. Mobiilipelien kehitys saavutti nopeasti pisteen, jossa suuret julkaisijat hyppäsivät mukaan. Useat sen aikaiset pelijulkaisijat julkaisivat suosituimmista PC-peleistään mobiiliversiot. (The Evolution of Mobile Games 2013.)

Suurin käännekohta mobiilipelien historiassa tapahtui vuonna 2007, kun Apple julkaisi kosketusnäytöllisen iPhonen. Vuonna 2008 Apple avasi App Store -nimisen jakelupalvelun laitteillensa, joka mahdollisti pelien ja sovellusten helpon jakamisen. App Storen avulla kehittäjät saivat sovelluksensa suoraan käyttäjille ilman välikäsiä. (Emt.)

Vuonna 2008 ilmestyivät ensimmäiset iPhone-pelit. Vuonna 2009 suomalainen Rovio julkaisi maailmalla menestyneen Angry Birds -pelisarjan ensimmäisen osan. Pelisarjaa on ladattu yli miljardi kertaa (Angry Birds Reaches 1 Billion Downloads 2012).

2.6 Mobiilipelit nykypäivänä

Nykypäivän älypuhelimet ovat suoritustasoltaan kasvaneet huomattavasti aikaisempaan verrattuna. Tämän seurauksena mobiililaitteilla alkaa näkyä entistä näyttävämpiä ja laajempia pelejä. Nopeat 3G- ja 4G-yhteydet sekä yleistyneet langattomat verkot ovat mahdollistaneet suurempienkin pelien julkaisut.

Mobiilipelimarkkinat ovat laajentuneet ja menestyneet hyvin. Applen App Storessa on tällä hetkellä yli miljoonaa sovellusta ja ladattuja sovelluksia on yli 60 miljardia. (Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio 2013.)

Mobiilipelien perinteinen ansaintamalli on perustunut pelin ostohinnan maksamiseen. Tämä on nykypäivänä alkanut katoamaan, sillä nykyinen trendi mobiilipelien ansaintamallina on Free to Play. Tämä tarkoittaa sitä, että itse peli on ilmainen, mutta pelin edetessä pelaamista voi helpottaa tai jouduttaa peliin ostettavalla rahalla.

3 IOS-KÄYTTÖJÄRJESTELMÄ JA -LAITTEISTOT

3.1 Mikä iOS on?

iOS on Applen kehittämä käyttöjärjestelmä mobiililaitteistolle. iOS perustuu Darwin BSD -käyttöjärjestelmään. Apple julkaisi ensimmäisen version käyttöjärjestelmästään kesäkuussa 2007. (iOS 2014.)

Käyttöjärjestelmä on suljettu, minkä vuoksi iOS-ohjelmistot täytyy kirjoittaa Applen omilla työkaluilla. Jotta ohjelmistoja tai pelejä voi luoda, tarvitsee rekisteröityä Applen kehittäjäksi. Rekisteröityminen maksaa kirjoitushetkellä noin 100 dollaria vuodessa, joka avaa tien mobiilimaailmaan. Tällä rahalla saa kaikki tarvittavat lisenssit ja oikeuden julkaista ohjelmia Applen mobiililaitteille. Ohjelmia voi virallisesti jakaa vain Applen omassa App Store -sovelluskaupassa. Ohjelmoidakseen iOS-laitteille kehittäjä tarvitsee Applen tietokoneen. (iOS Developer Program 2014.)

iOS -käyttöjärjestelmässä ei ole tukea Flash- tai Java-tekniikoille. Tämän vuoksi monet käyttäjät valitsevat kilpailijoiden tuotteita, kuten Googlen Android-laitteita.

3.2 iOS-käyttöjärjestelmä versiohistoria

Apple julkaisi 2009 ensimmäisen ison käyttöjärjestelmäpäivityksen, joka toi mukanaan koko käyttöjärjestelmän kattavan haun sekä tekstin leikkaamisen ja liittämisen. Päivitys julkaistiin Applen iPhone 3G S:n mukana. (iPhone 2014.)

Seuraava suuri päivitys tuli vuonna 2010, kun Apple julkaisi iOS 4 -käyttöjärjestelmän. Päivitys oli Applen suurin, ja se toi yli 100 uutta ominaisuutta käyttöjärjestelmään. Päivityksen tärkein uusi ominaisuus oli parannettu moniajo, joka mahdollisti paremman ja nopeamman monen ohjelmiston samanaikaisen käytön. (Emt.)

Lokakuussa 2011 Apple julkaisi seuraavan suuren päivityksensä, iOS 5:n. Päivitys piti sisällään tietoturvaan liittyviä muutoksia, ja suurimpana uutuutena tuli Applen iCloud-pilvipalvelu, joka mahdollisti kuvien, tiedostojen yms. pitämisen Applen palvelimella, jolloin ne olivat saatavilla joka paikasta. (Emt.)

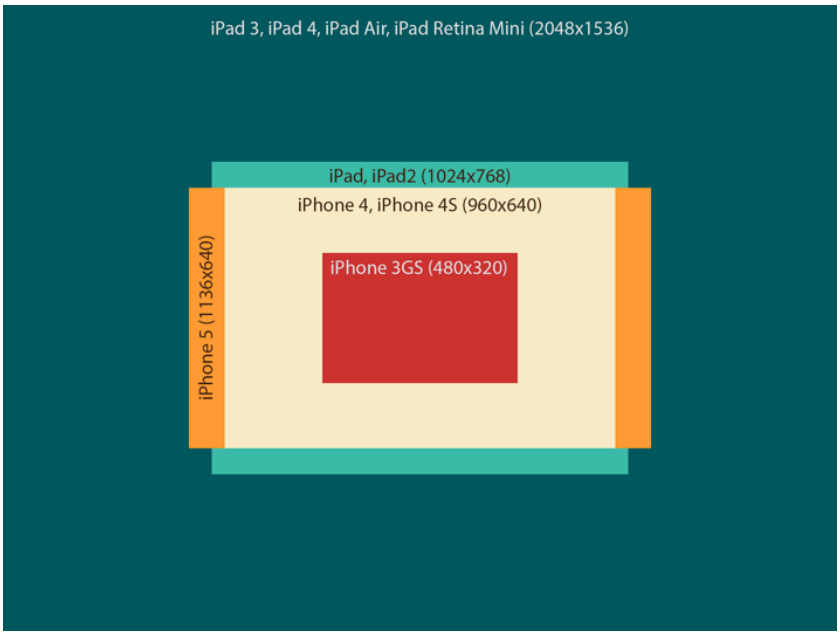
Syyskuussa 2012 Apple julkaisi iOS 6 -käyttöjärjestelmän, jonka luvattiin sisältävän yli 200 parannusta. Se julkaistiin Apple iPhone 5 -älypuhelimen kanssa. Käyttöjärjestelmäpäivitys ei tuonut suuria muutoksia ohjelmistokehittäjille. (Emt.)

Vuonna 2013 Apple julkaisi suurimman päivityksen iOS-käyttöjärjestelmäänsä. iOS 7 uudisti laitteiston ulkoasun täysin ja toi lukuisia uusia ominaisuuksia. iOS 7 sisälsi ensimmäistä kertaa standardin peliohjaimille, joka mahdollisti uudenlaisen tavan suunnitella pelejä iOS-laitteille. (Emt.)

3.3 Yleistä iOS-laitteistoista

Ensimmäinen iOS-laite eli iPhone julkaistiin kesäkuussa 2007 (iPhone 2014). iOS-laitteita käytetään pääsääntöisesti pelkällä kosketuksella. Laitteista löytyy yhteensä viisi fyysistä näppäintä. Nämä ovat äänenvoimakkuudensäädin, äänenvaimennuspainike, virtapainike sekä kotipainike, jolla pääsee ohjelmista ulos.

iOS-laitteiden näyttöjen resoluutiot eroavat toisistaan paljon, kuten kuviosta 1 nähdään. Peligrafiikkaa luodessa täytyy ottaa huomioon eri laitteiden resoluutiot, sillä jos peligrafiikkaa luodaan Apple iPad Airille ja samaa käytetään Apple iPhone 4 -laitteelle, laitteen resursseja käytetään turhaan.



Kuvio 1. iOS-laitteiden resoluutiot suhteutettuna toisiinsa

3.4 iOS-puhelimien tekniset ominaisuudet

iPhone-malleja on julkaistu yhteensä kahdeksan maaliskuuhun 2014 mennessä. Kaikki, paitsi ensimmäinen, tukevat 3G-yhteyksiä. Laitteet tukevat Bluetooth-yhteyksiä. Taulukossa 1 on eritelty iOS-puhelimien laitetiedot ja julkaisuvuodet.

Taulukko 1. iPhone-mallien vertailu (iPhone 2014)

	iPhone 4	**iPhone 4 S**	**iPhone 5/5C**	**iPhone 5S**
Vuosi	2010	2011	2012/2013	2013
Resoluutio	960x640	960x640	1136x640	1136x640
Prosessori	ARM Cortex-A8	ARM Cortex-A9	Apple A6	Apple A7
RAM	512 MB LPDDR2	512 MB LPDDR2	1GB LPDDR2	1GB LPDDR3

3.5 iOS-taulutietokoneiden tekniset ominaisuudet

iPad-malleja on julkaistu yhteensä seitsemän maaliskuuhun 2014 mennessä. Taulukossa 2 on eritelty eri iOS-taulutietokoneiden laitetiedot ja julkaisuvuodet. Kaikista malleista on saatavilla WiFi- ja 3G-versiot. Laitteet tukevat Bluetooth-yhteyksiä ja niissä on useita eri sensoreita, kuten kiihtyvyys-, etäisyys-, ja ympäröivän valon mittaussensorit sekä iPad 2:sta lähtien gyroskooppi. Apple ilmoittaa akkukeston olevan jopa yli 10 tuntia. (Compare iPad Models 2014.)

Taulukko 2. iPad-mallien vertailu (iPad 2014)

	iPad 2	**iPad 3**	**iPad Mini**	**iPad 4**	**iPad Air**	**iPad Mini Retina**
Vuosi	2011	2012	2012	2012	2013	2013
Reso-luutio	1024x768	2048x1536	1024x768	2048x1536	2048x1536	2048x1536
Proses-sori	ARM Cortex-A9	ARM Cortex-A9	ARM Cortex-A9	Apple A6X	Apple A7	Apple A7
RAM	512 MB DDR2	1 GB LPDDR2	512MB DDR2	1GB LPDDR2	1GB LPDDR3	1GB LPDDR3

4 UNITY

4.1 Mikä Unity on?

Unity on Unity Technologies -yhtiön luoma pelimoottori. Sen avulla voidaan luoda sekä 2D- että 3D-pelejä. Unity tarjoaa ympäristön, jossa voi nopeasti ja helposti luoda pelattavia peliprototyyppejä.

Unity on kehitetty erityisesti pelin kehitykseen, ja se sisältää työkalut, joiden avulla voi nopeasti luoda pelattavan maailman. Unityn toimintaperiaatteet ottavat huomioon tärkeimmät asiat pelinkehityksessä. Unityn käyttöliittymä on helposti laajennettavissa itse, ja se sisältää todella laajan valikoiman valmiita työkaluja.

Unityllä pystyy julkaisemaan usealle eri alustalle käyttäen lähes samaa koodia. Yleensä muutokset ovat laitekohtaisia optimointeja sekä ominaisuuksien käyttöönottoa. Unity on niin ajallisesti kuin rahallisestikin hyvä valinta, koska peliä ei tarvitse tehdä erikseen jokaiselle eri kohdelaitteelle. (Multiplatform 2014.)

Unityn ilmaisiin julkaisuvaihtoehtoihin kuuluu selaimessa ajettava Web Player, PC, Linux, OS X sekä mobiilikäyttöjärjestelmät Android, iOS ja Windows Phone. Maksullisen lisenssin vaativat laajennettu Android, BlackBerry, iOS ja Windows Phone 8 sekä Nintendo Wii, Nintendo Wii U, Playstation 3 ja Xbox360. (Emt.)

Unity myös tarjoaa tilausmallin Unityn maksullisille tuotteille, jolloin maksat joka kuukausi pienen erän sen sijaan, että maksaisit koko lisenssin kerralla. Tämän avulla aloituskynnys aloittaa pelien tekeminen on laskenut huomattavasti aikaisempaan verrattuna.

4.2 Historia

Unityn historia alkoi vuonna 2005, jolloin siitä julkaistiin ensimmäinen versio Applen pitämässä kansainvälisessä kehittäjäkonferenssissa. Se oli alunperin luotu pelkästään Mac OS X -käyttöjärjestelmälle, ja saamansa hyvän huomion ansiosta siitä alettiin luoda versioita muille alustoille. (Unity (game engine) 2014.)

Ajan myötä Unity-pelimoottori on kasvattanut mainettaan, ja siitä onkin tullut maailman suosituin pelimoottori indie-kehittäjien keskuudessa. Unity oli jo vuonna 2012 käytetyin pelimoottori mobiilipelien luomisessa. (Fast Facts 2014.)

Seuraavaan listaan on koottu merkittävimmät päivitykset Unityn eri versioissa. (Release Archive 2014.)

Versio 1

- Unityn viralliset dokumentaatiot päivitettiin
- Sisäänrakennetut shaderit lisättiin

Versio 2

- Uusi paranneltu Web Player
- Windows-tuki
- Maastoeditori
- DirectX 9 -tuki
- Unityn sisäänrakennettu profilointityökalu Profiler

Versio 3

- Tehokas uudistettu partikkelijärjestelmä
- Sisäänrakennettu reitinhaku
- HDR-renderointi
- Sisäänrakennettu ”Level of Detail”, eli LOD-työkalu
- Näytönohjaimen profilointityökalu

Versio 4

- 2D-työkalut
- DirectX 11 -tuki
- Mobiiliparannuksia, kuten reaaliaikaiset varjot
- Linux-tuki

4.3 Miksi Unity?

Unity-pelimoottori on kehitetty erityisesti pelien tekemiseen ja se sisältää kaikki tarvittavat työkalut, mitä pelin tekemiseen tarvitaan. Unity sisältää valmiiksi toteutettuja muokattavia komponentteja, valmiin fysiikkamoottorin, laajan tuen eri 3D-malliformaateille ja muita tavanomaisia toiminnallisuuksia. Unity käyttää niin sanottua komponenttimallia, jolloin jokainen ominaisuus on oma komponenttinsa. Tämän myötä on helppo luoda uudelleenkäytettäviä komponentteja, joka vähentää kehityskustannuksia pidemmällä aikavälillä.

Unityn ylläpitämä Asset Store -palvelu mahdollistaa pelin tekemisen myös muille kuin ohjelmoijille. Asset Storen kautta löytyy paljon valmisratkaisuja, joiden avulla voi luoda paljon erilaisia pelejä. Unity myös itse tuottaa laadukkaita tutoriaaleja pelimoottorinsa markkinoimiseen, joita he jakavat Asset Storen avulla.

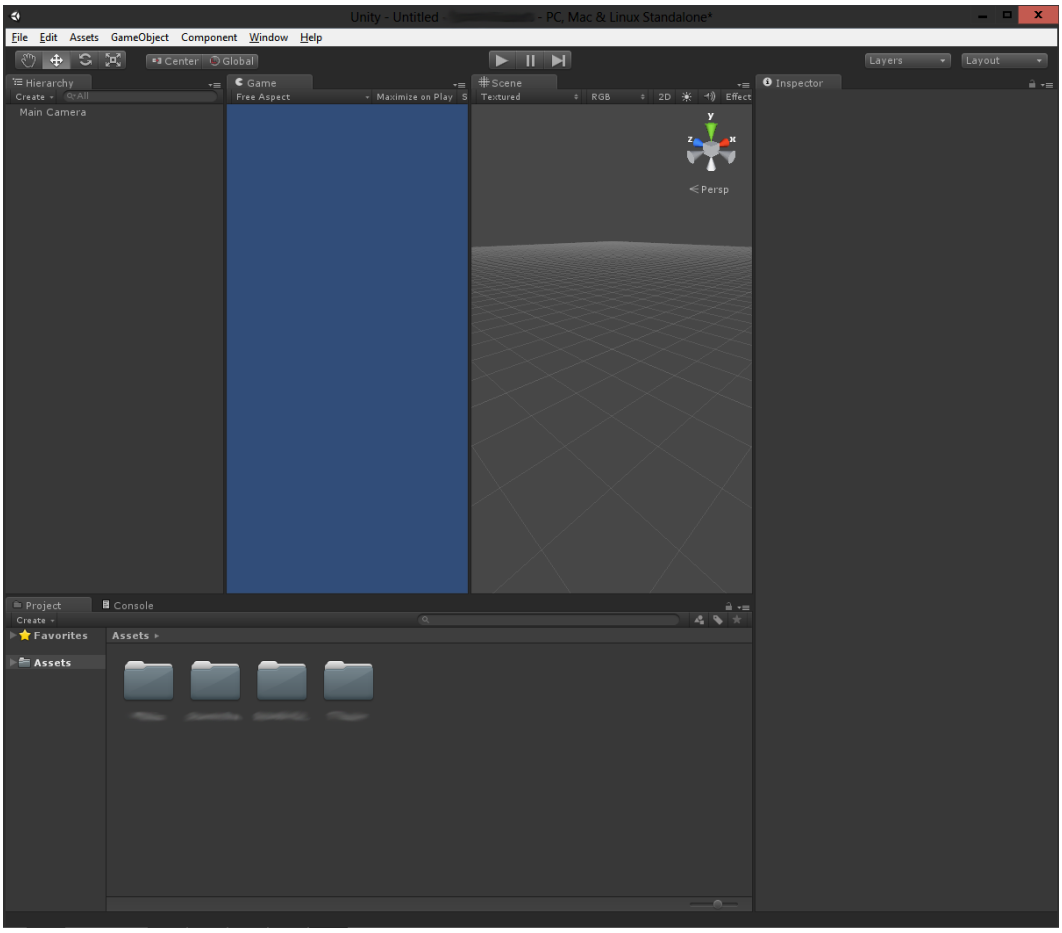
Unity mahdollistaa myös yksinkertaisen rajapinnan useille eri alustoille julkaisemiseen. Vaikka peliprototyyppiä lähdettiin tekemään, ei haluttu valita teknologiaa, jolla ei ole mahdollista kääntää muille alustoille helpolla. Unityllä muille alustoille kääntäminen ei vaadi erikoisosaamista, eikä suuria koodimuutoksia.

Se on saanut lyötyä itsensä läpi pelinkehitysmaailmassa, koska se tukee laajaa laitekantaa, on helposti lähestyttävissä, tarjoaa laajat tutoriaalit erilaisten asioiden luomiseen sekä on edullinen.

4.4 Unity Editor

Editor on Unityn päänäkymä, jossa peli koostetaan kasaan. Se on sekä ohjelmoijan, audiosuunnittelijan, kenttäsuunnittelijoiden, graafikoiden sekä mahdollisesti myös käsikir-

joittajien työkalu. Editorin perusnäkymä on selkeä ja siihen pääsee nopeasti sisään. Kuviossa 2 nähdään editorin perusnäkymä. Unityn vahvuus on siinä, että editori on modulaarinen, jolloin sen voi muokata omanlaisekseen. Editoria voi myös laajentaa omilla skripteillään.



Kuvio 2. Unity-editorin perusnäkymä.

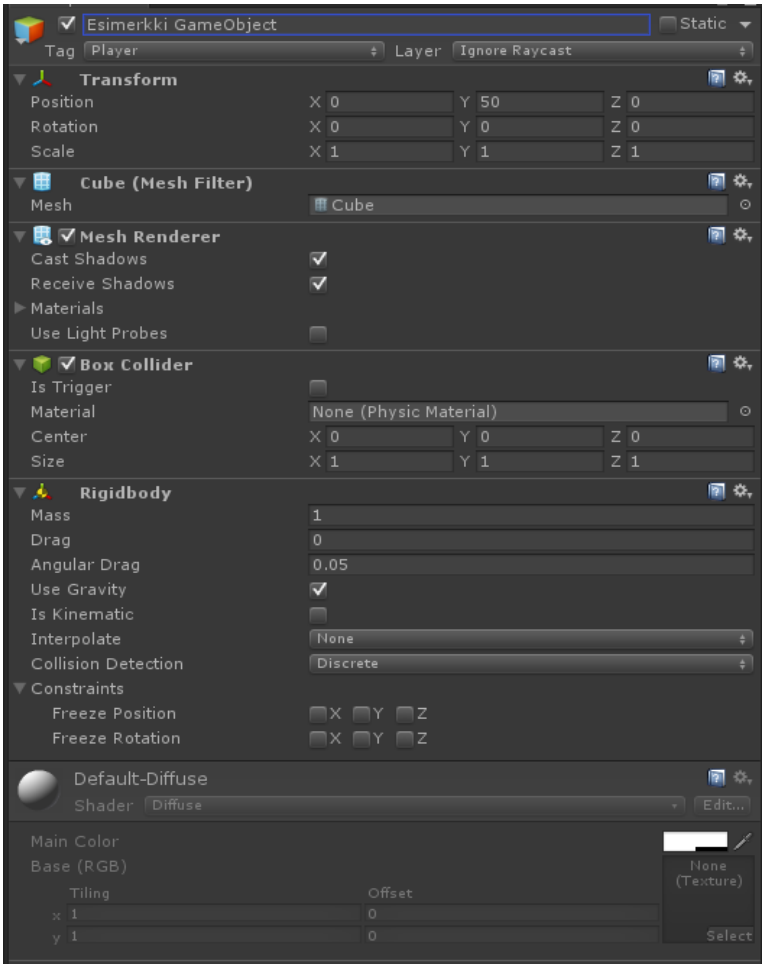
Editorin perusnäkymä koostuu tärkeimmistä asioista, mitä tarvitsee käyttääkseen Unityä. Perusnäkymässä näkyvät projektin tiedostot, scenessä olevat peliobjektit, pelinäkymä, scene-näkymä sekä Inspector-työkalu, jolla näkee tiedot valituista peliobjekteista.

Unity-editorissa on muutama tärkeä painike, jotka pitää tietää käyttääkseen editoria. Nämä painikkeet ovat:

- Käsityökalu, jota käyttämällä käyttäjä voi liikuttaa kameraa scene-näkymässä. 
- Siirtotyökalu, jolla voi siirtää peliobjektia pelimaailmassa. 

- Kääntämistyökalu, jonka avulla voi kääntää peliobjektia. 
- Skaalaustyökalu, jolla objektin kokoa voi skaalata. 

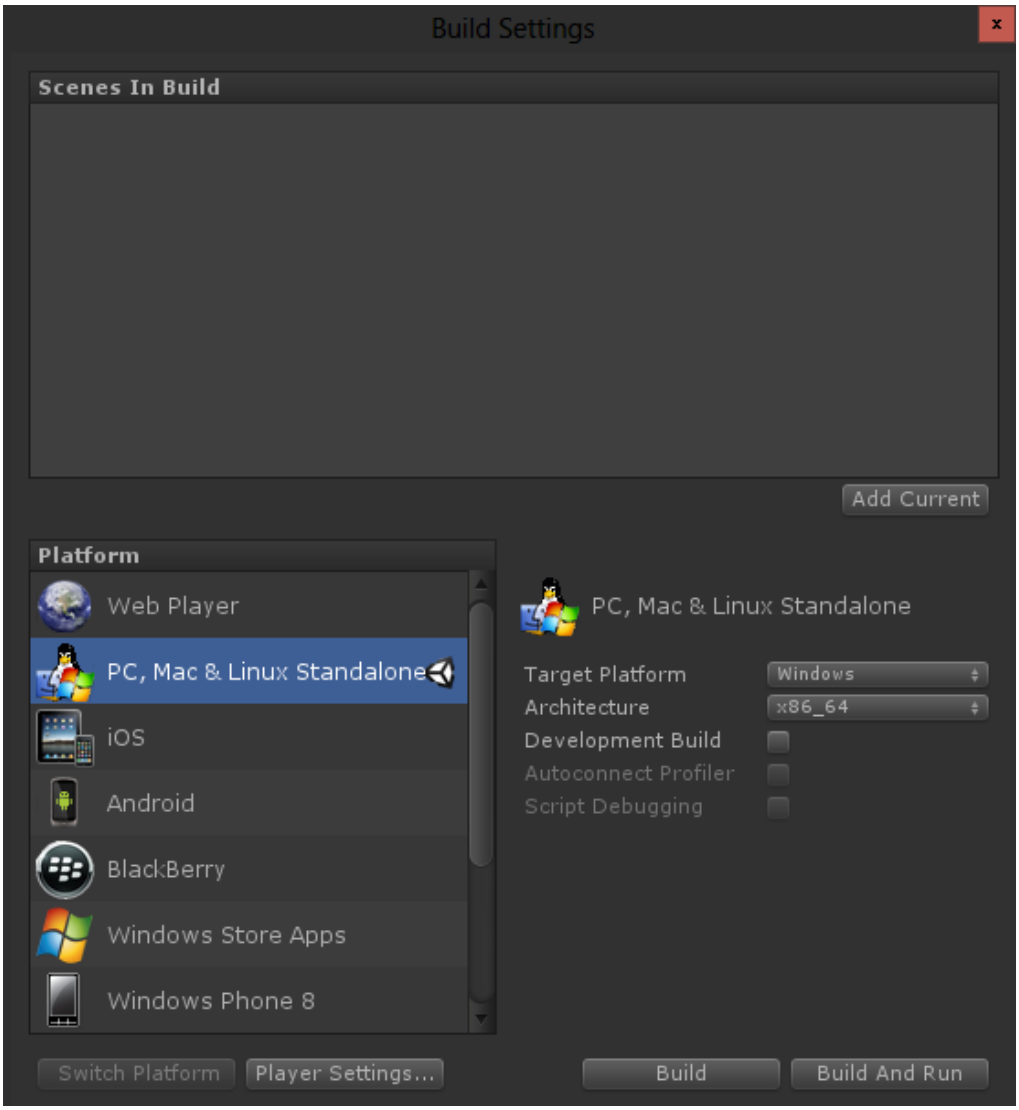
Peliobjekteja säädetään Unity Editorin kautta käyttäen Inspector-työkalua. Kuviossa 3 nähdään Inspector-työkalu käytössä sekä peliobjektin perusrakenne.



Kuvio 3. Unityn Inspector työkalu ja peliobjektin perusrakenne

Editorin kautta säädetään myös pelin erilaiset buildaus–asetukset, pelin eri laatutasot ja fysiikoiden asetukset.

Build-asetuksissa valitaan kohdealusta, mitkä scenet sisältyvät buildiin ja asetetaan buildille kehitysversiotagi päälle, joka taas mahdollistaa buildatun tuotteen profiloimisen (ks. kuvio 4).



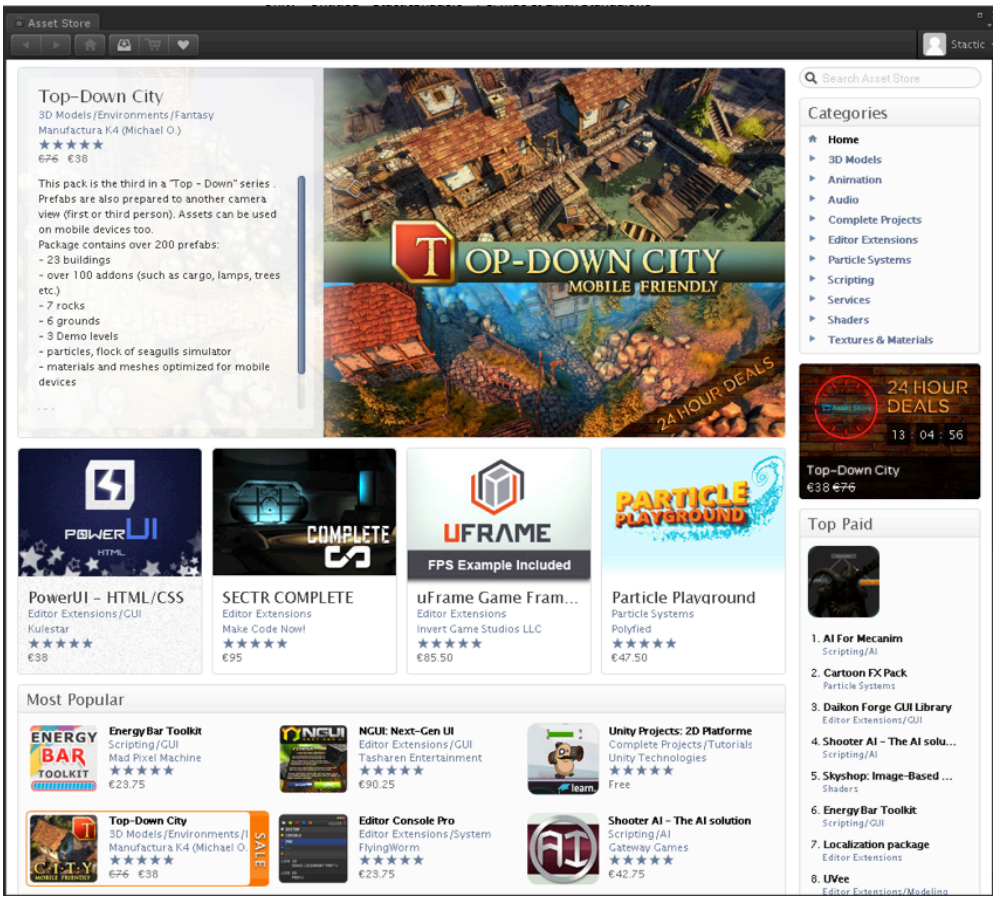
Kuvio 4. Unityn kääntämisasetusten valitsemisnäkymä

Kuviossa 5 esitellään laatutason asetukset. Asetuksissa voi luoda useita eri laatutasoja, joissa säädetään valojen, tekstuurien, varjojen, animoinnin ja eri optimointien asetuksia. Asetuksissa voidaan myös asettaa valmiiksi jonkun tietyn perustason eri alustoille.

Kuvio 5. Unityn laatutasojen asetukset

4.5 Asset Store

Marraskuussa 2010 Unity avasi Asset Store -palvelun, jossa käyttäjät voivat jakaa tekemiään assetteja muille käyttäjille niin ilmaiseksi kuin maksua vastaan. Unity ottaa 30 prosenttia myyntituloista itselleen. Kaupassa on nykypäivänä suuri määrä asset-paketteja, kuten tekstuureja, materiaaleja, valmiita ratkaisuja pelilogiikkaa varten ja 3D-malleja. Lisäksi kaupassa jaetaan aloittelijoille suunnattuja paketteja, kuten tutoriaaleja ja esimerkkiprojekteja. Kuviossa 6 näkyy Asset Storen aloitusnäkymä. (Sell Assets 2014.)



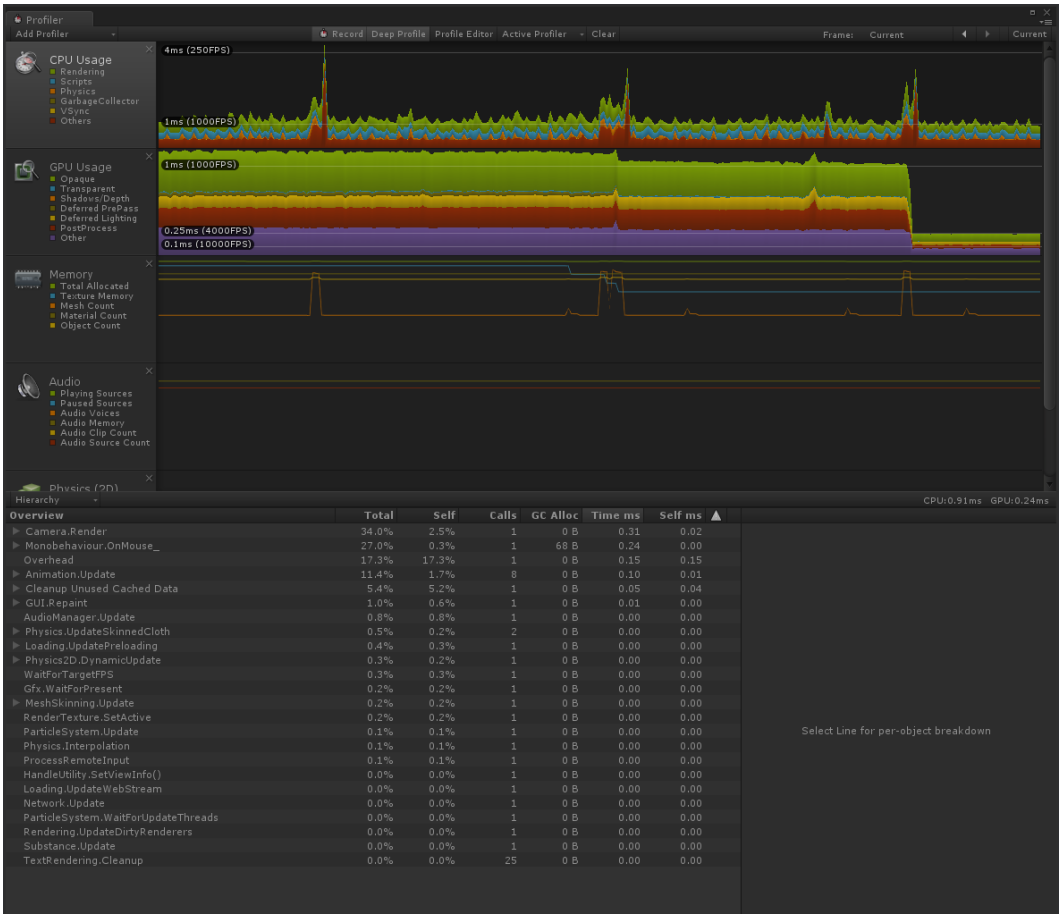
Kuvio 6. Unity Asset Storen aloitusnäkymä

Asset Storessa on kirjoitushetkellä saatavilla yhteensä yli 10 000 erilaista asset-pakettia. Suosituimpia paketteja ovat NGUI ja PlayMaker. Asset Storen hinnat ovat yleisesti ottaen matalat, joka mahdollistaa pienillekkin studioille paremmat mahdollisuudet, koska voivat halutessaan käyttää hieman rahaa valmisiiin ratkaisuihin, eikä kaikkea tarvitse tehdä itse.

4.6 Profiler

Unityn sisään on rakennettu valmis profilointityökalu Profiler. Profiler raportoi kuinka paljon suorituskykyä mikäkin osa-alue vie. Profilerin avulla saa tarkkaa tietoa ulos CPU:n käytöstä, GPU:n käytöstä, renderöinnistä, fysiikoista, muisteista ja äänistä. Profiler piirtää reaaliaikaisesti käyrät eri osa-alueille, joiden avulla näkee, missä tapahtuu suoritus-

kykyongelmia. Profilerin avulla saa informaatiota frame-kohtaisesti. Kuviossa 7 nähdään Profiler-työkalun perusnäkymä, jossa on valittuna profiloitavaksi prosessorin käyttö.



Kuvio 7. Unityn sisäänrakennetun Profiler-työkalun yleisnäkymä

CPU-osion alla näkee tarkat tiedot kauanko renderointi, komentosarjojen läpikäynti, roskienkeruu ja fysiikat vievät aikaa. Deep Profile -asetuksen aktivoimisella näkee entistä tarkempaa informaatiota, kuinka kauan eri skriptien suorittamiseen menee aikaa ja missä pullonkaulat ovat.

GPU-näkymässä tarkat tiedot näkyvät erilaisten materiaalien renderoinnin kestoista ja määristä sekä muuta tärkeää tietoa näytönohjaimen toiminnasta. GPU-näkymän avulla saadaan tietoa, jaksaako näytönohjain pyörittää erilaisia shader-ratkaisuja vai pitääkö shadereita yksinkertaistaa.

Muissa osissa näkee piirrettävien polygonien, verteksien ja draw callsien määrän sekä muistin kulutuksen, äänien määrän ja perusinformaatiota fysiikkamoottorilta.

4.7 Unityn toimintaperiaate

4.7.1 Ideologia

Unity pohjautuu komponenttipohjaiseen ratkaisuun. Tämä käytännössä tarkoittaa sitä, että Unityllä tehdyt pelit rakennetaan sceneistä jotka koostuvat peliobjekteista.

4.7.2 GameObject

Unity-pelimoottori käyttää GameObjecteja eli peliobjekteja. Pelit rakennetaan peliobjekteista, jotka kaikki voidaan määritellä erilaisiksi. Peliobjekti itsessään ei pidä mitään muuta sisällään kuin paikan maailmassa, mutta niihin voidaan liittää komponentteja, joten niitä voidaan kutsua eräänlaisiksi komponenttisäiliöiksi.

Peliobjektiin lisätään erilaiset komponentit, kuten skriptit, valolähteet tai colliderit. Peliobjekteihin asetetut komponentit voivat keskustella keskenään. Peliobjekteille voidaan asettaa oma tagi, taso ja nimi sekä niistä voidaan luoda valmisolio eli prefab.

4.7.3 Prefab

Prefab eli valmisolio on uudelleen käytettävä assetti ja se luodaan halutusta peliobjektista. Prefabia luodessa se kopioi kaikki ominaisuudet, komponentit ja arvot peliobjektista. Peliobjekteista luodaan prefabeja, koska halutaan saada uudelleen käytettäviä komponentteja sekä prefabeja voidaan luoda myös pelin aikana. Prefabit ovat hyödyllisiä esimerkiksi kerättävien esineiden, vihollisten, ammusten, rakennelmien tai muiden usein toistuvien asioiden luonnissa.

Sen sijaan, että skriptissä luotaisiin uusi peliobjekti, johon lisätään komponentteja ja määritellään kaikki tieto koodissa, luodaankin vain kopio prefabista. Prefabien avulla artistit voivat muokata prefabia suoraan editorinäkymässä eikä heidän tarvitse pyytää ohjelmoijaa muuttamaan esimerkiksi vihollisen kokoa tai väriä. (Instantiating Prefabs 2014.)

Kaikki prefabbeihin tehdyt muutokset tapahtuvat myös sen kopioissa, vaikka jokainen kopio onkin itsenäinen peliobjekti pelimaailmassa. (Prefab 2014.)

4.7.4 Tag

Tagit ovat lista tunnistimia, jossa on Unityn omia sekä käyttäjän itsensä määrittelemiä nimiä. Peliobjektit erotellaan toisistaan tageilla. Tagien avulla voidaan myös verrata, onko esimerkiksi ammus osunut tietyn tyyppiseen kohteeseen kuten pelaajaan.

4.7.5 Layer

Layerit eli tasot ovat Unityn toinen tapa erotella asioita toisistaan. Unityssä on sen omia tasoja sekä niitä voidaan luoda itse. Tasojen avulla voidaan eritellä erilaiset asiat toisistaan, esimerkiksi staattiset maailmassa olevat objektit, kuten kivet, ja dynaamiset objektit, kuten liikkuvat hahmot. Tästä on suuresti hyötyä fyysikoiden kanssa, jos halutaan, että tietyt tasot eivät osu toisiinsa. Kuviossa 8 näkyy fysiikoiden törmäysmatriisi, jonka avulla määritellään mitkä tasot osuvat toisiinsa ja mitkä eivät.

Kuvio 8. Törmäysmatriisi

4.7.6 Scene

Scenet eli pelimaailmat koostuvat peliobjekteista. Scenet sisältävät tiedon siitä, mitä halutaan tapahtuvan ja näkyvän pelissä. Scenet yleensä sisältävät pelimaailman geometrian, kamerat, valot, hahmot ja käyttöliittymän, mutta voidaan myös luoda scenejä, joissa on vain käyttöliittymä tai esimerkiksi lisää geometriaa maailmaan.

Scenejä muokataan editorinäkymässä. Scenen sisältämiä peliobjekteja voidaan poistaa tai lisätä sekä peliobjektien komponenttien arvoja voidaan muuttaa. Normaalisti kaikki peliobjektit tuhoutuvat scenen vaihdon yhteydessä, mutta objekteja voidaan asettaa skripteissä olemaan tuhoutumatta käytämällä DontDestroyOnLoad-metodia.

4.8 Ulkoiset työkalut

Unityn mukana tulee vain ohjelmistokehitysympäristö, mutta Unityyn voidaan integroida lukuisia muita ohjelmia, kuten Microsoft-yhtiön valmistama Visual Studio –kehitysympäristö sekä eri kuvankäsittelyohjelmia. Unity tukee myös ulkoista versionhallintaa.

Unityn mukana tuleva ohjelmistokehitysympäristö MonoDevelop on hieman muokattu versio oikeasta MonoDevelopista, sillä sen avulla voi itse testata Unityssä tehtyä peliä. MonoDevelop mahdollistaa ohjelmoimisen ja ohjelmakoodin debuggauksen.

4.9 Ohjelmointi

Unity tukee kolmea eri ohjelmointikieltä. Käytetyin kieli on C#, joka on alkuaan Microsoft–yhtiön kehittämä kieli. Unityn C# tulee hieman jäljessä verrattuna Microsoftin omaan, mutta kaikki tärkeimmät ominaisuudet löytyvät. Toiseksi käytetyin kieli on UnityScript, joka on johdannainen JavaScript-kielestä, ja viimeisenä löytyy Boo-kieli, joka muistuttaa vahvasti Python-ohjelmointikieltä. (Unity (game engine) 2014.)

Unityllä ohjelmoidessa ei aina voi käyttää kaikkia perinteisiä olio-ohjelmoinnin tapoja. Unityssä on poikkeuksellinen tapa, ettei muodostimia (Constructoreita) voi luoda silloin, kun luokka perii Unityn MonoBehaviour-luokan. Jokainen luokka, joka on perinyt MonoBehaviour-luokan, sisältää Unityn perusominaisuudet ja mahdollistaa valmiiden kompo-

nenttien käytön. MonoBehaviourin perineet skriptit voivat toteuttaa monta eri rajapintaa. MonoBehaviourin perineen skriptin voi yhdistää peliobjektiin editorinäkymässä tai ohjelmakoodissa. Yhdelle peliobjektille pystyy lisäämään monia skriptejä. Skriptiin voidaan myös kirjoittaa, että se vaatii jonkun muun komponentin.

Unity-skriptaus tapahtuu kirjoittamalla ohjelmointikoodit MonoBehaviourin perivään luokkaan. MonoBehaviourin perineet luokat voivat toteuttaa valmiita metodeja, joita Unity-pelimoottori kutsuu automaattisesti. Taulukossa 3 esitellään tärkeimmät MonoBehaviour-metodit sekä niiden käyttötarkoitus. Näistä metodeista tärkeimmät ovat Awake, Start, Update, FixedUpdate ja LateUpdate. (MonoBehaviour 2014.)

Taulukko 3. MonoBehaviour tyyppisten metodien käyttötarkoitukset avattu

Metodi	Käyttötarkoitus
Awake	Awake-metodi kutsutaan, kun skriptin instanssi on luotu. Awake-metodia voi ajatella eräänlaisena muodostimena, jossa skripti alustaa itsensä.
Start	Kutsutaan vain kerran juuri ennen ensimmäistä Update-kutsua. Start-metodissa haetaan viittaukset muihin peliobjekteihin ja komponentteihin, koska silloin ollaan varma, että ne ovat alustaneet itsensä.
Update	Kutsutaan joka frame kerran.
FixedUpdate	Kutsutaan fysiikoiden tahdissa, eli ei ole riippuvainen ruudunpäivitysnopeudesta. Käyttökohteena esimerkiksi pelaajan päivittäminen.
LateUpdate	Kutsutaan joka frame Update-metodien jälkeen. Käytännöllinen esimerkiksi kameraa siirtäessä.
OnDestroy	Kutsutaan, jos skripti tuhotaan.
OnDisable	Kutsutaan, jos skripti disabloidaan.
OnEnable	Kutsutaan, jos skripti aktivoituu.
OnCollisionEnter/	Kutsutaan, kun peliobjektiin kiinnitetty rigidbody/collider osuu toi-

OnTriggerEnter	seen rigidbodyyn/collideriin.
OnCollisionStay/ OnTriggerStay	Kutsutaan joka fysiikoiden päivityksellä kun peliobjektiin kiinnitetty rigidbody/collider on kosketuksissa rigidbodyyn/collideriin.
OnCollisionExit/ OnTriggerExit	Kutsutaan, kun peliobjektiin kiinnitetty rigidbody/collider lopettaa kosketuksen toiseen rigidbodyyn/collideriin.
OnApplicationQuit	Kutsutaan jokaisessa skriptissä ennen kuin peli sammuttaa itsensä.
OnApplicationFocus	Kutsutaan, jos pelaajan fokus katoaa tai tulee takaisin peliin.

MonoBehaviour koodin ajaminen suoritetaan seuraavassa järjestyksessä:

- Kaikki Awake-kutsut
- Kaikki Start-kutsut
- Fysiikoiden päivitys aina, kun lähestytään asetettua fysiikoiden päivitysaikaa
    - Kaikki FixedUpdate-kutsut
    - Fysiikoiden päivitys
    - Laukaisu-eventit
    - Törmäys-eventit
- Rigidbodien paikan ja rotaation päivitys
- Syötetapahtumien rekisteröiminen
- Kaikki Update-metodit
- Animaatioiden päivitys
- Kaikki LateUpdate-metodit
- Renderointi

Skriptien muuttujien initialisointi pitää tehdä Awake-metodin sisässä, sillä se kutsutaan aina ennen Start-metodeja. Mahdolliset viittaukset muihin komponentteihin tehdään tämän vuoksi Start-metodissa, jolloin jokainen skripti on initialisoinut itsensä valmiiksi.

Update–metodi kutsutaan kerran jokaisen frame aikana, jonka vuoksi siellä pyörii suurin osa pelin käyttäytymiseen liittyvästä logiikasta, kuten pelihahmon liikuttaminen. Seuraavassa on esimerkki, kuinka GUIText komponenttiin päivitetään kulunut aika pelin alusta kerran framessa.

```
public class PaivitaKelloa : MonoBehaviour {
	public GUIText KellonAikaPelinAlusta;

	void Update( ) {
		KellonAikaPelinAlusta.text = Time.realtimeSinceStartup.ToString();
	}
}
```

FixedUpdate-metodia Unity yrittää kutsua kiinteän aikavälin mukaan. Fysiikkamoottorin suorittamat päivitykset ja laskutoimitukset tapahtuvat heti FixedUpdaten suorittamisen jälkeen. On suositeltavaa tehdä kaikki fysiikoihin liittyvä ohjelmointi tämän metodin sisällä. Seuraavassa esimerkissä nähdään kuinka rigidbody komponenttiin lisätään eteenpäin kohdistuvaa voimaa:

```
public class LiikutaRigidBodya : MonoBehaviour {
	void FixedUpdate( )  {
		rigidbody.AddForce(Vector3.forward);
	}
}
```

LateUpdate–metodi kutsutaan myös kerran framen aikana, mutta vasta kun kaikki Update-metodit on suoritettu. LateUpdate on hyvä metodi sellaiselle logiikalle, joka on riippuvainen muista komponenteista, joiden tiedot saattavat muuttua Update-metodin aikana. Tämä sen vuoksi, koska tieto voi olla vanhentunut framen aikana, sillä suoritettavien skriptien suorittamisjärjestystä ei ole ennalta määritelty. Seuraavassa esimerkissä komponentti seuraa kohdetta, minkä vuoksi toiminnon pitää tapahtua LateUpdate metodissa, koska pelaajan paikka saattaa muuttua Update metodin aikana.

```
public class SeuraaPelaajaa : MonoBehaviour {
	public Transform Kohde;
	void LateUpdate() {
		transform.position = Kohde.position;
	}
}
```

Unityn skriptit toimivat peliobjektien komponentteina, jonka vuoksi ne voivat myös keskustella keskenään ja Unity on luonut monia tapoja toteuttaa sen. Voit etsiä eri peliobjekteja tai niiden komponentteja tyypin, nimen, tagin ja tason mukaan.

Skriptissä voi olla suora viittaus johonkin toiseen komponenttiin, joka voidaan asettaa editorin avulla vetämällä se public-tyyppiseen muuttujaan, tai se voidaan etsiä eri tavoilla.

Seuraavassa on esitelty eri tapoja käsitellä muita komponentteja, kuten komponenttien haku peliobjektista tai niiden lisääminen siihen:

```
public class EsimerkkiKomponenttiHausta : MonoBehaviour {
	public Komponentti1 komponentti1;
	public Komponentti2[] komponentti2;
	public Camera paaKamera;

	private Rigidbody Rigidbody;
	private Transform LapsiTransform;

	void Start( ) {
		// Haetaan ensimmäinen löydetty Komponentti1 komponentti
		komponentti1 = GetComponent<Komponentti1>( );

		// Haetaan kaikki Komponentti2 tyyppiset komponentit
		komponentti2 = GetComponents<Komponentti2>( );

		// Etsitään GameObject, jonka tagiksi on asetettu Paakamera
		// ja haetaan siitä Camera-komponentti
		paaKamera =
			GameObject.FindWithTag("Paakamera").GetComponent<Camera>();

		// Lisätään GameObjectiin uusi Rigidbody
		Rigidbody = gameObject.AddComponent<Rigidbody>( );

		// Etsitään itsestämme objektia nimeltään LapsiTransform
		LapsiTransform = transform.Find("LapsiTransform");
	}
}
```

Yleinen tapa tehdä ajastettuja asioita Unityssä on Coroutinet. Niiden avulla voidaan ajastaa asioita tapahtumaan tietyn ajan päästä, framen lopussa ennen piirtämistä sekä niillä voidaan vaihtoehtoisesti jopa korvata Update metodin toiminta. Coroutinet luodaan samalla tavalla kuin normaalit metodit, mutta niiden tyypiksi asetetaan IEnumerator.

Seuraavassa esimerkki coroutinesta, joka muuttaa materiaalin alpha arvoa -0.1 kerran framessa:

```
IEnumerator Fade() {
	for (float f = 1f; f >= 0; f -= 0.1f) {
		Color color = renderer.material.color;
		color.a = f;
		renderer.material.color = color;
		yield return;
	}
}
```

5 TANK HOGS -PROTOTYYPIN TOTEUTUS

5.1 Mikä Tank Hogs on?

Tank Hogs on mobiilipeliprototyyppi, joka on erityisesti suunniteltu Apple iPad 4 ja sitä uudemmille Applen mobiililaitteille. Peli on reaaliaikainen verkkopeli, jossa kaksi pelaajaa taistelee toisiaan vastaan. Peli on toteutettu täysin 3D-tekniikalla. Pelaajan tehtävänä on ohjata omaa päätankkia strategisiin paikkoihin yrittäen tuhota vastapelaajan tankit sekä tämän tukikohdan. Sen lisäksi pelaajalla on muutama heikompi tekoälyn ohjaama tankki, jotka automaattisesti ajavat kohti vastustajan tukikohtaa. Taistelut ovat nopeita kahden minuutin tiiviitä taisteluita.

Pelin graafisessa ulkoasussa on tavoiteltu piirrosmaisuutta sekä pelin teemassa on vahva humoristinen sävy. Kuviossa 9 on Tank Hogs -pelin latausruutu, josta selkenee pelissä käytetty graafinen suuntaus.

Kuvio 9. Tank Hogs -pelin latausruutu

Taistelukenttä koostuu kahden pelaajan tukikohdista, eli farmeista. Näiden välistä löytyy taistelualue, jossa on erilaisia esteitä ja kerättäviä esineitä. Tavoitteena on tuhota vastapelaajan tankit, tukikohta sekä kerätä mahdollisimman paljon kolikoita taistelukentältä. Taistelun voittaja on vihollisensa tuhonnut tai ajan loppuessa enemmän pisteitä kerännyt pelaaja.

Ennen taistelun alkua esitellään taistelun osapuolet (ks. kuvio 10). Alun esittelyn jälkeen alkaa itse peli, jolloin pienemmät joukot lähtevät automaattisesti hyökkäämään vastustajan farmia kohti. Ensikosketus viholliseen tapahtuu taistelualueella ja viimeinen taistelu käydään jommankumman pelaajan tukikohdassa.

Kuvio 10. Tank Hogs -pelin joukkojen esittely intro

5.2 Työskentelymenetelmät

5.2.1 Suunnittelu

Peliä suunniteltaessa täytyi saada selville, kuinka hyvin Unity soveltuu pelien tekemiseen, kuinka pelit pitää suunnitella mobiiliympäristöön, sekä mitä rajoitteita nykypäivän mobiililaitteet asettavat.

Pelin ideaa kehitettäessä tutkittiin useita suosittuja pelejä, kuinka ne toimivat ja miksi ne ovat suosittuja sekä kuinka ne ovat teknisesti toteutettu, kuten Clash of Clans, Hill Climb Racing, Dead Trigger ja Minigore. Pelejä tutkiessa huomaa välittömästi mobiililaitteiden asettamat rajoitteet, sillä pelit ovat graafisesti huomattavan huonon näköisiä verrattuna nykypäivän tietokone- tai konsolipeleihin. Peleissä oli rajoitettu ruudulla näkyvän tavaran määrää heikoimmilla laitteilla.

Prototyyppiä tehdessä käytettiin mahdollisimman paljon Unity Asset Storesta löytyviä ratkaisuja. Asset Storesta löytyy valtava määrä 3D-malleja, ääniä sekä tekstuureja, joita hyödynnettiin peliä tehdessä. Lopulliseen tuotteeseen Asset Storesta löytyneet assetit korvattiin omilla.

5.2.2 Työlaitteistot ja -välineet

Pelin kehitys tapahtui pääsääntöisesti etätyöskentelynä. Projektin alusta asti pidettiin muutaman kerran viikossa yhteyttä Skype-ohjelman avulla sekä projektin tiedostot pidettiin BitBucket-palvelun tarjoamassa Mercurial-pohjaisessa versionhallintajärjestelmässä.

Kehityksen aikana päätyökaluina toimi Applen Macbook Pro -kannettava tietokone, Sublime Text -tekstieditori sekä Unity Pro -versio 4.3. Ohjelmointikielenä käytettiin C#, koska se tarjosi paremman tuen olio-ohjelmointiin sekä se oli entuudestaan tuttu. Testilaitteena käytettiin iPad 4 sekä iPhone 4s, joiden käyttöjärjestelmänä oli iOS 7.0. Graafikon työkaluina toimivat Adobe Photoshop CS5 sekä Autodesk 3DS Max 2011.

Koska Unity tarjoaa ilmaiseksi Web Player -käännösten tekemisen, niitä käytettiin hyödyksi pelin kehityksessä. Näiden avulla saatiin helposti uusin versio testattavaksi ilman, että täytyi erikseen tehdä käännöstä mobiililaitteille.

5.2.3 Roolit pelin teossa

Pelien luominen on muuttunut alkuaikojen yhden hengen projekteista erilaisiksi ja niihin liittyy nykypäivänä todella paljon erilaisia rooleja. Pelin tekeminen vaatii ohjelmoijia, 3D-mallintajia, graafikoita, kenttäsuunnittelijoita, markkinoijia, tuottajia, audiosuunnittelijoita sekä lukuisia muita tahoja.

Ohjelmoijien tehtävänä on ohjelmoida pelin logiikka, luoda efektit, säätää pelin fysiikat halutunlaisiksi sekä ottaa käyttöön graafikoiden ja audiosuunnittelijoiden luomat assetit. Suuremmissa projekteissa on omat ohjelmoijat eri osa-alueilla.

Graafikot sekä 3D-mallintajat luovat pelin ulkoasun ja vaikuttavat vahvasti pelin tunnelmaan. He luovat käyttöliittymät, 3D-mallit, tekstuurit ja materiaalit eri objekteille. Yleensä 3D-mallin ja sen tekstuurit mallille tekee sama henkilö. Graafikoiden ja 3D-mallintajien lisäksi suuremmissa projekteissa on erilliset tasosuunnittelijat, jotka suunnittelevat ja toteuttavat pelin eri kenttiä. He lisäävät 3D-mallintajien tekemiä esineitä ja hahmoja kenttiin.

Peliprojektiin tarvitsee myös muita osaajia, kuten tuottajan, joka hallitsee projektin kulun, hoitaa ajanhallinnan, pitää budjetista kiinni ja pitää ns. ”narut käsissään”. Tämän lisäksi myös markkinointiin tarvitaan työvoimaa, sillä se on todella tärkeää pelialalla valtavan pelitarjonnan vuoksi. Huono markkinointi yleensä johtaa huonoihin myyntilukuihin ja peli ei menesty odotetulla tavalla, jonka vuoksi yritys saattaa joutua lopettamaan toimintansa.

Laaduntestauksella pyritään siihen, että peli toimii moitteettomasti. Pelin kaatumiset ja ohjelmavirheet pyritään havaitsemaan ja korjaamaan. Pienemmissä projekteissa ohjelmoijat tekevät ohjelmakoodilleen testit itse, mutta suuremmissa projekteissa on testaustyökalujen kehittäjiä, joiden vastuulla on luoda kunnon testausvälineet peliä varten.

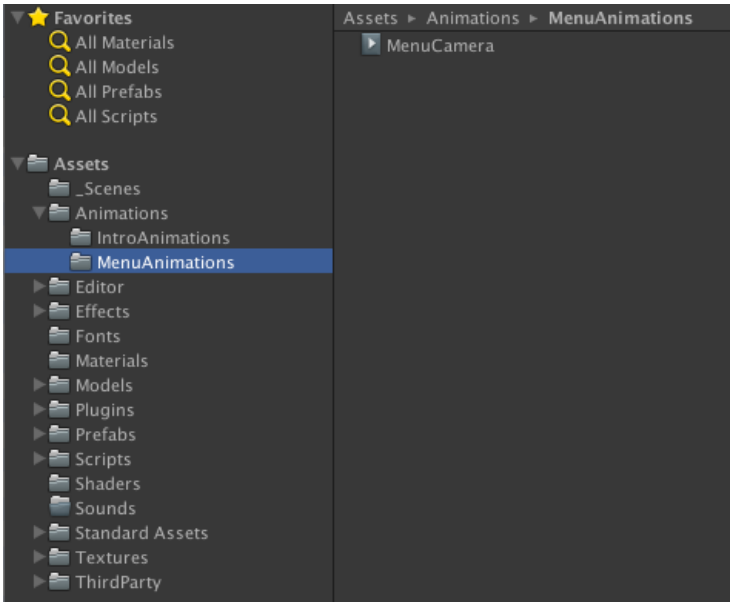
Koska Tuotantoyhtiö Taso Oy on uusi digitaalipelejä luova yritys, sen sisäiset roolijaot ovat häilyviä. Koska yritys oli pieni ja työmäärän ollessa suuri, jokainen joutui tekemään asioita oman osaamisalueen ulkopuolelta, eli jokainen henkilö osallistui peli-idean kehittämiseen ja suunnitteluun, markkinoinnin suunnitteluun, pelin testaamiseen ja projektin hallinnointiin.

5.3 Pelin rakenne

5.3.1 Kansiorakenne

Niin Unity-projektin kuin minkä tahansa peliprojektin kansiorakennetta kannattaa suunnitella hyvissä ajoin ennen toteutuksen aloittamista, sillä ilman suunnittelua projektirakenteesta tulee helposti yksi suuri sekamelska.

Tank Hogsin kansiorakenne suunniteltiin olemaan mahdollisimman selkeä myös sellaisille, jotka eivät ole olleet aikaisemmin mukana peliprojekteissa. Tämän vuoksi projektissa on jaettu kaikki 3D-mallit, tekstuurit, materiaalit, skriptit ja äänet tarkasti nimettyihin kansioihin. Tämä mahdollistaa helpon ja vaivattoman työskentelyn eri osapuolien välillä, mutta vaikeuttaa asioiden luontia dynaamisesti, koska assetin nimen sijaan tarvitaankin pitkä tiedostopolku. Kuviossa 11 on Tank Hogs -projektin kansiorakenteen juuritaso.



Kuvio 11. Tank Hogs -projektin kansiorakenne

Unityssä on muutama kansionimike varattu erikoistehtäviä varten. Nämä nimikkeet ovat Editor, Plugins, Resources, Gizmos, WebPlayerTemplates, StreamingAssets, Standard Assets ja Pro Standard Assets. Taulukossa 4 on avattu jokaisen erikoiskansion perusidea.

Taulukko 4. Erikoiskansioiden selitys (Special Folder Names in your Assets Folder 2014I.)

Kansio nimike	Tarkoitus
Editor	*Editor*-kansio mahdollistaa skripteissä pääsyn Unity Editor API:in. Jos skripti käyttää jotain luokkaa tai toiminnallisuutta UnityEditor luokasta, skriptin täytyy olla *Editor*-kansiossa.
Plugins	*Plugins*-kansioon pitää laittaa kaikki natiivilisäosat, joihin halutaan päästä käsiksi skripteistä. *Plugins*-kansio ei myöskään voi olla minkään kansion alakansio, vaan se pitää sijoittaa projektin juureen eli *Assets*-kansioon.
Resources	*Resources*-kansioon sijoitetaan kaikki assetit, johon halutaan päästä käsiksi nimen tai polun mukaan skripteistä. *Resources*-kansion kanssa pitää olla tarkka, sillä Unity sisällyttää kaikki assetit *Resources*-kansiosta, vaikka niitä ei käytettäisi ollenkaan.
Gizmos	*Gizmos*-kansioon sijoitetaan kaikki tekstuuri- ja ikoniassetit, joita halutaan käyttää Gizmos.DrawIcon() toiminnon avulla. Assetteja voidaan kutsua nimen mukaan kansiosta.
WebPlayerTemplates	*WebPlayerTemplates*-kansioon sijoitetaan uudet sivupohjat selainversiota varten.
StreamingAssets	*StreamingAssets*-kansioon sijoitetut tiedostot kopioidaan käännöksen yhteydessä sellaisenaan pelin juureen. Polku kansioon vaihtelee eri käännöksissä, mutta polku on saatavilla Application.streamingAssetsPath muuttujan kautta.
Standard Assets	*Standard Assets* -kansioon sijoitetut skriptit käännetään aina ensimmäisenä. Tämä on hyödyllinen ominaisuus, koska esimerkiksi se

	on yksi tapa avata pääsy C#-skripteistä toisen kielen skripteihin.
Pro Standard Assets	Pro Standard Assets -kansio toimii samalla periaatteella kuin *Standard Assets* -kansio, mutta sinne tulisi sijoittaa skriptit, jotka käyttävät maksullisen version ominaisuuksia.

5.3.2 Scenejen rakenne

Tank Hogs -peli jaettiin yhteensä kolmeen sceneen. Peli alkaa *InitScene*-scenestä, joka hoitaa pelin yhdistämisen verkkopalvelimeen, initialisoi pelin äänet, valaistuksen ja asetukset. Kun peli on saanut initialisoitua itsensä ja yhdistänyt palvelimelle, peli vaihtaa itsensä automaattisesti *Menu*-sceneen.

Menu-scene on Tank Hogsissa yksinkertainen. Taustalla näkyy taistelualue ja yksinkertainen animaatio pyörittää kameraa kentän ympärillä. Tämän lisäksi ruudussa on vain kaksi näppäintä, joista toisella yhdistetään peliin ja toisesta päästään asetuksiin. Menussa näkyy myös yrityksen logo sekä yksi pelin hahmoista.

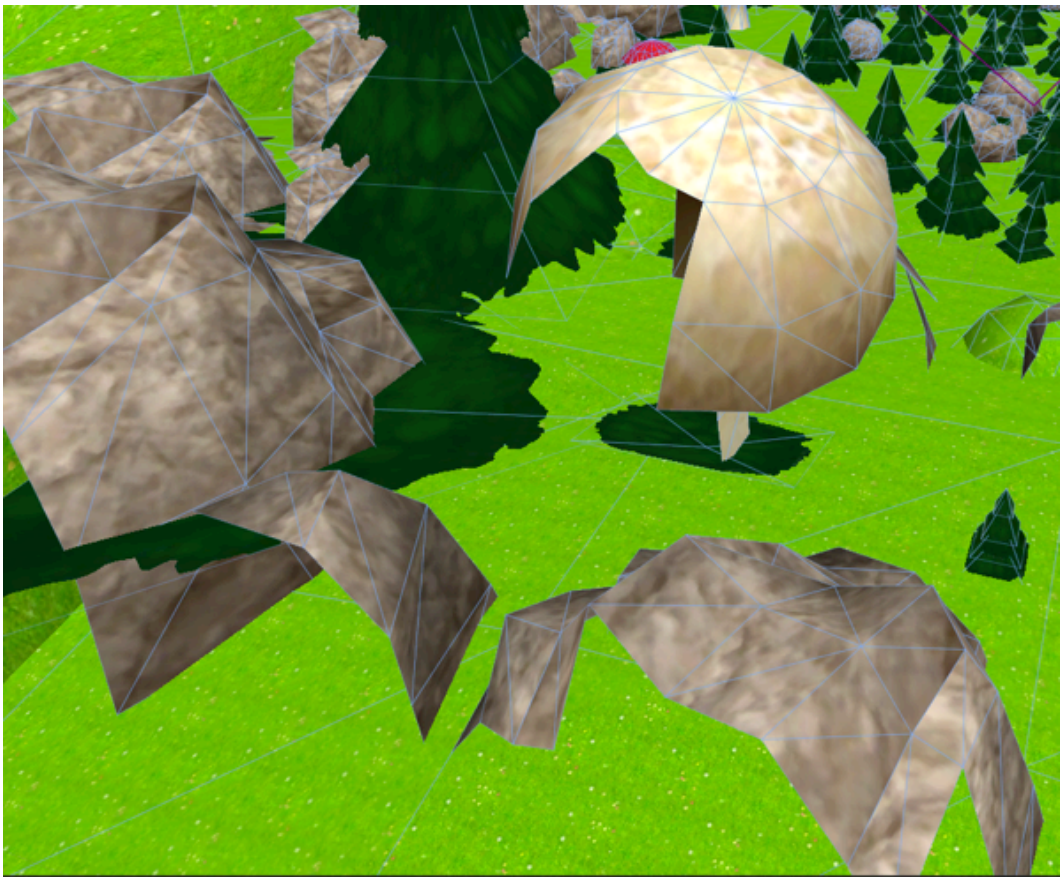
Toisen pelaajan löytämisen jälkeen peli lataa automaattisesti *Game*-scenen, jossa tapahtuu huomattavasti enemmän kuin muissa sceneissä. *Game*-scene toimii pelin taistelualustana. Scenen latauksen aikana pelikentän keskialue luodaan satunnaisesti muutamasta valmiiksi rakennetusta mallista, minkä avulla saadaan hieman vaihtelua. Kun molempien pelaajien joukot on luotu, alkaa esittelyanimaatio, jossa käydään läpi sekä omat että vastustajan joukot. Tämän jälkeen taistelu alkaa. Jos jommankumman pelaajan yhteys palvelimeen katkeaa, peli palaa automaattisesti *Menu*-sceneen.

5.4 Audiovisuaalinen toteutus

5.4.1 3D-mallit

Tank Hogs -pelin kaikki 3D-mallit loi yrityksen graafikko Autodeskin 3DS Max 2011 -ohjelmalla. Sallitut kokonaisverteksimäärät iOS-laiteilla ovat hyvin matalat, minkä vuoksi 3D-mallit suunniteltiin alusta alkaen sisältämään mahdollisimman vähän verteksejä. Pitääkin muistaa, että mallinnusohjelmistojen näyttämät verteksimäärät eivät vastaa

Unityyn vietyjen mallien verteksimäärää, koska verteksit sisältävät myös muuta informaatiota kuin oman sijaintinsa, kuten UV koordinaatit, normaalit, tangetit ja materiaalien ID:n. Kuviossa 12 nähdään, kuinka Tank Hogsissa käytettyjen 3D-mallien verteksimääriä saatiin vähennettyä poistamalla turhaa geometriaa.



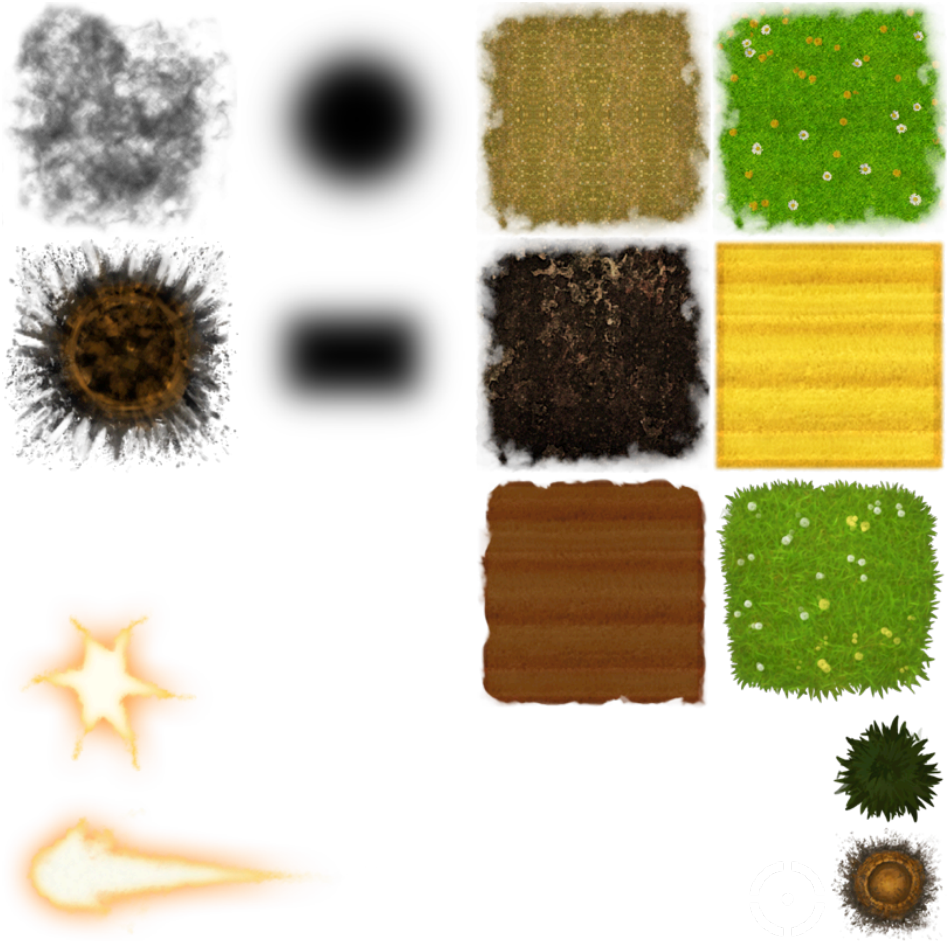
Kuvio 12. Esimerkki optimoiduista 3D-malleista, joista on poistettu turha geometria

5.4.2 Tekstuurit ja materiaalit

Tekstuurit ja materiaalit määrittävät millaiselta 3D-mallit näyttävät. Tekstuurit ovat käytännössä kuvia, joita 3D-mallien päälle ikään kuin pingotetaan. Materiaalit määrittelevät näyttääkö tekstuuri esimerkiksi kiiltävältä vai mattapintaiselta sekä käytetäänkö tekstuurin läpinäkyvyyttä.

Jokainen materiaali luo yhden draw callsin eli piirtokomennon, jonka vuoksi Tank Hogsin tekstuurit koostettiin muutamaan tekstuuriatlakseen, joka mahdollisti vain muutaman eri materiaalin käytön. 3D-malleille pitää luoda erikseen UV-kartoitus, että ne löytävät

oikean tekstuurin atlaksesta. Tekstuureja varten koostettiin kolme eri atlasta, joista luotiin kolme eri materiaalia. Materiaaleilla oli kolme pääkohdetta, joista ensimmäinen oli läpinäkyvyyttä sisältäville, toinen oli ns. cutout-materiaali, ja kolmas oli normaaleille tekstuureille. Kuviossa 13 nähdään läpinäkyvyyttä sisältävä tekstuuriatlas.



Kuvio 13. Läpinäkyvyyttä sisältävä tekstuuriatlas

5.4.3 Käyttöliittymän suunnittelu

Pelin käyttöliittymä on suunniteltu toimimaan erityisesti taulutietokoneilla. Päävalikon käyttöliittymä pidettiin yksinkertaisena, eikä sen suunnitteluun ja toteutukseen käytetty paljoa resursseja. *BattleScenen* käyttöliittymään sen sijaan käytettiin huomattava määrä resursseja niin suunniteluun kuin toteuttamiseen. Se sisältää kaiken tarvittavan tiedon mitä pelaaja tarvitsee pelatessaan, kuten taistelun jäljellä olevan ajan, elämät, painikkeet eri toimintojen päälle kytkemistä varten sekä kosmeettisia asioita.

Taistelun käyttöliittymä koostuu neljästä osiosta, jotka ovat *Intro*-, *Battle*-, *Score*- ja *SettingsGUI*. Näistä osioista vain yksi kerrallaan on käytössä. Taisteluun siirtyessä peli alkaa introsta, jolloin *IntroGUI* on käytössä, joka on yksinkertainen, sillä siinä on vain muutama komponentti. *IntroGUI* näyttää pelin alussa ajastimen ennen kuin taistelu alkaa. Ajastimen jälkeen taistelu alkaa ja *BattleGUI* aktivoidaan.

BattleGUIssa näkyy kaikki olennainen tieto. Kuviossa 14 näkyy *BattleGUIn* käyttöliittymä, joka on toteutettu yrityksen saatujen assettien ja suunnitelmien mukaan.

Kuvio 14. Tank Hogs -pelin taistelukäyttöliittymä

BattleGUIn yläpalkissa näkyy tiedot, kuinka monta tankkia vastustajalla on jäljellä ja montako elämää hänen päähahmollaan on. Yläpalkissa on myös ajastin, jossa näkyy paljon aikaa on kulunut ja paljon sitä on jäljellä sekä mittari, jossa näkyy ansaittujen tähtien määrä.

BattleGUIn oikeassa alareunassa on taistelussa käytettävät napit. Nappeja on kolme, joista yksi aktivoi päähahmon erikoisliikkeen, yksi käskee pienempien joukkojen seurata pelaajaa ja viimeinen näppäin aktivoi pelaajan päähahmolle nopeamman liikkumisen ja

ampumisen lyhyeksi ajaksi. Erikoisliikepainike ja tehon parannuspainike ovat deaktivoituina, jos ne eivät ole käytettävissä.

BattleGUIn vasemmassa alareunassa sijaitsee kaksi painiketta, joista toisella voidaan fokusoida kamera takaisin ohjattavaan päähahmoon ja toisella voidaan kysyä vastapelaajalta, haluaisiko tämä aloittaa pelin alusta.

5.4.4 Käyttöliittymän toteutus

Käyttöliittymä toteutettiin maksullisella NGUI-lisäosalla, koska Unity ei tarjonnut ratkaisua kunnon käyttöliittymien tekemiseen sen tekohetkellä. NGUI sisältää huomattavan määrän toimintoja käyttöliittymien tekoon. NGUI on nopea ja kuluttaa vähän laitteen resursseja. Sen avulla on helppo luoda responsiivisia käyttöliittymiä, jotka toimivat kaikenkokoisilla resoluutioilla.

NGUIn käyttöönotto tapahtuu purkamalla ostettu NGUI-paketti Unity-projektissa. Purkamisen jälkeen Unityn ylävalikkoon ilmestyy NGUI, josta löytyy kaikki NGUIn työkalut. Valikosta löytyy Create 2D UI -painike, joka luo scenen hierarkiaan UI Root -nimisen peliobjektin. Tämä peliobjekti sisältää NGUIn tarvitsemat komponentit eli UIRoot- ja UICamera-komponentit. UIRoot-komponentin avulla määritellään, minkä kokoiselle resoluutiolle käyttöliittymää lähdetään tekemään, ja UICamera-komponentti hoitaa syötteiden käsittelyn.

NGUI käyttää niin sanottua tekstuuriatlasta toimiakseen nopeammin. Tämä tarkoittaa sitä, että kaikki käyttöliittymään luodut tekstuurit yhdistetään yhdeksi isoksi tekstuuriksi NGUIn tarjoamalla työkalulla. Työkalu luo tekstuurin valmiiksi ja laskee automaattisesti tiukimman mahdollisen tuloksen, että saatu atlas olisi mahdollisimman pieni. Atlakseen voidaan lisätä myös fontteja (ks. kuvio 15), jolloin peliin voidaan kirjoittaa tekstiä hyvin kevyesti.

Kuvio 15. NGUIn mukana tuleva esimerkkiatlas

5.4.5 Efektit

Pelissä käytettiin efektejä pääasiassa räjähdyksiä varten. Pelin efektit luotiin täysin 2D:nä, jotka aseteltiin samaan kulmaan kuin kamera. 2D-efektit luotiin käymällä eräänlaisen tekstuuriatlaksen, eli tässä tapauksessa spritesheetin, kohtia läpi halutulla nopeudella. Tämä tekniikka mahdollisti useamman efektin näyttämisen yhtäaikaisesti ilman, että pelin suorityskyky kärsii.

Seuraavassa esimerkissä asetellaan objekti katsomaan kohti kameraa:

```
Vector3 kohdeVektori = paaKamera.transform.position - transform.position;
transform.LookAt(kohdeVektori);
```

5.4.6 Äänet

Pelin äänimaailma luotiin yrityksen omien assettien avulla käyttäen MasterAudio-lisäosaa. MasterAudio on valmis ratkaisu, joka tarjoaa kevyen ratkaisun unohtamatta erikoisempiakaan toimintoja. MasterAudiolla voidaan luoda useita soittolistoja, jotka

voivat sisältää useita eri kappaleita. Soittolistat sisältävät valmiit toiminnot kappaleiden sekoittamiseen ja vaihtamiseen, sävelkorkeuden muokkaamiseen sekä useita muita ominaisuuksia.

MasterAudiossa ääniefektit toimivat ääniefektiryhmänä, jotka luodaan tietyllä nimellä ja ne voidaan lisätä johonkin haluttuun kokoelmaan. Eri kokoelmia voidaan hiljentää tai vahvistaa kokoelman nimen mukaan. Ääniefekteille luotu nimi mahdollistaa sen, että yhden ääniefektin alle voidaan asettaa useita eri versioita, joita voidaan painottaa haluamansa mukaan. Painottaminen tarkoittaa sitä, että eri ääniefekteille voidaan asettaa eri määrä instansseja, joista sitten satunnaisesti haetaan yksi. Esimerkiksi Tank Hogsin räjähdysefektille on olemassa kolme ääniefektiä, joista kahdesta on luotu viisi instanssia ja yhdestä 10 instanssia. Tämän avulla varmistettiin, että räjähdysäänet ovat useasti saman kuuloisia ja silloin tällöin soi hieman erilainen versio.

Seuraavissa esimerkeissä esitellään kuinka MasterAudion avulla soitetaan ääniefektejä:

```
// Esimerkki 1: Ääniefekti ammusten tuhoutumisessa
if (ammusOsuiJohonkin) {
    MasterAudio.PlaySound3D("OsumaRajahdysEfekti", transform);
}
else {
    MasterAudio.PlaySound3D("HarhalaukausRajahdysEfekti", transform);
}

// Esimerkki 2: Ääniefekti pelin alkamiseen
MasterAudio.PlaySound("PeliAlkaa");
```

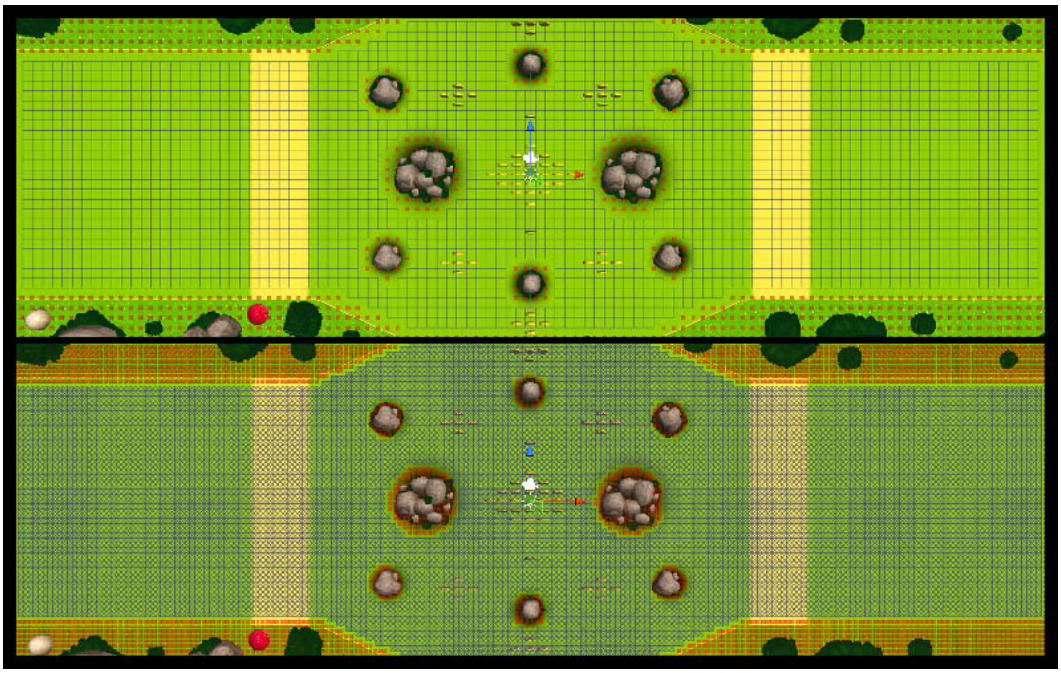
MasterAudion avulla voidaan soittaa efektejä niin 2D- kuin 3D-muodossa. Tämä käytännössä tarkoittaa sitä, että jos räjähtää kauempana, se voidaan soittaa 3D-muodossa, jolloin se soi hiljempaa kuin lähellä räjähtänyt olisi soinut.

5.5 Reittihaku

Reittihaun avulla voidaan laskea reittejä kahden tai useamman objektin välille. Reittihaku ottaa huomioon staattiset objektit, jotka ovat pisteiden välillä, mutta kehittyneemmät ratkaisut ottavat myös huomioon muut liikkuvat objektit.

Tank Hogsin reittihakuun koitettiin monia eri ratkaisuja ennen kuin päädyttiin nykyiseen. Ensimmäinen reittihaku toteutettiin Unity-pelimoottorin sisäänrakennetulla työkalulla.

Reitinhakua tehtäessä ei ollut epäselvää, että reitinhakua pitää optimoida todella paljon. Tärkeimpänä ja suurimpana suorituskyvyn kannalta oli tiputtaa navigointiverkon tarkkuutta. Kuviossa 16 nähdään navigointiverkon erot eri tarkkuuksilla.



Kuvio 16. Navigointiverkon erot optimoinneilla: Ylempi epätarkka ja alempi tarkka

Navigointiverkon tarkkuuden heikentämisen lisäksi reitinhakua käyttävien tankkien toimintaa muutettiin niin, että tankit päivittävät reittinsä muutaman framen välein, eikä jokaisella framella. Tämä oli mahdollista toteuttaa helposti käyttämällä coroutineja, kuten seuraavassa esimerkissä nähdään:

```
// Alkuperäinen toteutus: Lasketaan uusi reitti jokaisella framella
void Update() {
	LaskeUusiReitti(transform.position, kohteenPosition);
}

// Parempi totetutus:
// Käynnistetään coroutine alussa, joka laskee reitin 0.5s välein
void Start() {
	StartCoroutine(“ReitinPaivitysCoroutine”);
}

IEnumerator ReitinPaivitysCoroutine() {
	WaitForSeconds laskemisVali = new WaitForSeconds(0.5f);
	while(true) {
		// Laske uusi reitti
		LaskeUusiReitti(transform.position, kohteenPosition);
		// Nuku 0.5 sekuntia ennen seuraavaa laskemista
		yield return laskemisVali;
```

```
        }
}
```

Navigointiverkkoa pitää pystyä jotenkin käyttämäänkin. Tähän käyttämämme lisäosa tarjoaa valmiita ratkaisuja. Pelissä käytimme Seeker-komponenttia, jolla voidaan laskea reitti kahden pisteen välille.

Seuraavassa esimerkki Seeker-komponentin käytöstä reitin laskemiseen:

```
private Path nykyinenReitti;
public void LaskeUusiReitti(Vector3 aloitusPiste, Vector3 kohdePiste) {
        nykyinenReitti = GetComponent<Seeker>.StartPath(aloitusPiste, kohdePiste);
}
```

Seekerin palauttama Path-olio pitää sisällään vectorPath-nimisen listan pisteistä, joita seuraamalla päästään aloituspisteestä kohteeseen.

Seuraavassa esimerkissä käydään läpi yksinkertainen liikkuminen reitin mukaan:

```
private Path nykyinenReitti;
private int nykyinenPiste; // nollataan reitinlaskemisen yhteydessa

public void Update () {
        // Jos ei ole laskettua reittia, lopetetaan funktion lapikayminen
        if (nykyinenReitti == null) return;

        // Jos nykyisen pisteen indeksi on suurempi kuin reitin sisaltama,
        // niin lopetaan funktion lapikayminen
        if (nykyinenPiste > nykyinenReitti.vectorPath.Count) return;

        if (nykyinenPiste == nykyinenReitti.vectorPath.Count) {
                Debug.Log ("Ollaan reitin lopussa");
                nykyinenPiste++;
                nykyinenReitti = null;
                return;
        }
        Vector3 nykyinenKohde = nykyinenReitti.vectorPath[nykyinenPiste];
        // Suunta seuraavalle pisteelle
        Vector3 suunta = (nykyinenKohde - transform.position).normalized;
        suunta *= Time.deltaTime;
        transform.Translate (suunta);

        if (Vector3.Distance (transform.position, nykyinenKohde)
                < matkaSeuraavallePisteelle) {
                nykyinenPiste++;
        }
}
```

5.6 Pelihahmon toiminta

Pelaaja ohjaa pelkästään päätankkia, mutta voi käyttää erikoisliikettä, käskeä apurit seuraamaan päätankkia sekä aktivoida väliaikainen päätankin tehostaminen. Pelaajan teh-

tävänä on ohjata tankki vastustajan joukkojen luo ja tekoäly automaattisesti tähtää ja ampuu muutaman sekunnin välein lähintä kohdetta kohti.

Päätankin ohjaaminen tapahtuu koskettamalla ruutua lyhyesti kohdasta, johon haluaa hahmon menevän. Kosketuskohta muunnetaan pisteeksi pelimaailmassa, jonka jälkeen reittihaun käyttämälle Seeker-komponentille annetaan nykyinen paikka sekä kosketuskohta maailmassa. Seeker-komponentin palauttaman Path-olion sisältämän listan avulla tankki liikutetaan pisteen luo.

Apurien ohjaus toimii samalla ajatuksella, mutta pelaaja ei käske niiden liikkua, vaan apurit automaattisesti hyökkäävät kohti vihollisen tukikohtaa. Jos apureiden reitille tulee vihollisia, ne pysähtyvät taistelemaan. Jos kohde katoaa ampumisetäisyydeltä, apurit lähtevät hyökkäykseen uudelleen.

Pelaaja voi aktivoida erikoisliikkeen, joka prototyypissä tarkoittaa hurrikaaniliikkeen aktivoimista. Erikoisliike kestää kolme sekuntia ja sinä aikana päätankin tykkiosio pyörii lujaa ampuen samalla kuulia hurrikaanimaisesti ympärilleen. Erikoisliikkeen voi käyttää kolme kertaa taistelun aikana.

Tämän lisäksi pelaaja voi aktivoida pienen tehostuksen lyhyeksi aikaa. Tehostuksen aikana päähahmo liikkuu sekä ampuu nopeammin ja tehokkaammin. Tehostuksen käyttäminen kuluttaa energiaa, jota kertyy pikkuhiljaa pelin edetessä sekä sitä voidaan kerätä kentältä löytyvistä energiakolikoista.

5.7 Verkkoratkaisu

5.7.1 Suunnittelu

Tank Hogsin verkkototeutusta ei suunniteltu aluksi kunnolla, koska ei ollut tarvittavaa tietotaitoa. Tämän vuoksi kokemusta kerättiin Unityn sisäänrakennetun networking-paketin avulla siihen asti, että perusymmärrys verkko-ohjelmointiin oli saavutettu.

Tämän jälkeen lähdettiin suunnittelemaan pysyvämpää ratkaisua. Verkkoratkaisusta haluttiin mahdollisimman yksinkertainen, koska projektiin varattua aikaa ei haluttu käyttää pelkästään yhden osa-alueen hiomiseen ja ylläpitämiseen. Suunnittelun jälkeen pää-

ajatuksena oli saada palvelimen ylläpito pois yrityksen harteilta ja siirtää se pilveen. Tähän ExitGamesin tarjoama Photon Cloud -palvelu (PUN) sopi täydellisesti.

5.7.2 Photon Cloudin toimintaperiaate

Photon Cloud eli PUN on ExitGamesin tarjoama pilvipalvelu verkkopelien tekemiseen. Unity käyttää omassa palvelussaan niin sanottua server-client-pohjaa, jossa yksi pelaajista toimii serverinä. Myös PUN käyttää server-client-pohjaa, mutta heidän ratkaisussaan kukaan pelaajista ei toimi serverinä, vaan ExitGames tarjoaa oman kohdistetun palvelimen. Palvelimia on tarjolla ympäri maailmaa.

PUN toimii lähes samalla tavalla, kuin Unityn sisäänrakennettu networking-ratkaisu. Sen API on samanlainen ja PUN tarjoaakin työkalun, jolla voidaan konvertoida Unity-projekti, joka käyttää Unityn omaa networking-ratkaisua käyttämään PUNin ratkaisua.

Ottaakseen käyttöön PUNin, kehittäjä joutuu luomaan tilin ExitGamesin järjestelmään, jossa hän luo pelin. Pelille asetetaan järjestelmään nimi, jonka jälkeen järjestelmä generoi uniikin Application ID:n, joka on muotoa 06000e5e-xxxx-46ea-xxxx-bb69xb000b. Kyseinen koodi asetetaan PUNin asetuksiin Unity-editorissa. Asetuksissa myös valitaan mihin pilveen yhdistetään.

5.7.3 Toteutus

Jokaiselle peliobjektille, joka haluttiin synkronoida laitteiden välillä, piti asettaa PhotonView-komponentti. Kyseinen komponentti yksilöi objektit sekä se mahdollistaa tiedon lähettämisen toiselle laitteelle kyseisestä objektista. PhotonView-komponentista löytyy isMine-ominaisuus, jonka avulla nähdään, kenelle objekti kuuluu.

Objekteja voi instansioida monella eri tapaa näkymään kaikilla clienteillä. Kaikki peliobjektit, jotka eivät tarvinneet myöhemmin selitettyä object pool -järjestelmää, luotiin Photonista löytyvät PhotonNetwork.Instantiate(”Prefab”)- tai PhotonNetwork.InstantiateSceneObject(”Prefab”) -metodin avulla. Toinen tapa instansioida peliobjekteja on käyttää Unityn omaa Instantiate-metodia. Tämä vaatii sen, että instansoitaviin peliobjekteihin pitää erikseen käsin luoda tiedot photonView-komponenttiin.

Peliobjektien automaattinen synkronointi ei tapahdu automaattisesti käyttämällä PhotonView-komponenttia, vaan sille pitää asettaa jokin skripti tai transformi, jonka se synkronoi. PhotonView-komponenttia voi käyttää myös pelkästään lähettämään RPC-käskyjä.

Seuraavassa esimerkki tankkien tykkiosion synkronoimisesta:

```
public class InterpolatedRotation : Photon.MonoBehaviour
{
    void Awake() {
        // Disabloidaan skripti jos emme omista peliobjektia
        if (photonView.isMine) enabled = false;
    }

    void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info) {
        // Jos olemme lahettava osapuoli
        if (stream.isWriting) {
            Quaternion rotaatio = transform.localRotation;
            stream.Serialize(ref rotaatio);
        } // Jos vastaanotetaan tietoa
        else {
            Quaternion rotaatio = Quaternion.identity;
            stream.Serialize(ref rotaatio);

            transform.localRotation = rotaatio;
        }
    }
}
```

5.8 Optimointi

5.8.1 Skriptien optimointi

Unityllä ohjelmoitaessa voidaan tehdä yksinkertaisia, mutta tehokkaita optimointeja. Seuraavassa listattuna lyhyesti muutama tärkeä optimointi, mitä käytettiin Tank Hogsia luodessa (Performance Optimization 2014.):

1. Ei käytetä dynaamisia tyyppejä

```
// Huono tapa
Component ammus = GetComponent<Ammus>();
ammus.Rajahda();

// Hyvä tapa
Ammus ammus = GetComponent<Ammus>();
ammus.Rajahda();
```

2. Tallennetaan haetut komponentit muistiin eikä haeta joka framella

```
public class HuonoTapa : MonoBehaviour {
    void Update() {
```

```
            GetComponent<Pathfinder>().LaskeReitti(new Vector3(10, 10, 10));
        }
    }

    public class OikeaTapa : MonoBehaviour {
        private Pathfinder reittihaku;
        private Vector3 suunta;
        void Awake() {
            reittihaku = GetComponent<Pathfinder>();
            suunta = new Vector3(10, 10, 10)
        }
        void Update() {
            reittihaku.LaskeReitti(suunta);
        }
    }
```

3. Ei päivitetä kaikkea jokaisella framella, vaan käytetään coroutineja

```
WaitForSeconds etsimisVali;
IEnumerator EtsiVihollista(float aika) {
    etsimisVali = new WaitForSeconds(aika);
    while(true) {
        // Etsi vihollinen
        ...
        // Nuku etsimisvälin verran ennen kuin etsit uudelleen
        yield return etsimisVali;
    }
}
```

5.8.2 Object Pool

Object Pool, eli niin sanottu objektien tai esineiden allastus, toimii todella tärkeässä osassa optimointia. Se tarkoittaa sitä, että objektien luomisen ja tuhoamisen sijaan niitä kierrätetään, eli pelin alussa luodaan valmiiksi tarvittava määrä objekteja, joita aktivoidaan ja deaktivoidaan käyttötarpeen mukaan. Seuraavaksi on esitelty hyvin yksinkertainen toteutus Object Poolingista:

```
using UnityEngine;
using System.Collections.Generic;

public class ObjectienKierratys {
	private Stack<GameObject> saatavillaOlevatObjektit;
	private GameObject alkuperainenPrefab;

	// Luokan constructor
	public ObjectienKierratys( GameObject prefab, int altaanHaluttuKoko )
	{
		alkuperainenPrefab = prefab;
		saatavillaOlevatObjektit = new Stack<GameObject>(altaanHaluttuKoko);

		LuoObjekteja(altaanHaluttuKoko);
	}

	private void LuoObjekteja( int count ) {
```

```
                for ( int i = 0; i < count; i++ ) {
                        GameObject peliObjekti =
                        Object.Instantiate(alkuperainenPrefab) as GameObject;
                        peliObjekti.SetActive(false);

                        saatavillaOlevatObjektit.Push(peliObjekti);
                }
        }

        // Haetaan altaasta (pinosta) yksi objekti.
        // Jos allas on tyhjä, luodaan yksi objekti lisää
        public GameObject HaeObjektiAltaasta( )  {
                if ( saatavillaOlevatObjektit.Count == 0 ) {
                        LuoObjekteja(1);
                        return HaeObjektiAltaasta( );
                }
                else {
                        return saatavillaOlevatObjektit.Pop( );
                }
        }

        // Palautetaan objekti altaaseen.
        // Disabloidaan objekti ja työnnettään se takaisin pinoon
        public void AsetaObjektiTakaisinAltaaseen( GameObject peliObjekti ) {
                peliObjekti.SetActive(false);
                saatavillaOlevatObjektit.Push(peliObjekti);
        }
}
```

5.8.3 Dynamic batching

Dynamic batching eli dynaaminen piirtokomentojen (Draw call) yhdistäminen tarkoittaa sitä, että dynaamisten peliobjektien piirtäminen yhdistetään yhden piirtokomennon alle. Yhdistääkseen objektit niiden täytyy täyttää tarvitut kriteerit. Yhdistettävien objektien kuuluu käyttää samaa materiaalia ja verteksiattribuuttien määrä pitää olla alle 900. Tekstuuriatlasoinnin avulla saadaan eri objektit käyttämään samaa materiaalia, mutta verteksiattribuuttien pitäminen alhaisena vaatii suurempia ratkaisuja.

Tank Hogsia tehdessä suurin ongelma oli saada tankkien verteksiattribuutit alhaiseksi. Tankeista poistettiin hyvin paljon geometriaa, kuten tankkien pohjat, tykin pohjat sekä ylipäänsä malleja jouduttiin yksinkertaistamaan. Tämän lisäksi tankit koostettiin useammasta 3D-verkosta, jotka koostettiin yhteen isompaan.

Toteutetussa pelissä kaikki tankit, kerättävät kolikot ja rakennukset menevät keskenään muutamaan piirtokomentoon. Kuviossa 17 näkyy testi scene, jossa piirtokomentojen määrää testattiin.

Kuvio 17. Piirtokomentojen testaamiseen luotu testi scene

5.8.4 Static batching

Static batching eli staattinen piirtokomentojen yhdistäminen on Unityn tarjoama toinen tapa, jolla pystyy vähentämään piirtokomentojen määrää. Se toimii lähes samalla tavalla kuin dynamic batching, mutta sillä erolla, että staattisesti yhdistetyt objektit eivät saa liikkua scenessä ja niissä ei ole rajoitusta verteksimäärille. Jotta objekti voi staattisesti yhdistyä, se pitää erikseen merkitä Unity-editorissa staattiseksi ja sen pitää jaksaa sama materiaali muiden kanssa.

Tank Hogsia tehdessä static batchingia käytettiin taustaelementteihin sekä taistelualueen rakennuksiin ja kiviin.

5.8.5 Tekstuurien pakkaus

Mobiililaitteille tehdessä tekstuureja on käytännössä pakko pakata vähäisen muistin takia. Pakkaamattomat tekstuurit kuluttavat paljon muistia, joka voi johtaa siihen, että peli tai jopa itse laite voi kaatua pelin ollessa käynnissä. iOS-laitteet käyttävät tekstuuriformaattinaan PVRTC-formaattia. Kyseinen formaatti mahdollistaa tekstuurin pakkauksen joko kahdella tai neljällä bitillä per pikseli. Esimerkiksi käyttöliittymän tekstuuriatlaksen koko pakkaamattomana on 64 MB ja PVRTC-formaattiin pakattuna 8 MB.

5.9 Pelitestaus

5.9.1 Yleistä pelitestauksesta

Testaus on tärkeä osa niin ohjelmistojen kuin pelien kehittämistä. Se on erityisen tärkeää varsinkin mobiililaitteille tehtäessä, jossa myös laitteiden suorituskyky on hyvin rajoitettua. Peliprojektia tehdessä testaamista on monenlaista, mutta suurin osa kohdistuu ohjelmakoodin testaamiseen sekä peli-idean testaamiseen käyttäjillä.

Ohjelmakoodia voidaan ja kuuluu testata projektin missä tahansa kehitysvaiheessa. Pelejä tehdessä suurin osa testaamisesta kehitysvaiheessa tapahtuu pelimoottorin puolella sekä vahvasti suorituskykyyn vaikuttavissa komponenteissa. Varsinkin pienemmässä yrityksessä kannattaa testaaminen aloittaa muissa komponenteissa vasta kun peli-idea ja spesifikaatio on lyöty lukkoon.

Peli-idean testaaminen kannattaa aloittaa jo ennen varsinaista pelin kehittämistä. Peli-idean testaaminen on yksi tärkeimmistä vaiheista pelikehityksessä, sillä ei riitä vaikka peli olisi kuinka hyvin tehty, jos peli-idea ei ole viimeistelty. Peli-ideaa pitää testata potentiaalisilla käyttäjillä, koska peli-ideaan sokaistuu itse helposti, jolloin ei huomaa puutteita. Jos joku käyttäjistä antaa negatiivista palautetta, siihen on usein syy, eikä se ole useinkaan testaajassa.

5.9.2 Tank Hogsin testaus

Tank Hogsin testaus kohdistui pääasiassa peli-idean testaamiseen. Pelin ohjelmakoodiakin testattiin, mutta se keskittyi pääasiassa reittihaun ja verkkoratkaisuiden ongelmiin ja pullonkaulojen etsintään. Muiden komponenttien testaaminen oli vähäistä, sillä ne olivat yksinkertaisia ja niiden tuli toimia, että peli toimisi oikein.

Kun Tank Hogsia lähdettiin kehittämään, sen peli-idea ei ollut lähellekään valmis. Peli-ideaa ja mekaniikkoja pohdittiin paljon, koska sen avulla päästiin suurimmista sudenkuopista heti eroon. Tämä ei kuitenkaan riitä, vaan peli-ideaa piti testauttaa myös muilla osa-puolilla, jolloin saatiin näkemyksiä kehittäjätiimin ulkopuolelta. Tämän avulla estet-

tiin sokaistuminen omaan projektiin, jolloin saatiin peli-ideasta huomattavasti kutsuvampi.

Tank Hogsia tehdessä suurin hyöty ohjelmakoodin testaamisesta oli selvästi verkkokoodia tehdessä. Ongelmia ilmeni huomattavia määriä, kun peli vastaanotti ja lähetti suuria määriä dataa. Pienetkin virheet datan lähetyksessä vaikuttivat todella suuresti pelin toimintaan, jonka vuoksi kaikki eri tapaukset piti testata tarkkaan. Pelin reittihakua piti testata hyvin paljon, koska se oli yksi suurimmista tehosyöpöistä peliä tehdessä. Unityn sisäinen Profiler-työkalu mahdollisti tarkan skriptien suorituskyvyn testaamisen. Sen avulla pystyi selvittämään, kuinka paljon tietyn ohjelmakoodin suorittamisessa kesti.

Pelin verkko-ominaisuuksia ohjelmoidessa kannattaa simuloida erilaisia verkkoyhteyksiä laidasta laitaan, koska muuten ei voi olla varma toimiiko peli toisen käyttäjän verkossa ollenkaan, vaikka omassa verkossa peli toimisi täydellisesti. Simuloimiseen käytettiin Photon Cloudin mukana tulevaa simulointityökalua, jolla voitiin simuloida erilaisia verkkoyhteyksiä. Eri verkkoyhteyksiä simuloidessa tuli selväksi, että on hyvin vaikea tehdä reaaliaikaista verkkopeliä huonoille yhteyksille, jonka vuoksi prototyyppi rajoitettiin väliaikaisesti toimimaan vain WiFi-yhteydellä 3G-yhteyden sijaan.

Sovelluksen suorituskykyä testattiin myös Applen iPhone 4 -laitteella. Testin perusteella huomattiin, että käytämme liian suuria tekstuureja kyseiselle laitteelle, joka aiheutti sen, että osa objekteista oli mustia, koska eivät saaneet oikeaa tekstuuri-informaatiota. Tämän saisi korjattua helposti käyttämällä pienempiä tekstuureja, koska ei ole mitään hyötyä käyttää iPadille suunniteltuja tekstuureja, jotka ovat liian tarkkoja iPhonen resoluutiolle.

Koska Tank Hogs on suunnattu Applen mobiililaitteistolle, voitiin käyttää Unityn lisäksi myös Applen tarjoamia testaustyökaluja pelin suorituskyvyn testaamiseen. Apple tarjoaa todella laajat työkalut iOS-ohjelmien testaamista varten. Niiden avulla saa tarkkaa tietoa prosessorin, näytönohjaimen ja muistin käytöstä sekä mahdollisista muistivuodoista. Ohjelmisto tarjoaa myös useita muita työkaluja, joita ei projektia tehdessä koitettu.

6 TYÖN TULOKSET

Lopputulokseksi saatu peli toimii hyvänä prototyyppinä. Pelin toteutus täyttää kaikki tavoitteet, joita prototyypiltä haluttiin. Pelistä tuli sen verran hyvä ja laaja, että voin käyttää sitä hyvänä referenssinä osaamisestani.

Peli-idea saatiin suunniteltua huomattavasti pidemmälle, kuin mihin prototyyppi haluttiin viedä. Peli-ideasta tuli mielestäni hyvin mielenkiintoinen, ja siinä voisi olla potentiaalia kaupalliseksi peliksi.

Peli toimii niin kuin pitää muuten, mutta sitä ei voi pelata kunnolla 3G- tai 4G-verkoissa. Peli kuitenkin toimii moitteettomasti WiFi-yhteyden yli, jonka vuoksi peli vaatii WiFi-yhteyden toimiakseen.

7 POHDINTA

Opinnäytetyöni tavoitteina oli tutkia Unityn käyttöä oikeassa peliprojektissa sekä saada selville iOS-laitteiden asettamat rajoitteet. Mielestäni tavoitteet saavutettiin ja haluttu informaatio saatiin.

Aiheeseen päädyin saatuani mahdollisuuden päästä mukaan oikeaan peliprojektiin, sillä tahdoin saada kokemusta pelialasta ennen kuin siirryn työelämään. Vaikka peliala olikin kiinnostanut minua nuoruudesta asti, olin kuullut paljon huonoa myös siitä. On totta, että pelialalla on aina kiire ja tekniikat muuttuvat vuosittain, mutta saamani kokemuksen mukaan se pitää työn samalla virkistävänä ja palkitsevana.

Opinnäytetyö tehtiin lähes kokonaan etätyöskentelynä. Tämä asetti omat haasteensa, kuten suurten assettien jakamisen vaikeutuminen ja aikataulujen venyminen sekä peli-idean suunnitteleminen Skypen välityksellä. Opinnäytetyö olisi valmistunut huomattavasti aikaisemmin, jos työskentely oltaisiin suoritettu alusta saakka samassa tilassa.

Opinnäytetyötä tehdessä tuli selväksi, että Unity on loistava työkalu pelikehityksessä. Unityn sisäänrakennettu profilointityökalu on todella hyvin toteutettu ja auttaa hyvin paljon pelin kehityksessä. Unityn viralliset dokumentaatiot ovat laajat ja lähes jokaisesta komponentista on esimerkki sen käytöstä. Unityn käyttäjäyhteisö on myös todella aktii-

vinen ja sen kautta saa nopeasti apua omiin ongelmiinsa. Yritys aikoo jatkossakin käyttää Unityä pelien tekemisessä.

Pelinkehityksen alussa kävi ilmi, että peli-idea oli hyvin keskeneräinen, eikä projektissa voitu käyttää perinteisiä ohjelmistokehityksen menetelmiä. Peli-ideaa vietiin eteenpäin Skype-keskusteluissa ja uusia ominaisuuksia tuli alkuvaiheessa viikoittain. Pelin ohjelmakoodia piti tästä johtuen kirjoittaa uudelleen useasti, sillä toiminnallisuudet muuttuivat usein dramaattisesti. Onneksi peliä kehitettäessä itselleni oli todella paljon uusien teknologioiden opiskelua, jonka vuoksi pystyin keskittymään niiden opiskeluun sillä välin kun peli-idea muuttui useasti.

Pelinkehitysprosessi oli äärimmäisen opettava kokemus. Vaikka olin käyttänyt Unityä hieman ennen projektin alkamista, oli sen käytössä todella paljon uutta opiskeltavaa. Myös uusien teknologioiden, kuten verkkoratkaisuiden luominen ja suunnitteleminen, nostivat osaamistasoani huomattavasti aikaisempaan verrattuna. Peliä tehdessä yritettiin käyttää kanban-pohjaista taulua, mutta sen käyttö jäi hyvin vähäiseksi peliä tehdessä ja helpommaksi tuli pitää tekstitiedostossa ranskalaisin viivoin, mitä tulisi vielä tehdä.

Verkkoratkaisuksi valittu Photon Cloud osoittautui hyvin toimivaksi järjestelmäksi. Sen avulla on helppo luoda verkkopeli, joka toimii lähes millä tahansa alustalla. Kehitysvaiheessa Photon Cloud tarjoaa hyvät tutoriaalit, joiden avulla pääsee hyvin alkuun. Jos Photon Cloud ei riitä, myös ExitGames tarjoaa pakettia, jolla voi ostaa serverikoodin ja pystyttää oman palvelimen.

Peliä tehdessä minulle on varmistunut että haluan tehdä pelejä vielä tulevaisuudessakin. Tulevaisuus näyttää lupaavalta, koska tämän hetkisen tilanteen mukaan mobiilipelejä kehitetään vielä hyvin pitkään ja laitteiden tehot kasvavat vuosittain. Täytyy kuitenkin todeta, että oma kiinnostukseni on enemmän suuremman luokan projekteissa. Mobiilipeleistä on kuitenkin hyvä aloittaa, varsinkin Suomessa. Mobiilipelejä tehdessä saa tarvittavaa kokemusta koodin optimoinnista mahdollisimman nopeaksi, sillä mobiililaitteet edellyttävät tiukkaa optimointia niiden ollessa edelleen hitaita verrattuna keskiverto tietokoneeseen.

Tank Hogs -peli saatiin hyvälle alulle. Peli-ideaa vietiin hyvin pitkälle ja sen laajuuden vuoksi ikävä kyllä yritys ei jatka projektia heti, vaan toteuttaa toisen pelin ensimmäisenä mobiilipelinään. Kehitetystä peli-ideasta tuli hyvä, mutta liian laaja näin pienen yrityksen toteuttavaksi ilman suurempaa kehitystiimiä. Peli toimii niin kuin pitää, mutta pelin ohjelmakoodia täytyy muokata valtavasti, jos pelin kehitystä jatketaan. Ohjelmakoodissa on paljon pikaisia ratkaisuja, joihin päädyttiin kiireellisen aikataulun sekä viikoittain muuttuvan spesifikaation vuoksi.

LÄHTEET

Angry Birds Reaches 1 Billion Downloads. 2012. PCWorld-verkkosivut. Viitattu 20.3.2014 http://www.pcworld.com/article/255337/angry_birds_reaches_1_billion_downloads.html

Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio. 2013. The Vergen -verkkosivut. Viitattu 2.3.2014 http://www.theverge.com/2013/10/22/4866302/apple-announces-1-million-apps-in-the-app-store/

Application. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 http://docs.unity3d.com/Documentation/ScriptReference/Application.html

AssetStore. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Manual/AssetStore.html

Attributes. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Manual/Attributes.html

Creating And Using Scripts. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Manual/CreatingAndUsingScripts.html

Create and Destroy Objects. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Manual/CreateDestroyObjects.html

Compare iPad models. 2014. Apple-verkkosivut. Viitattu 31.3.2014 https://apple.com/ca/ipad/compare/

Controlling GameObjects Using Components. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Manual/ControllingGameObjectsComponents.html

Coroutines. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Manual/Coroutines.html

Event Functions. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Manual/EventFunctions.html

Fast Facts. 2014. Unity-verkkosivut. Viitattu 17.3.2014 http://unity3d.com/company/public-relations/

GameObject. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014 https://docs.unity3d.com/Documentation/Components/class-GameObject.html

GSM. 2014. Wikipedia-verkkosivut. Viitattu 30.3.2014
http://en.wikipedia.org/wiki/GSM

Instantiating Prefabs at runtime. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014
https://docs.unity3d.com/Documentation/Manual/InstantiatingPrefabs.html

iPad. 2014. Wikipedia-verkkosivut. Viitattu 31.3.2014
http://en.wikipedia.org/wiki/IPad

iPhone. 2014. Wikipedia-verkkosivut. Viitattu 1.4.2014
http://en.wikipedia.org/wiki/IPhone

iOS. 2014. Wikipedia-verkkosivut. Viitattu 25.4.2014
http://en.wikipedia.org/wiki/IOS

iOS Developer Program. 2014. Apple-verkkosivut. Viitattu 25.4.2014
https://developer.apple.com/programs/ios/

Mobiilipeli. 2013. Wikipedia-verkkosivut. Viitattu 25.3.2014
http://fi.wikipedia.org/wiki/Mobiilipeli

Mobile device. 2014. Wikipedia-verkkosivut. Viitattu 25.3.2014
http://en.wikipedia.org/wiki/Mobile_device

Mobile operating system. 2014. Wikipedia-verkkosivut. Viitattu 26.4.2014
http://en.wikipedia.org/wiki/Mobile_operating_system

Mobile Phone. 2014. Wikipedia-verkkosivut. Viitattu 30.3.2014
http://en.wikipedia.org/wiki/Mobile_phone

MonoBehaviour. 2014. Unity Docs -verkkosivut. Viitattu 27.3.2014
https://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html

Multiplatform. 2014. Unity-verkkosivut. Viitattu 25.4.2014
https://unity3d.com/unity/multiplatform

Nokia N-Gage QD - The Same, But Different. 2004. Mobiledia-verkkosivut. Viitattu 25.3.2014
http://www.mobiledia.com/phones/nokia/n-gage-qd.html

PDA. 2014. Wikipedia-verkkosivut. Viitattu 31.3.2014
http://en.wikipedia.org/wiki/PDA

Performance Optimization. 2014. Unity Docs -verkkosivut. Viitattu 26.4.2014
http://docs.unity3d.com/410/Documentation/ScriptReference/index.Performance_Optimization.html

Photon Unity Networking (PUN) Compared To Unity Networking (UN). 2014. ExitGames-verkkosivut. Viitattu 27.4.2014
http://doc.exitgames.com/en/pun/current/reference/pun-and-un

Prefabs. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014
https://docs.unity3d.com/Documentation/Manual/Prefabs.html

Release Archive. 2014. Unity-verkkosivut. Viitattu 25.4.2014
http://unity3d.com/unity/whats-new/archive

Sell Assets. 2014. Unity-verkkosivut. Viitattu 27.4.2014
https://unity3d.com/asset-store/sell-assets

Sharp J-SH04: World's First Ever Phone With Integrated Camera. 2010. Gadgetizor-verkkosivut. Viitattu 27.4.2014
http://gadgetizor.com/sharp-j-sh04-worlds-first-ever-phone-with-integrated-camera-pictures-2001/5482/

Special Folders and Script Compilation Order. 2014. Unity Docs -verkkosivut. Viitattu 25.3.2014
https://docs.unity3d.com/Documentation/Manual/ScriptCompileOrderFolders.html

Special Folder Names in your Assets Folder. 2014. Unity Wiki -verkkosivut. Viitattu 20.4.2014
http://wiki.unity3d.com/index.php/Special_Folder_Names_in_your_Assets_Folder

The Evolution of Mobile Games. 2013. ESA-verkkosivut. Viitattu 27.3.2014
http://www.theesa.com/games-improving-what-matters/mobile-games.asp

Unity (game engine). 2014. Wikipedia-verkkosivut. Viitattu 5.4.2014
http://en.wikipedia.org/wiki/Unity_(game_engine)

Älypuhelimia ja tabletteja myytiin vuonna 2013 yhteensä lähes miljardilla eurolla. 2014. Kotek-verkkosivut. Viitattu 27.4.2014
http://www.kotek.fi/tiedotteet/aelypuhelimia-ja-tabletteja-myytiin-vuonna-2013-yhteensae-laehes-miljardilla-eurolla/