OULUN AMMATTIKORKEAKOULU

Henri Viljamaa

UNITY3D, PALVELIMET JA MOBIILIPELIN OPTIMOINTI

UNITY3D, PALVELIMET JA MOBIILIPELIN OPTIMOINTI

Henri Viljamaa
Opinnäytetyö
Kevät 2017
Tietotekniikan koulutusohjelma
Hyvinvointiteknologian suuntautumisvaihtoehto
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, hyvinvointiteknologian suuntautumisvaihtoehto

---

Tekijä: Henri Viljamaa
Opinnäytetyön nimi: Unity3D, palvelimet ja mobiilipelin optimointi
Työn ohjaajat: Jukka Jauhiainen ja Kaisa Orajärvi
Työn valmistumislukukausi ja -vuosi: Kevät 2017 Sivumäärä: 74 sivua

---

Vuodesta 2014 lähtien tietotekniikan koulutusohjelmassa on ollut kokeella koosteopinnäytetyöt. Tämä opinnäytetyö koostuu kahdesta osasta. Ensimmäisestä 5 opintopisteen osakokonaisuudesta, joka valmistui vuoden 2015 keväällä ja toisesta 10 opintopisteen osakokonaisuudesta, joka valmistui vuoden 2017 keväällä. Molempien osakokonaisuuksien projektit on tehty Unity3D-pelinteko alustaa käyttäen.

Ensimmäisen osan tarkoituksena oli kerätä tietoa jostakin itseään kiinnostavasta aiheesta ja koostaa se tietopaketiksi. Aiheeksi valikoituivat palvelimet ja Unity3D. Ensimmäisessä osassa käydään läpi, mitä ylipäätään ovat palvelimet, minkälaisia erilaisia vaihtoehtoja palvelimiksi on, miten palvelin otetaan käyttöön ja miten sitä käytetään Unity3D:ssä. Aiheen pohjalta toteutettiin pieni mobiilipeli, jossa toisen pelaajan laite toimii palvelimena. Ensimmäinen osa antoi hyvän yleiskuvan siitä, mitä palvelimet ovat ja kuinka niitä voidaan hyödyntää sovelluksissa. Aihe oli kiinnostanut jo pidempään ja opinnäytetyö antoi hyvän mahdollisuuden sen tutkimiseen.

Alkuperäisen kaavan mukaan koosteopinnäytetyö olisi toteutettu kolmessa osassa, mutta toinen ja kolmas osa yhdistettiin aiheen laajuuden vuoksi yhdeksi kokonaisuudeksi. Yhdistetyn toisen ja kolmannen opinnäytetyön aiheena oli jo aiemmissa projektikursseissa aloitetun 3D-mobiilipeliprojektin pohjalta 3D-pelin suunnittelun ja optimoinnin tutkiminen. Työssä kerrotaan peliprojektista *Tales From The North* ja siitä, kuinka sitä on toteutettu mobiililaitteita varten. Samalla on haettu tietoa ja yleisiä käytänteitä peliprojektin suunnitteluun ja optimointiin liittyen.

Työ antoi hyvän käsityksen siitä, miten paljon resursseja ja työtunteja sovelluksen viimeistely julkaisukelpoiseksi vaatii. Työn aikana opittiin, miten peliprojekti voidaan pakata pelilaitteen tehonkulutusta ajatellen pienemmäksi ja vähemmän kuluttavaksi.

Avainsanat: Unity, palvelin, optimointi

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology and Telecommunication, Medical Engineering

Author: Henri Viljamaa
The names of parts of the thesis: Unity3D, Servers and optimization for mobile devices
Supervisors: Jukka Jauhiainen and Kaisa Orajärvi
Term and year when the thesis was submitted: Spring 2017 Pages: 74 pages

The thesis consist of two separate parts. This type of method has been experimenting since 2014. The first part is a sub-total of five credits completed in spring 2015. The second part was completed in the spring of 2017 and has a total of ten credits. Both subdivisions are implemented on Unity3D game development platform.

The purpose of the first part was to gather information about something that is of interest to itself and to compile it into an information packet. Unity3D and servers were selected as the topic. The subject was based on a small mobile game where the other player's device is a server. The first part gave a good overview of what servers are and how they can be used in applications. The subject had long been interested and the thesis offered a good opportunity to study it.

According to the original plan, the thesis project would have been implemented in three parts but the second and third parts were merged into one entity.
The topic of the combined second and third thesis was the study of the design and optimization of the 3D mobile game project initiated at earlier project courses. The thesis talks about the mobile game project *Tales From The North* and how it is designed and optimized for mobile devices. At the same time, information and common practices for design and optimization have been sought.

The work provided a good idea of how much resources and hours were required to finalize the application for publishing. During the course of the thesis, it was learned how the game project can be packed in order to reduce the power consumption of the game device to a lesser extent.

Keywords: Unity, servers, optimization

SISÄLLYS

1 JOHDANTO

Informaatioteknologian koulutusohjelmassa on vuodesta 2014 lähtien kokeiltu koosteopinnäytetöitä. Koosteopinnäytetyössä normaali 15 opintopisteen opinnäytetyö jaetaan kolmeen pienempään viiden opintopisteen opinnäytetyöhön. Opinnäytetyö oli mahdollista suorittaa kahdessa tai kolmessa osassa. Kahden osaopinnäytetyön mallissa ensin tehtiin pienempi viiden opintopisteen työ ja lopuksi isompi kymmenen opintopisteen työ. Ensimmäisessä työssä tarkoituksena oli hankkia haluamastaan aiheesta tietoa ja koostaa se tietopaketiksi. Myöhemmissä vaiheessa tarkoituksena oli tehdä projekteja, jonka perusteella opinnäytetyö kirjoitetaan.

Kokeilu oli siinä mielessä hyvä, että ensimmäinen osa tehtiin pääsääntöisesti opettajien valvoessa ja opastaessa, jolloin sen työstäminen oli jouhevaa, kun apu oli aina saatavissa. Työn aloittaminen ei ollut niin vaikeaa, kuin se voisi olla ison opinnäytetyön kohdalla. Ensimmäisen osan tekeminen auttoi toisen ja kolmannen osan kirjoittamista, kun oli jo tehty aikaisemmin vastaava työ.

Suurin osa opiskelijoista valitsi kolmen sijaan kahden opinnäytetyön tekemisen. Työ venyy helposti viiden opintopisteen kokoisesta työstä suurempaan, jolloin kolmea pienempää työtä tehdessä materiaalia normaalin viidentoista opintopisteen työhön verrattuna olisi tullut reilusti liikaa.

2 OPINNÄYTETYÖN OSAN 1 AIHE, TAVOITE JA TULOKSET

Opinnäytetyön ensimmäisen osan (liite 1) aiheena olivat palvelimet ja Unity3D. Tarkoituksena oli tutustua erilaisiin palvelinvaihtoehtoihin, kartoittaa tietämystä niistä ja tehdä pienimuotoinen peliprojekti, jossa pelataan palvelimen välityksellä toista pelaajaa vastaan. Opinnäytetyön taustoihin liittyy mobiililaitteissa ja varsinkin pelimaailmassa lisääntynyt tarve reaaliaikaiseen tiedonsiirtämiseen. Yhä useammat käyttäjät vaativat mobiilipeleihin moninpelimahdollisuuden. Työssä selvitettiin, mitä eri lisäosia Unityyn täytyy liittää, jotta palvelimet saataisiin toimimaan. Työssä selvitettiin myös, minkälaista koodaamista tarvitaan palvelimen käyttöönotossa ja ylläpidossa.

Opinnäytetyön tavoitteena oli oppia käyttämään ja muokkaamaan palvelimen ylläpidossa käsiteltävää koodausta. Työssä luodaan kokonaiskuva palvelimen valinta- ja luomisprosessista mobiililaitteille. Valitaan järkevimmät vaihtoehdot ja näytetään, kuinka asiat tapahtuvat käytännössä. Lopulta katsotaan, kuinka esimerkkiprojekti toimii kahdella mobiililaitteella.

Opinnäytetyössä saatiin hyvä yleiskäsitys siitä, mitä palvelimet ovat ja miten niitä voi ottaa käyttöön sekä tuotettiin pieni palvelinta käyttävä mobiilipeli. Ensimmäistä osaa lukiessa tulee huomioida, että työssä käytettävät koodit ja menetelmät Unity3D:n kanssa on tehty veriolla 4.0. Palvelimen luonti ja ylläpito ovat muuttuneet uuteen 5.0-versioon.

3 OPINNÄYTETYÖN OSAN 2 + 3 AIHE, TAVOITE JA TULOKSET

Opinnäytetyön toisen ja kolmannen osan (liite 2) aiheena oli 3D-mobiilipelin optimointi ja suunnittelu keskivertokuluttajan pelilaitteelle. Työn taustalla oli jo aiemmissa projektikursseissa aloitettu toimintaroolipeli *Tales From The North.* Työssä käydään läpi yleisiä menetelmiä peliprojektin suunnittelun ja optimoinnin kannalta sekä kerrotaan, kuinka näitä toteutettiin projektissa *Tales From The North.*

Tavoitteena oli oppia niitä seikkoja, joilla 3D-peli saadaan optimoitua tehonkulutuksen kannalta niin kevyeksi, että sitä voidaan keskivertokuluttajan mobiililaitteella pelata sujuvasti. Ohjelmistotuotannosta ja pelisuunnittelusta haluttiin hakea yleisiä käytänteitä ja hyödyntää niitä omassa projektissa.

Työtä tehdessä opittiin runsaasti siitä, kuinka ohjelmakoodia, 3D-mallinnusta ja 2D-grafiikkaa kannattaa suunnitella ja optimoida peliprojektissa. Työn aikana saatiin valmiiksi testiversio, joka julkaistiin Android-käyttöjärjestelmän sovelluskaupassa (Google Play).

4 YHTEENVETO

Kokonaisuutena opinnäytetyön tekeminen antoi paljon tietoa projektien suunnittelusta, toteutuksesta ja viimeistelystä. Tietoa karttui yleisesti ohjelmistotuotantoon, ohjelmointiin, palvelimiin, pelisuunnitteluun, pelin tekemiseen, 3D-mallintamiseen, 2D-grafiikan kuvankäsittelyyn ja piirtämiseen sekä ohjelmiston optimointiin liittyen. Unity3D-pelimoottorin, Blender-3D-mallinnus ohjelman, Krita-kuvankäsittely ohjelman ja C#-ohjelmointikielen käyttö luonnistuvat töiden jälkeen hyvin.

Peliohjelmointi ei suoraan kuulu tietotekniikan alla olevan hyvinvointiteknologian koulutusohjelmaan, vaan innostus aiheen tutkimiseen lähti omasta tahdosta. Projektikurssit sekä opinnäytetyöt sai tehdä itselle mieleisistä aiheista, jolloin into niiden tekemisessä pysyi hyvänä. Turhaan ei projekteja tullut tehtyä, sillä opinnäytetöiden ja projektien lomassa käytetyn C#-ohjelmointikielen oppiminen poiki myöhemmässä vaiheessa työpaikan.

Ensimmäisen osan aikana suurin oppi tuli siitä, kuinka suuria tekstikokonaisuuksia työstetään ja miten tietoa haetaan hyvistä lähteistä. Samalla itseä kiinnostava aihe palvelimet ja niiden käyttöönotto tuli tutuksi. Toisen osan aikana suuren projektin suunnittelu, toteutus ja projektin viimeistely olivat suurimmat oppimisen kohteet. Samalla peliala, peliprojektien suunnittelu ja toteutus sekä ohjelmistosuunnittelu tulivat tutuiksi.

Näkyvät tulokset opinnäytetyöstä ovat molempien osakokonaisuuksien aikana valmistuneet pelien testiversiot. Ensimmäisen osan kilvanajo-peliä ei ole enää saatavilla missään, mutta toisen ja kolmannen osan tuotos *Tales From The North* on saatavilla GooglePlay-sovelluskaupassa.

LIITTEET

Liite 1 Palvelimet ja Unity3D

Liite 2 Pelin suunnittelu ja optimointi mobiilialustalle Unity3D:llä

LIITE 1

OULUN AMMATTIKORKEAKOULU

Henri Viljamaa

PALVELIMET JA UNITY3D

PALVELIMET JA UNITY3D

Henri Viljamaa
Opinnäytetyö, osa 1
Kevät 2015
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön taustoihin liittyy mobiililaitteissa ja varsinkin pelimaailmassa lisääntynyt tarve reaaliaikaiseen tiedonsiirtämiseen. Yhä usemmat käyttäjät vaativat mobiilipeleihin moninpelimahdollisuuden sekä tarpeen kilpailla ennätyksistä maailmanlaajuisesti. Työn tarkoituksena on selvittää, mitä eri lisäosia Unityyn täytyy liittää ja minkälaista koodaamista tarvitaan palvelimen käyttöönotossa ja ylläpidossa. Työssä selvitetään myös, mikä on palvelin ja mistä palvelin on järkevin ottaa. Palvelinta kokeillaan käytännössä kahden mobiililaitteen välillä ja luodaan sitä varten Unityssä esimerkkiprojekti, jonka pohjalle palvelin asetetaan. Työ toteutetaan Unity3D -kehitysalustalla, joka on alusta pelisovelluksille mobiililaitteiden lisäksi myös monelle muulle laitteistolle.

Opinnäytetyön tavoitteena on oppia käyttämään ja muokaamaan palvelimen ylläpidossa käsiteltävää koodausta. Työssä luodaan kokonaiskuva palvelimen valinta- ja luomisprosessista mobiililaitteille. Valitaan järkevimmät vaihtoehdot ja näytetään, kuinka asiat tapahtuvat käytännössä. Lopulta katsotaan, kuinka esimerkkiprojekti toimii kahdella mobiililaitteella.

2 PALVELIN

Palvelimella (serverillä) tarkoitetaan tietokoneessa ylläpidettävää palvelinohjelmistoa tietoliikenteen välille. Palvelimien tehtävänä on tarjota erilaisia palveluja verkon tai paikallisten yhteyksien välille. Laitetta, joka muodostaa yhteyden palvelimeen, kutsutaan asiakkaaksi. Taulukossa yksi on lueteltu yleisimmät palvelintyypit ja niiden ominaisuudet.

Taulukko 1 Palvelintyypit (1.)

Yleisimpiä palvelimia	
Sovelluspalvelin	Suorittaa ohjelmia tietokoneessa (tietojärjestelmät).
Nimipalvelin	Numeroi verkossa käytettäviä palveluita ja koneita nimien mukaan ja vastaavasti nimeää verkko-osoitteita.
WWW-palvelin	Antaa tiedostot ja muun sisällön mitä www-selain pyytää.
Sähköpostipalvelin	Vastaanottaa ja toimittaa sähköpostia.
Tiedostopalvelin	Jakaa palvelimelta massamuistitilaa ja tiedostoja asiakasohjelmistojen käyttöön.
Tietokantapalvelin	Hallinnoi tietokantoja, joita muut sovellukset voivat käyttää.
Tulostuspalvelin	Käsittelee sovellukselta tulevia tulostuspyyntöjä, laittaa ne jonoon ja syöttää ne tulostimelle.
Pelipalvelin	Internetissä toimivan, yleensä moninpelipalvelimen ylläpitäjä.

Palvelin on mahdollista sijoittaa joko fyysiselle kotikoneelle, mobiilipalvelimelle tai pilvipalveluun. Pilvipalvelulla tarkoitetaan käyttäjälle tarjottavaa virtuaalista resurssia. Resurssien fyysiset laitteistot, kuten palvelintietokoneet, eivät näy palvelimien käyttäjille. Yhä enemmän palvelimia pystytettäessä valitaan ratkaisumalli, jossa palvelinsovellukset asetetaan omien fyysisten palvelinkoneiden sijaan johonkin tietokonekeskuksen tarjoamaan pilvipalveluun. Tällöin ei tarvitse investoida laitteistoihin ja niiden päivityksiin, vaan pilvipalvelimen ylläpitäjä tarjoaa käyttäjälleen nämä ominaisuudet. (2.)

Mobiilipalvelimet ovat vielä uusi konsepti IT-maailmassa. Yritykset ovat tottuneet asentamaan palvelimensa lämpösäädeltyihin huoneisiin, jotka ovat täynnä koneita, johtoja ja raskaita hyllystöjä. Mobiilipalvelin on kuin tästä perinteisistä konehuoneesta tehty muistio, joka pyrkii tekemään samat ydintoiminnot ilman konehuoneen rajoituksia. Mobiilipalvelimet ovat enimmäkseen olleet poliisien ja erikoisjoukkojen käytössä, mutta vähitellen mobiilipalvelimet ovat siinä pisteessä, että ne alkavat olla varteenotettava vaihtoehto myös yrityksissä. Mobiilipalvelimet ovat jo yhtä päivitettävissä olevia kuin perinteisetkin palvelimet. (3.) Kuvassa yksi on mobiilipalvelin salkkumallissa.

Kuva 1. Mobiilipalvelin

2.1 Mobiilisovellusten palvelimet

Tänä päivänä mobiilisovellusten kehittäjät haluavat harvoin toteuttaa sovelluksiensa käyttöliittymiä ilman internetpalveluita. Mobiilisovellusten palvelimet ovat suurimmalta osaltaan käyttöjärjestelmien kehittäjille tarkoitetuissa pilvipalveluissa. Esimerkiksi Azure tarjoaa palvelimet iOs, Android ja Windows-kehittäjille,

Amazonilla on AWS SDK -Android-kehittäjille ja AWS SDK -iOS-kehittäjille sekä Google tarjoaa Google App Engine -palvelun ja sen lisäosat Eclipse-kehitysalustalle omille tuotekehittäjilleen. (4.) Esimerkiksi Azure lupaa omilla nettisivuillaan palvelut käyttäjien tunnistamiseen (tietokannat) ja muokattavissa olevan backend (välityspalvelin) logiikan C#-ohjelmointikielelle. Azuren palvelimet käyttävät tietokantoinaan Active Directoryä, joka on mikrosoftin tuottama hakemistopalvelu. Azurella on mahdollista asettaa auto-skaalaus määrittämään tarvittava kapasiteetti palvelimelle. Se kasvattaa tai pienentää kapasiteettiä liikennevirran perusteella ja laskuttaa sen mukaan. (5.)

2.2 Verkkopelien palvelimet

Verkkopelien palvelimet on yleisesti toteutettu palvelin-asiakasmallin mukaisella järjestelmällä. Palvelinsovellus vastaa pelin suorittamisesta ja sen pelisimulaation toteuttamisesta. Pelaajat käyttävät asiakassovelluksia, jotka välittävät pelaajien tekemät muutokset ja komennot palvelimelle. Palvelimella muutetaan pelin tilaa saatujen komentojen mukaan ja lähetetään tiedot muille pelaajille. (6.)

Myös asiakassovellus tekee omaa simulaatiotansa, jotta pelaajan kokemus pelimaailmasta olisi reaaliaikainen. Pelikokemuksen mahdollisimman aidoksi tuottamisen vuoksi, pyritään toimintojen vaikutusta ennustamaan. Esimerkiksi kun pelaaja antaa pelille komennon, asiakasohjelma suorittaa niiden vaikutukset pelimaailmassa viimeisimmän palvelimelta saadun tiedon perusteella. Tätä komentoa tarvittaessa korjataan, kun palvelimelta saadaan päivitetyt tiedot pelin tilasta. (7.)

Pelisimulaation toteuttaminen tapahtuu siten, että asiakkaalta palvelimelle tulevat tiedot kerätään puskuriin ja prosessoidaan asetetuin väliajoin. Tiedoissa on mukana asiakkaan aikaleima, jonka perusteella tapahtumat prosessoidaan oikeassa järjestyksessä. Tämän jälkeen simuloinnin uusi tila toimitetaan kaikille asiakkaille. Kun tätä päivitysväliä säädetään, voidaan kontrolloida simuloinnin tarkkuutta. Huomioon on otettava myös se, että jos verkon viive on suurempi kuin päivitysvälit, saattaa pelimaailmassa syntyä ristiriitoja. (8.)

Yksinkertaisimmillaan moninpelisovellus asiakkaan päätteellä on ns. tyhmä pääte. Sen tehtäviin kuuluvat vain pelaajan suorittamien komentojen välittäminen palvelimelle ja pelisimulaation tilan, kuvan ja äänten päivittäminen päätteellä. Tällaisella rakenteella minimoidaan peleissä huijaaminen sillä palvelin kontrolloi täysin peliä eikä asiakas voi tehdä omalla päätteellään asioita ilman, että palvelin olisi siitä tietoinen. (9.)

Suuremman kokoluokan peleissä on yleensä myös mukana tietokantapalvelin, joka ylläpitää käyttäjien tietoja. Lisäksi yleisiä palvelinsovelluksia ovat pikaviestipalvelimet ja WWW-palvelimet, jotta pelin tietoja voitaisiin esitellä selaimessa. Joukkueiden ja pelien muodostamiseen käytetään ns. matchmaking-palvelimia. Verkkopelien viestintä rakennetaan käyttämään IP-verkkoja, koska lähes kaikki pelaamiseen soveltuvat laitteet hyödyntävät tätä internetin protokollaa. (8.)

3 UNITY3D JA PALVELIN

Unity on useille eri laitteille suunniteltu pelinkehitysalusta. Sen kehitystyö on aloitettu vuonna 2001. Ensimmäinen julkaisu on vuodelta 2005. (10.) Unity on kehittäjilleen ilmainen alusta, johon voidaan ostaa lisätoimintoja lisenssin myötä. Ilmaisella versiolla saa tuottaa rajattomasti sovelluksia kaupalliseen käyttöön, ja se vaatii ainoastaan 2 sekunnin mittaisen Unityn logon näyttämisen tuotetun sovelluksen käynnistysvaiheessa. Unity oli myös ensimmäisiä iPhone-yhteensopivia pelinkehitystyökaluja. Sen suosio mobiilisovellusten kehityksessä johtunee suurelta osin myös tästä syystä.

Unityyn on integroitu sen oma pelimoottori ja helppokäyttöinen editori. Editorissa luodaan pelimottoorin komponentteja hyödyntäen objekteja, joita yhdistelemällä toteutetaan pelikokonaisuus. Peliä voidaan kokeilla suoraan editorissa, mikä helpottaa huomattavasti pelin toteutusta. Esimerkiksi objektien parametreja voidaan muokata lennosta pelitilan aikana. (11.)

Ohjelma on tehty erityisesti yksittäisten tai pienten yritysten helppokäyttöiseksi pelintekoalustaksi. Toimintoja on paljon automatisoitu ja Asset Storen työkalujen ja tiedostojen kauppapaikasta voi helposti ladata ilmaisia tai maksullisia lisäosia. (8.)

3.1 Palvelimet unityssä

Unityssä on UDP-protokollaa käyttävä RakNet-verkkokirjasto, joka toteutuu IP-verkkojen yli. RakNet on yksinkertaisten moninpelattavien pelien toteuttamiseen suunniteltu väline. Sen säätömahdollisuudet ovat melko rajalliset, joten kovin monimutkaisia pelejä sillä ei voi toteuttaa. Säädettävissä on lähinnä datapakettien lähetysväli. Lähetysväli ilmaistaan lähetyskertojen määränä sekunnissa. (12.)

Unity tukee vain palvelinmuotoa, jossa yksi pelaajista toimii palvelimena ja muut yhdistävät tähän pelaajaan. Palvelin on käynnistettävissä, ja siihen liittäminen tapahtuu yksinkertaisella funktiokutsulla. (13.) Olemassa on kuitenkin monia

erillisiä palvelinsovelluksia, joissa on tuki Unity-pelimoottorille. Osa niistä mahdollistaa myös Unityn editorin käytön.

3.1.1 NetworkView ja tilasynkronointi

NetworkView-komponentit ovat pelissä objektien tarkkailijoita, jotka voidaan asettaa seuraamaan yhden objektin tilanmuutoksia. Se lähettää muutoksia tilastaan muille pelaajille. Komponenteissa on oma tunnistenumero, jonka perusteella muut pelaajat tunnistavat, mistä viesti tulee. Tunnistenumero kertoo, mille objektille NetworkView-viesti on tarkoitettu. Tätä kutsutaan myös nimellä tilasynkronointi. (14.)

Synkronointi on toteutettavissa parillakin eri tavalla. Deltapakkaukseksi kutsutaan tapaa lähettää vain tapahtuneiden muutosten suuruudet objektissa. Deltapakattu tieto lähetetään luotettavalla siirtomenetelmällä, koska jos delta-pakkauksia hukkuisi, niin pelien tilat eroaisivat toisistaan. Toinen vaihtoehto on lähettää aina kerrallaan koko komponentin tila. Kyseinen tapa syö enemmän tiedonsiirtoresursseja kuin deltapakattujen tietojen lähettäminen, mutta jos matkalla hukkuu paketteja, niitä ei tarvitse lähettää uudelleen pakettien toisistaan riippumattomuuden takia. Vastaanottaja pystyy päivittämään tietonsa oikeaan tilaan minkä tahansa vastaanotetun paketin avulla. Jos paketti ei tule perille, se saattaa aiheuttaa näkymässä objektien äkillistä liikahtelua. Se ei kuitenkaan vaikuta itse peliin. (12.)

3.1.2 Remote procedurecall

NetworkViewin avulla voidaan tehdä myös etäkutsuja tietyn skriptin metodeihin ja parametreihin. Kutsuja on mahdollista lähettää rajatusti tai kaikille sessiossa oleville pelaajille. Näitä etäproseduurikutsuiksi määriteltyjen tietoja lähetetään välittömästi kun kutsu on tehty, eikä synkronointipakettien päivitysväli ole niihin sidoksissa. (15.)

RPC tarkoittaa verkon yli lähetettäviä metodeja. Näitä metodeja on merkittävä kyseisellä RPC-määreellä. Unity ei tunnista verkon yli lähetettäessä kuin vain muutaman datatyypin. Sen vuoksi ohjelmaa käännettäessä metodien paramet-

rit, jotka on merkitty RPC-määreellä, käydään läpi. Jos on virheitä, ne ilmoitetaan käyttäjälle. Etäproseduurikutsut tehdään kutsumalla RPC-metodia NetworkView-komponentissa. (15.)

3.2 Palvelin käyttäjien ulkopuolella

Ensisijaiset kriteerit pienen yrityksen tarpeita vastaavalle palvelimelle ovat tuotteen kypsyys, kehitysaktiivisuus ja soveltuvuus tarvittavaan käyttötarkoitukseen. Tuotteen kypsyydellä tarkoitettaan sitä, että palvelintarjoajan lupaamat ominaisuudet toteutuvat käytännössä ja että ne toimivat luotettavasti. Dokumentaation kattavuus ja sen laatu ovat kypsyyttä arvioitaessa tärkeässä roolissa. Kehitysaktiivisuutta tutkiessa on huomioitava, että jos tuotetta ei kehitetä, se saattaa lakata toimimasta sen käyttämien ohjelmistokirjastojen tai rajapintojen päivityessä. Kehitysaktiivisuutta voidaan tarkastella tuotteiden julkaisu- ja päivityshistorioiden, uutisten ja tiedotteiden perusteella. (8.)

Kun palvelinohjelmistoa suunnitellaan nimenomaan virtuaalimaailman toteuttamiseen, täytyy ottaa huomioon palvelimeen yhdistetty tietokanta ja sisäänrakennetut kuormantasaustoiminnot. Tuki ulkoisille vastaaville toiminnoille suurten käyttäjämäärien mahdollistamiseksi on myös olennaista. Palvelimen lisäksi tarvitaan rajapinta, jonka kautta peli käyttää palvelimen toimintoja. Tätä varten tuotteen kehittäjä tarjoaa yleensä ohjelmistokehityspaketin (SDK), joka sisältää joko binääri- tai lähdekoodimuodossa palvelimen kanssa viestimiseen tarvittavat toiminnot. Tuotetta valittaessa siis Unity-yhteensopiva SDK on edellytys. (8.) Määritelmällä CCU tarkoitetaan sitä, kuinka monta samanaikaista käyttäjää palvelimella voi yhtäaikaisesti olla.

Taulukossa 2 on vertailtuna Unity-yhteensopivia palvelinvaihtoehtoja. Joissakin tuotteista on saatavilla kattavammin ominaisuuksia kuin toisissa vaihtoehdoista. Yhteensopivuuden toteuttaminen on kuitenkin käyttäjästä kiinni. Esimerkiksi Unitypark toimii kokonaan Unityn varassa ja säädetään editorin avulla, kun taas muissa tuotteissa on pelkkä koodirajapinta binäärimuotoiseen kirjastoon. Pelipalvelimien perustoimintojen lisäksi yleisiä lisätoimintoja ovat esimerkiksi tieto-

kantapalvelimet ja pikaviestijärjestelmät. Laskennan suoritusteho on riippuvainen ohjelmistoalustasta, joita tyypillisesti ovat joko .NET, java tai Mono. Palvelinkoneen tai pilvipalvelun tarjoama suorituskyky on myös olennainen osa laskentaa. (8.)

Taulukko 2 Ulkoisia palvelinvaihtoehtoja (8.)

Tuote	**UnityPark Suite**	**Photon Server**	**Player.IO**	**Electroserver 5**
Kehittä- jä/palveluntarj oaja	MuchDifferent	Exit Games	Yahoo! (ent. PlayerScale)	Electrotank
URL	http://www.muchdifferent.com/?page=game-unitypark	https://www.exitgames.com/en/OnPremise	http://playerio.com/	http://www.electrotank.com/es5.htm l
Palve- lun/tuotteen tyyppi	Palvelinohjelmisto	Palvelinohjelmis- to	Pilvipalvelu	Palvelinohjelmisto
Tuetut palvelinkäyttöjär- jes- telmät	Windows, Linux, OSX	Windows	-	Windows, Linux, OSX
Ilmainen / kokeiluversio saatavilla?	On	On	On	On
Hostauspalvelu saatavilla?	Ei	Ei	On (kuuluu tuotteeseen)	Ei
Kohdemarkkinat / mihin tarkoitettu	Kaikenlaiset verkkopelit. MMO:t mainittu erikseen.	Kaikki verkkopelit	Flash- ja Unity- pohjaiset monin- pelit	Kaikki verkkopelit, saatavilla myös MMO-kehitykseen tarkoitettu tuotepaketti
Lisenssityypit ja hinnat	Indie-lisenssi 550EUR / julkaistu tuote, enterprise-lisenssit neuvoteltavissa erikseen	Sovellus-/palvelinkohtaiset lisenssit: <= 100 CCU: ilmainen => 100 CCU $500 > 500 CCU: $3500 Tilauslisenssit (per sovellus/serveri): <= 100 CC: ilmainen <= 500 CCU: $25 / kk > 500 CCU: $175 / kk	<= 500 CCU: Ilmainen <= 5000 CCU: 24,95$/kk <= 25000 CCU: 500$/kk > 25000 CCU: Neuvoteltava erikseen Lisensseissä myös rajoitukset dataliikenteelle, tietokantaobjektien määrälle sekä levytilalle	<= 50 CCU: Ilmainen <= 1000 CCU: $999 > 1000 CCU: $4999
Palvelimen tukemat skriptauskielet	Unityn tukemat kielet	.NET	.NET	Java, Python
Verkkoprotokollat	UDP	TCP, UDP, HTTP	TCP	TCP, UDP, HTTP

Kuromantasaus palvelinten välillä	Saatavilla erikseen	On	On	Ei
Interest managment	On	On	Ei	On
Web UI for server administration	Ei	Dashboard statis- tiikkojen katse- luun	On	On
Sisältääkö tietokantapalvelimen?	On (no-sql)	Ei	On (no-sql)	Ei (tukee ODBC/JDBC-yhteyksiä ulkoisin tietokantoihin
Tuki ulkoisille autentikointipalveluille?	Ei	Ei	On (Facebook, Kongregate)	Ei
Lobby- toiminnallisuus?	On	On	Esimerkkitoteutus	On
Rajapinnat ylläpitoa varten	Ei erillisiä rajapintoja, palvelinsovellukset Unity-ohjelmia → Käytössä Unityllä toteutetut ylläpitotoiminnot	WWW, telnet, Windows- sovellus	WWW	WWW
Unity- integraatio	Hyvä, uLinkin omat komponentit löytyvät komponenttivalikosta ja ovat helppokäyttöisiä.	Ok. Itse kirjasto ei toteuta editoriin toiminnallisuutta, mutta esimerkki- projekteista löytyvät komponentit ovat hyvin toteutettuja ja melko käyttökelpoisia sellaisenaankin.	Heikohko. Editoriin integroituja toimintoja ei ole.	Ei selvillä.
Muuta	Sekä clientti että palvelin ovat Unity-sovelluksia.		Tuote on tarkoitettu kevyiden selain-/mobiilipelien palvelimeksi.	

4 PROJEKTIN TOTEUTTAMINEN

Työn tavoitteena oli siis saada valittua järkevin vaihtoehto palvelimen muodostamiseksi mobiililaitteden välille. Vaihtoehtoina on käyttää joko mobiililaitetta itseään palvelimena tai muodostaa palvelin jollekulle ulkoiselle laitteelle tai pilvipalveluun.

Ulkoista vaihtoehtoa mietittäessä järkevintä olisi sijoittaa palvelin pilvipalveluun ja antaa palvelun ylläpitäjien hoitaa mm. tietoturvapuoli. Jos kapasiteettiä tarvittaisiin lisää, sitä olisi helposti saatavilla pilvipalvelun ylläpitäjiltä. Taulukossa 2 vertailluista tuotteista Player IO olisi siis ainoa vaihtoehto toteuttaa kyseinen palvelin.

Kun mietitään millaiseen käyttöön monen CCU:n palvelimia käytännössä tarvittaisiin, tulee ensimmäisenä mieleen massiiviset online roolipelit. Näissä peleissä samalla serverillä saattaa olla jopa tuhansia pelaajia yhtäaikaisesti. Vastaavasti pinempiin esimerkiksi kaksinkamppailupeleihin riittäisi, kun toinen käyttäjistä toimisi palvelimena, johon vastustaja sitten yhdistäisi. Siihen lisäksi asetettaisiin pieni tietokantapalvelin, joka ylläpitäisi kaksinkamppailuissa saatuja tilastoja ja näyttäisi ne maailmanlaajuisesti esimerkiksi internetin kautta. Kun mietittiin opinnäytetyön kokoa (5 op), päädyttiin vaihtoehtoon, jossa palvelin luodaan käyttäjälle mobiililaitteeseen ja pyrittiin sitä kautta saamaan laitteet yhdistettyä toisiinsa pelin sisälle. (Luku 3.1.)

Pelin ideaa ei haettu liian suurista kuvitelmista, vaan päädyttiin yksinkertaiseen ratkaisuun. Työn ideanahan oli saada ratkaistua, kuinka luodaan yhteys mobiililaitteiden välille eikä niinkään keskittyä hienojen pelijuonten ja efektien luomiseen. Ideaksi muodostui siis yksinkertainen yläperspektiivistä kuvattu kiihdytyspeli, jossa kaksi hahmoa kilpailee keskenään lähtöviivasta maaliviivaan. Hahmoina toimi kaksi palloa. Kun näyttöä naputetaan mahdollisimman nopeasti, niin hahmo kiihdyttää vauhtiaan ja voittajaksi selviytyy se kumpi on ennemmin maalissa.

4.1 Scriptit ja niiden käyttö

Unity-pelialustalla pelien toteuttaminen tapahtuu lisäämällä objekteihin skriptejä. Skripteillä on tarkoitus muokata objektien ja itse pelin tilaa. Ohjelmointikielinä on mahdollista käyttää C#-, JavaScript- ja Boo-ohjelmointikieliä. Pelien alustana toimii Mono-kirjasto, joka kääntää skriptit CLI-standardin mukaiseksi tavukoodiksi. Skripteillä voidaan myös toteuttaa editoriin uusia toimintoja. Monimutkaisia skriptejä varten kannattaa tehdä pienimuotoinen käyttöliittymä, jossa skriptien muokkaaminen ja tarkastelu helpottuvat huomattavasti. (8.)

4.2 Palvelimen liittäminen projektiin

Tässä luvussa avataan, kuinka itse pelipalvelimen asennus tapahtuu ja minkälaista koodausta siihen tarvitaan. Liitteenä on koko palvelimen alustus -skripti kommentoituna. Serverin luonti on toteutettu JavaScript-kielellä.

Kuvassa kaksi luodaan funktio palvelimen alustukselle. Kun halutaan luoda serveri, se täytyy ensiksi alustaa verkossa sekä rekisteröidä Unityn pääpalvelimelle. Alustus vaatii enimmäispelaajamäärän (tässä tapauksessa 1) sekä portin numeron (25002). Palvelimen rekisteröimiseksi, pelin nimi täytyy olla omaperäinen, jottei tulisi ongelmia muiden hankkeiden kanssa. Nimi voi olla käytännössä mikä tahansa. Tässä tapauksessa alustamme nimen gameName-nimiseen muuttujaan.

```
38 function startServer(){
39         Network.InitializeServer(1,25002,!Network.HavePublicAddress);
40         MasterServer.RegisterHost(gameName, stringToText + "'s game", "PVP Acceleration");
41 }
```

Kuva 2. Palvelimen alustus

Jos pelipalvelin on alustettu onnistuneesti, kutsutaan OnServerInitialized()-funktiota kuvan kolme mukaisesti. Lisätään tieto lokiin, jotta saataisiin siitä myös palautetta itsellemme. Lisäksi funktiossa luodaan uusi pelaaja.

```
106 function OnServerInitialized(){
107     spawnPlace = 0;
108     Debug.Log("Server initialized");
109     spawnPlayer();
110 }
```

Kuva 3. Palvelin alustettuna

Kuvan neljä mukaista OnMasteServerEvent()-funktiota kutsutaan, kun tapahtuu muutos pääpalvelimella ja siitä halutaan välittää tieto pelipalvelimen ylläpitäjälle ja asiakkaille. Nyt kysytään, onko palvelin rekisteröity, ja jos on, saadaan siitä palaute.

```
122 function OnMasterServerEvent(mse:MasterServerEvent){
123 if(mse == MasterServerEvent.RegistrationSucceeded){
124     Debug.Log("Registered Server");
125     }
126 }
```

Kuva 4. Rekisteröimisen tarkastus

Seuraavaksi tarvitaan jonkinlainen nappi tai painike, joka käynnistää palvelimen niin haluttaessa, tai vaihtoehtoisesti liittyy jo luodulle palvelimelle. Nappien luomiseksi käytetään OnGUI() funktiota. Napit halutaan nähdä vain jos ei ole jo liitytty jollekin palvelimelle. Kuvassa 5 luotiin erityinen kenttä, johon voidaan kirjoittaa pelaajan nimi ja tallennettiin se Unityssä olevaan yksinkertaiseen talletusominaisuuteen (PlayerPrefs). Sen jälkeen on palvelimen alustusnappi, joka antaa palautteen lokiin painiketta painettaessa sekä menee startServer() funktioon (kuva 2).

```
143 //GUI
144
145 var style : GUIStyle = new GUIStyle();
146 var style2 : GUIStyle = new GUIStyle();
147 var style3: GUIStyle = new GUIStyle();
148
149 function OnGUI(){
150
151     if(!Network.isClient && !Network.isServer){
152         stringToText = GUI.TextField (Rect(btnX * 1.5 + btnW, btnY , btnW*3,btnH*0.5), stringToText, 10);
153         PlayerPrefs.SetString("nickName", stringToText);
154         GUI.skin.textField.fontSize = 40;
155         //Nickname field, stored in stringToText variable and saved in a pesific PlayerPref.
156
157             if (GUI.Button(Rect(btnX,btnY,btnW,btnH),"Start\nServer",style)){
158             Debug.Log("Starting Server");
159             startServer();
160             }
161         //Start server button.
162
163
164             if (GUI.Button(Rect(btnX*0.85,btnY * 1.2 + btnH,btnW*1.2,btnH), "Refresh\nHosts", style)){
165             Debug.Log("Refreshing");
166             refreshHostList();
167             }
168             if(hostData){
169                 for(var i:int = 0; i <hostData.length; i++){
170                     if(GUI.Button(Rect(btnX * 1.5 + btnW, btnY*1.2 + (btnH * i), btnW*3,btnH*0.5), hostData[i].gameName, style)){
171                         Network.Connect(hostData[i]);
172                         Debug.Log("Connecting");
173                     }
174                 }
175             }else{
176             GUI.Button(Rect(btnX * 1.5 + btnW, btnY*1.2 + (btnH * i * 0.1 ), btnW*3,btnH*0.5), "Refresh for games", style3);
177             }
178         //Serverlist buttons
179
180
181     GUI.Button(Rect(btnX*2,btnY - btnH*1.5,btnW*3,btnH*0.5), "PVP Acceleration", style2);
182     //Game Name (made in button because it's easier to modificate).
183     }
184 }
```

Kuva 5. Painikkeiden luonti

Alustusnapin alapuolelle haluttiin tehdä palvelinlistan päivitysnappi (Refresh Hosts), jota painettaessa nappi kutsuu refreshHostList() funktiota. (Kuva 5.) Tämän funktion tarkoitus on päivittää pääpalvelimella olevat tähän peliin integroidut palvelimet. Kuvassa 6 olevan refreshing muuttujan mukaan erillisessä

Update()-funktiossa tapahtuu itse pelipalvelimien tallennus hostData-muuttujaan. HostData-funktion perusteella palvelimia asetetaan MasterServeriltä haettujen tietojen mukaan painikkeiksi ruudulle. Update()-funktiota käytetään tilanteissa, joissa halutaan jotakin tapahtuvan jokaisessa päivitysvälissä. Tässä funktiossa esimerkiksi luodaan usein pelaajan toiminnot. Kuvan seitsemän koodi on kirjoitettu Update()-funktion sisälle.

```
45 function refreshHostList(){
46         MasterServer.RequestHostList(gameName);
47         refreshing = true;
48         Debug.Log(MasterServer.PollHostList().Length);
49 }
```

Kuva 6. Pelipalvelinten päivitys

```
95  if(refreshing){
96     if(MasterServer.PollHostList().Length > 0){
97     refreshing = false;
98     Debug.Log(MasterServer.PollHostList().Length); // Log tells us how many servers we have online.
99     hostData = MasterServer.PollHostList(); // Saving Hostdata in variable.
100    }
101 }
```

Kuva 7. Jos refreshing muuttuja on totta

Tämän jälkeen, jos tietoja on (hostData), ne tulostetaan painikkeina allekkain. Jos palvelimia ei ole tai niitä ei ole päivitetty, näytetään ”Refresh for games”-tekstiä. Vielä lopussa luodaan pelin nimi, ”PVP Acceleration” erilliselle painikkeelle ruudun ylälaitaan. Tekstikentätkin on luotu painikkeina, koska niihin voi asettaa editorissa muokattavan tyylin (GUIStyle). Ne ovat myös helpommin muokattavissa erikokoisille näytöille, joita puhelimissa ja tableteissa on runsaasti. (Kuva 5.)

4.3 Spawnpoints

Kun halutaan ”synnyttää” uusi pelaaja pelinäkymän tiedettyyn kohtaan, helpoin ja kätevin tapa toteuttaa se on tehdä erillinen objekti, joka määrittää paikan ja jonka suhteen uusi hahmo synnytetään. Luotiin kaksi objektia, SpawnPoint1 ja SpawnPoint2. Sitten tarvitaan vain ehdot, jossa määritellään, kumpaa paikkaa käytetään. Ehtoina käytettiin joko palvelimen luojana olemista (OnServerIntialized()-funktio) tai palvelimen asiakkaana olemista (OnConnectedToServer()-funktio). Jos on palvelimen ylläpitäjä, syntyy SpawnPoint1:een ja jos tulee palvelimelle asiakkaana, syntyy SpawnPoint2:een.

```
98 // 2 different spawnpoints
99
100 function spawnPlayer(){
101 if (spawnPlace == 0){
102     Network.Instantiate(playerPrefab, spawnObject1.position,Quaternion.identity,0);
103     }
104 if (spawnPlace == 1){
105     Network.Instantiate(playerPrefab, spawnObject2.position,Quaternion.identity,0);
106     }
107 }
```

Kuva 8. Spawnpointtien luonti

```
123 function OnServerInitialized(){
124     spawnPlace = 0;
125     Debug.Log("Server initialized");
126     spawnPlayer();
127 }
128
129 function OnConnectedToServer() {
130     spawnPlace = 1;
131     spawnPlayer();
132 }
```

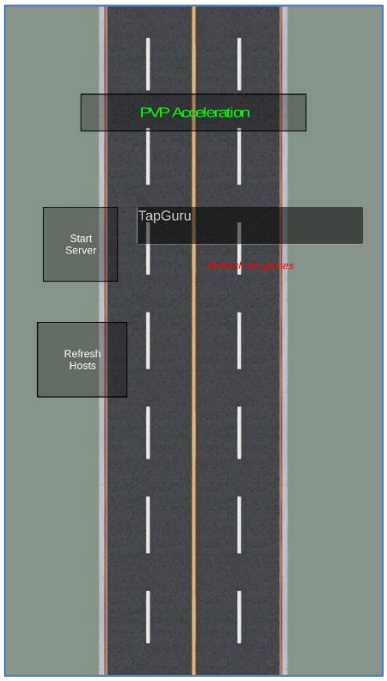
Kuva 9. OnServerIntialized() ja OnConcectedToServer()

5 PROJEKTI MOBIILILAITTEELLA

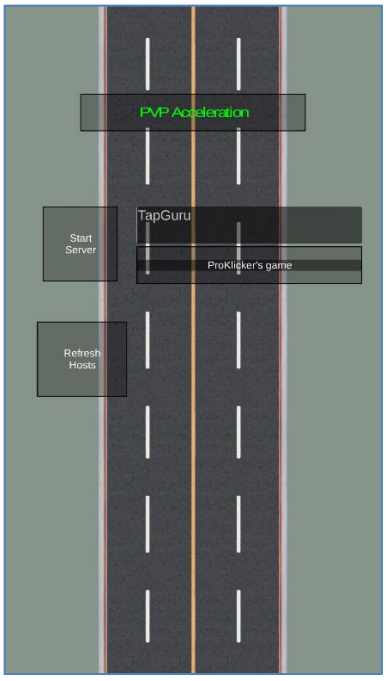
Projektin kokeileminen mobiililaitteella onnistui pienten näytön resoluutioihin liittyvien korjausten jälkeen moitteettomasti. Tilasynkronointi onnistuu, mutta pientä nykimistä liikkeissä on. Tähän tulisi käyttää ns. ennustusta, jonka avulla nykimistä voitaisiin hiukan laimentaa. Se ei kuitenkaan ollut projektin kannalta oleellista, joten se jätettiin myöhempään kehitystyöhön.

5.1 Pelinäkymä päävalikossa

Päävalikon näkymä on kuvan kymmenen ja kuvan yksitoista kaltainen. Kuvassa kymmenen ei ole vielä painettu ”Refresh Hosts” -nappia tai vaihtoehtoisesti master serverillä ei ole yhtään luotua palvelinta. Kuvassa yksitoista on päivityksen jälkeinen näkymä. Nimenkirjoituskentän alapuolelle on ilmestynyt yksi uusi pelipalvelin. Painiketta klikkaamalla liitytään palvelimelle.



Kuva 10. Ennen päivitystä

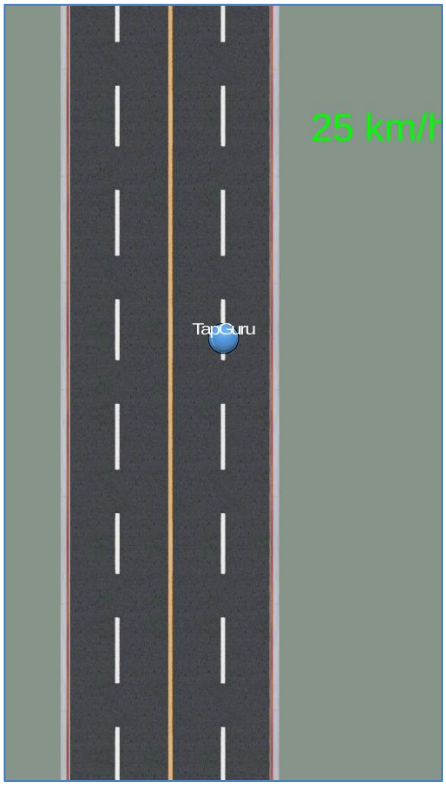


Kuva 11. Päivityksen jälkeen

5.2 Pelinäkymä pelitilanteessa

Kuvassa kaksitoista molemmat pelaajat ovat liittyneet peliin ja ”syntyneet” omille koodissa määritetyille paikoille. Toisen pelaajan liittyessä pelipalvelimelle alkaa lähtölaskenta. Kuvassa 13 näkyy kuinka TapGuru-niminen pelaaja on päässyt karkumatkalle 25 kilometrin kuvitteellisella tuntinopeudella.

Kuva 12. Pelaajat palvelimella



Kuva 13. TapGuru johdossa

6 YHTEENVETO

Koska opinnäytetyön tavoitteena oli selventää palvelimen ylläpidossa käsiteltävää koodausta, työssä ei keskitytty miten sen varsinainen pelimoottori rakennettiin. Työn aikana avattiin, mikä on palvelin ja miten pelipalvelin eroaa muista palvelimista. Tutkittiin eri vaihtoehtoja palvelimen luomiseksi Unitylle ja valittiin työtä varten järkevin vaihtoehto. Opinnäytetyössä saatiin toteutettua esimerkkiprojekti Unityllä ja liitettyä palvelin siihen. Projekti ja palvelin toimi mobiililaitteilla hyvin. Suuria ongelmia palvelimen luonnissa mobiililaitteelle ei ollut. Palvelimen luominen on tehty suhteellisen yksinkertaiseksi Unity3D:n avulla.

LÄHTEET

1. Ei leimoja. Saatavissa: http://www.tieke.fi/pages/viewpage.action?pageId=3441286 Hakupäivä: 8.4.2015.

2. Hakala, Anni. 2014. Korkeasti käytettävän pilvipalvelun toteuttaminen. Saatavissa: http://www.theseus.fi/bitstream/handle/10024/85283/Hakala_Anni.pdf?sequence=2 Hakupäivä: 8.4.2015.

3. Van Winkle, William. 2013. Mobile Servers: Get a Handle on Your Server. Saatavissa: http://www.tomsitpro.com/articles/mobile-server-eurocomm-getac-disaster-recovery-fault-tolerant,2-594.html Hakupäivä: 8.4.2015.

4. Saatavissa: http://mashable.com/2011/11/16/mobile-app-cloud-servers/ Hakupäivä: 8.4.2015.

5. Saatavissa: http://azure.microsoft.com/en-us/services/app-service/mobile/ Hakupäivä: 8.4.2015.

6. Fiedler, Glenn. 2010. What every programmer needs to know about game networking. Saatavissa: http://gafferongames.com/networking-for-gameprogrammers/what-every-programmer-needs-to-know-about-gamenetworking/ Hakupäivä: 8.4.2015.

7. Bernier, Yahn W. 2001. Latency Compensating Methods in Client/Server Ingame Protocol Design and Optimization. Saatavissa: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_i n_Client/Server_In-game_Protocol_Design_and_Optimization. linkki: Client Side Prediction. Hakupäivä: 8.4.2015.

8. Halmetoja, Heikki. 2014. https://www.theseus.fi/bitstream/handle/10024/70618/Heikki_Halmetoja.pdf?sequence=1 Hakupäivä: 10.4.2015.

9. Gambetta, Gabriel. Fast-paced Multiplayer (Part 1): Introduction. Saatavissa: http://www.gabrielgambetta.com/fpm1.html Hakupäivä: 10.4.2015.

10. Unity Fast Facts. 2013. Unity Technologies. Saatavissa: http://unity3d.com/company/public-relations Hakupäivä: 10.4.2015.

11. Unity Asset Workflow. 2013. Unity Technologies. Saatavissa: http://unity3d.com/unity/workflow/asset-workflow Hakupäivä: 10.4.2015.

12. State Synchronization Details. 2013. Unity Technologies. Saatavissa: http://docs.unity3d.com/Documentation/Components/netStateSynchronization.html Hakupäivä: 10.4.2015.

13. Networking Elements in Unity. 2013. Unity Technologies. Saatavissa: http://docs.unity3d.com/Documentation/Components/netUnityNetworkElements.html Hakupäivä: 10.4.2015.

14. Network Views. 2013. Unity Technologies. Saatavissa: http://docs.unity3d.com/Documentation/Components/net-NetworkView.html Hakupäivä: 10.4.2015.

15. RPC Details. 2013. Unity Technologies. Saatavissa: http://docs.unity3d.com/Documentation/Components/net-RPCDetails.html Hakupäivä: 10.4.2015.

```
var players : int;

var playerPrefab:GameObject;
var spawnObject1:Transform;
var spawnObject2:Transform;

var gameName : String = "PVP_Acceleration";

public static var spawnPlace : int;

private var refreshing:boolean;
private var hostData:HostData[];

private var btnX:float;
private var btnY:float;
private var btnW:float;
private var btnH:float;


var stringToText : String;


var intervalTimer=0.0; var interval=0.0;

//--------------------------------------------------------------------------
//When the scene starts, we set and get these.

function Start(){
stringToText = "Give Name";
interval = 3;
btnX = Screen.width * 0.1f;
btnY = Screen.height * 0.3f;
btnW = Screen.width * 0.2f;
btnH = Screen.width * 0.2f;
stringToText = PlayerPrefs.GetString("nickName");
}

//--------------------------------------------------------------------------
//These functions are connected also to GUIButtons in OnGUI Fuction
function startServer(){
        Network.InitializeServer(1,25002,!Network.HavePublicAddress);
        MasterServer.RegisterHost(gameName, stringToText + "'s game", "PVP Acceleration");
}

function refreshHostList(){
        MasterServer.RequestHostList(gameName);
        refreshing = true;
        Debug.Log(MasterServer.PollHostList().Length);
}

```

```
50
51
52 //--------------------------------------------------------------------------------
53 //Autorefresh function
54 function autoRefresh(){
55         refreshHostList();
56         Debug.Log("Refreshing");
57                     if(hostData){
58                 for(var i:int = 0; i <hostData.length; i++){
59                     if(GUI.Button(Rect(btnX * 1.5 + btnW, btnY*1.2 + (btnH*0.5 * i), btnW*3,btnH*0.5), hostData[i].gameName, style)){
60                         Network.Connect(hostData[i]);
61                     }
62                 }
63             }
64
65 }
66
67
68 //--------------------------------------------------------------------------------
69
70
71 function Update(){
72
73 players = Network.connections.Length; //The amount of clients on the server
74
75 //Shutting server and reloading scene if opponent leaves. (Make your own if section, this wont work with you)
76 if (CDown.paalla == 0 && players == 0){
77             Application.LoadLevel(0);
78             Network.Disconnect();
79             MasterServer.UnregisterHost();
80 }
81
82 //Making autorefresh in Update function
83 if(!Network.isClient && !Network.isServer){
84     if(intervalTimer >= 3){
85       autoRefresh();
86        intervalTimer = 0;
87       }
88     if(intervalTimer >= 0){
89        intervalTimer += 0.02;
90        interval = 3;
91        }
92 }
93 //----
94 // Asking if masterserver has any servers online (check refreshHostList() Function).
95  if(refreshing){
96     if(MasterServer.PollHostList().Length > 0){
97     refreshing = false;
98     Debug.Log(MasterServer.PollHostList().Length); // Log tells us how many servers we have online.
99     hostData = MasterServer.PollHostList(); // Saving Hostdata in variable.
100     }
101  }
102 }
```

```
105
106 //--------------------------------------------------------------------------------
107 // 2 different spawnpoints
108
109 function spawnPlayer(){
110 if (spawnPlace == 0){
111     Network.Instantiate(playerPrefab, spawnObject1.position,Quaternion.identity,0);
112     }
113 if (spawnPlace == 1){
114     Network.Instantiate(playerPrefab, spawnObject2.position,Quaternion.identity,0);
115     }
116 }
117
118
119
120 //--------------------------------------------------------------------------------
121 //Messages
122
123 function OnServerInitialized(){
124     spawnPlace = 0;
125     Debug.Log("Server initialized");
126     spawnPlayer();
127 }
128
129 function OnConnectedToServer() {
130     spawnPlace = 1;
131     spawnPlayer();
132 }
133
134
135 function OnMasterServerEvent(mse:MasterServerEvent){
136 if(mse == MasterServerEvent.RegistrationSucceeded){
137     Debug.Log("Registered Server");
138     }
139 }
140
141
```

```
142 //--------------------------------------------------------------------------------
143 //GUI
144
145 var style : GUIStyle = new GUIStyle();
146 var style2 : GUIStyle = new GUIStyle();
147 var style3: GUIStyle = new GUIStyle();
148
149 function OnGUI(){
150
151     if(!Network.isClient && !Network.isServer){
152         stringToText = GUI.TextField (Rect(btnX * 1.5 + btnW, btnY , btnW*3,btnH*0.5), stringToText, 10);
153         PlayerPrefs.SetString("nickName", stringToText);
154         GUI.skin.textField.fontSize = 40;
155         //Nickname field, stored in stringToText variable and saved in a pesific PlayerPref.
156
157             if (GUI.Button(Rect(btnX,btnY,btnW,btnH),"Start\nServer",style)){
158             Debug.Log("Starting Server");
159             startServer();
160             }
161         //Start server button.
162
163
164             if (GUI.Button(Rect(btnX*0.85,btnY * 1.2 + btnH,btnW*1.2,btnH), "Refresh\nHosts", style)){
165             Debug.Log("Refreshing");
166             refreshHostList();
167             }
168             if(hostData){
169                 for(var i:int = 0; i <hostData.length; i++){
170                     if(GUI.Button(Rect(btnX * 1.5 + btnW, btnY*1.2 + (btnH * i), btnW*3,btnH*0.5), hostData[i].gameName, style)){
171                         Network.Connect(hostData[i]);
172                         Debug.Log("Connecting");
173                     }
174                 }
175             }else{
176             GUI.Button(Rect(btnX * 1.5 + btnW, btnY*1.2 + (btnH * i * 0.1 ), btnW*3,btnH*0.5), "Refresh for games", style3);
177             }
178         //Serverlist buttons
179
180
181     GUI.Button(Rect(btnX*2,btnY - btnH*1.5,btnW*3,btnH*0.5), "PVP Acceleration", style2);
182     //Game Name (made in button because it's easier to modificate).
183     }
184 }
```

LIITE 2

OULUN AMMATTIKORKEAKOULU

Henri Viljamaa

PELIN SUUNNITTELU JA OPTIMOINTI MOBIILIALUSTALLE UNITY3D:LLÄ

Osaopinnäytetyöt 2 + 3

LIITE 2

PELIN SUUNNITTELU JA OPTIMOINTI MOBIILIALUSTALLE UNITY3D:LLÄ

Henri Viljamaa
Osaopinnäytetyöt 2 + 3
Kevät 2017
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, hyvinvointiteknologian suuntautumisvaihtoehto

---

Tekijä: Henri Viljamaa
Opinnäytetyön nimi: Pelin suunnittelu ja optimointi mobiilialustalle Unity3D:llä
Työn ohjaaja: Jukka Jauhiainen
Työn valmistumislukukausi ja -vuosi: Kevät 2017 Sivumäärä: 36 sivua

---

Työssä kerrotaan peliprojektista *Tales From The North* ja siitä, kuinka sitä on suunniteltu ja optimoitu mobiililaitteita varten. Samalla on haettu tietoa ja yleisiä käytänteitä suunnittelusta ja optimoinnista.

Projekti toteutettiin Unity3D-alustalla ja mallintamisessa käytettiin Blender-mallinnusohjelmaa. 2D-grafiikoiden tekemisessä on käytetty Krita-nimistä kuvankäsittelyohjelmaa.

Työssä käydään läpi kokonaisuutena pelin rakentamista, suunnittelua ja optimointia. Tarkoituksena oli löytää ne menetelmät, miten 3D-pelin ohjelmointi ja grafiikka tehdään mahdollisimman tehokkaasti. Työn lähtökohtana ovat 3D-pelit mobiililaitteille, mutta samat säännöt pätevät myös muille pelialustoille, kuten pelikonsoleille ja tietokoneille.

Tuloksena saatiin selville ne seikat, kuinka isosta pelistä saadaan tiivistettyä mahdollisimman kevyt kokonaisuus, jota tämänhetkisellä keskivertokuluttajan mobiililaitteella voi pelata. Opinnäytetyön aikana julkaistiin myös testiversio 3D-mobiilipelistä *Tales From The North.*

Asiasanat: Unity, ohjelmistosuunnittelu, 2D-grafiikka, 3D-grafiikka, optimointi

SISÄLLYS

1 JOHDANTO

Projektin lähtökohtana on tietotekniikan opiskelijoiden unelma omasta pelialan yrityksestä, jossa voisi toteuttaa omaa kädenjälkeänsä itselle mieluisessa toimintaympäristössä ja työolosuhteissa mielekkäiden projektien parissa. Projektin taustalle on perustettu toiminimi Nobles. Toiminimi perustettiin, jotta projektille saadaan jokin konkreettinen yritys, jota voidaan hyödyntää koulun harjoitteluissa sekä projektitöissä. Haaveena on saada lyötyä suurelle yleisölle läpi jokin projekteista ja vaihtaa sen jälkeen yhtiömuodoksi osakeyhtiö. Pelit tullaan julkaisemaan nimellä Nobles GameStudio.

Opinnäytetyössä paneudutaan siihen, miten 3D-peliä kannattaa suunnitella ja optimoida jo aivan projektin alkuvaiheessa aina läpi koko projektin toteuttamisen. Työssä käytettävät menetelmät on suunnattu mobiililaitteille. Työssä kerrotaan yleisesti projektista *Tales From The North*, jonka pohjalta tämä opinnäytetyö on rakennettu.

Opinnäytetyössä tarkastellaan sitä, kuinka tällaista isompaa peliprojektia kannattaa lähteä suunnittelemaan yleisten kriteerien mukaan ja kuinka sitä lähdettiin toteuttamaan *Tales From The North* -projektissa. Suunnitteluvaihetta pilkotaan aina kokonaisuudesta pienempiin yksityiskohtiin, kuten koodin rakentamiseen ja grafiikan suunnitteluun.

Optimointi on tärkeä osa pelin tekemistä. Sillä varmistetaan pelin sujuvuus ja mukava kokemus loppukäyttäjälle. Karkeasti sanottuna optimoinnilla haetaan ennen kaikkea sitä, että saadaan mahdollisimman hyvä päivitysnopeus (FPS, frames per second) tehollisestikin vaativissa kohdissa peliä. Työssä paneudutaan myös siihen, miten projektissa *Tales From The North* on toteutettu optimointia pelin tekemisen aikana ja viimeistelyvaiheessa. Lisäksi selvitetään, kuinka muissa projekteissa optimointia on suoritettu, sekä haetaan yleisiä käytänteitä optimoinnille.

2 PELISUUNNITTELU

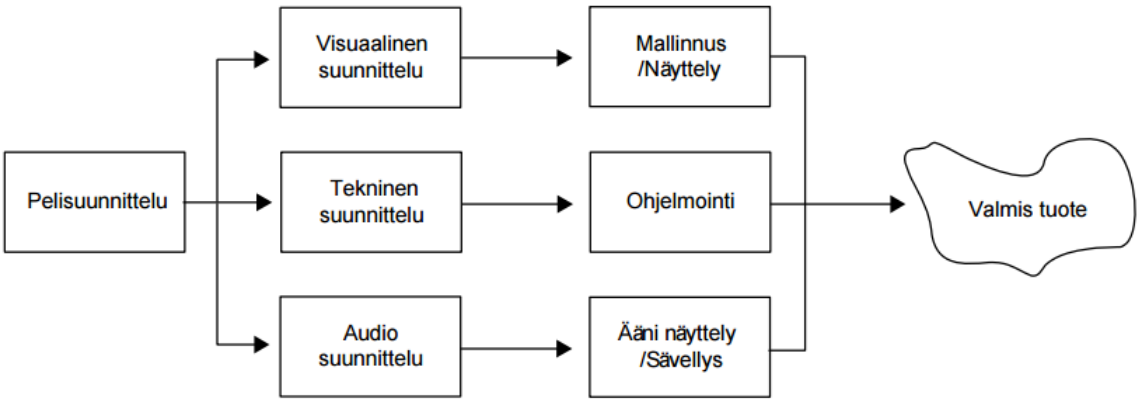
Pelien kehittäminen on muuttunut muutaman viimeisen vuosikymmenen aikana radikaalisti. Aikaisemmin tarvittiin runsaasti resursseja ja laiteläheistä osaamista pelinkehittäjäorganisaatiolta, jos haluttiin tehdä peli, joka saavuttaa pelimarkkinoilla suuren suosion ja laajan levikin. Pelimoottorien ja fysiikoiden koodaaminen täytyi aloittaa käytännössä aivan alusta ilman suurempia valmiita pohjia, ellei organisaatio ollut jo valmiiksi työstänyt pelimoottorin, johon uutta ruvettiin rakentamaan.

Nykyisin laadukkaan pelin voi tuottaa pienelläkin innokkaalla ja ammattitaitoisella ryhmällä. Pelintekemisen on mullistanut valmiiden, suurimmaksi osaksi ilmaisten pelialustojen tuleminen markkinoille ja yksityisten käyttäjien saataville. Valmiita pelinkehitysympäristöjä hyödyntämällä pienet yritykset ja yksittäiset henkilötkin voivat tuottaa läpimurtoja suurelle yleisölle peleillään.

Unity3D (1) on tällä hetkellä yksi yleisimmin käytössä olevista pelialustoista. Unity3D:ssä on laadukkaiden 2D- ja 3D-fysiikoiden lisäksi upeat grafiikka- ja erikoistehoste -ominaisuudet peleille. Sen lisäksi, että ohjelmassa saa valmiiksi mukana laajat valikoimat tehosteita ja ominaisuuksia, saa lisäosia myös ladattua niin fysiikka- kuin grafiikkapuolellekin Asset Storesta (2). Lisäosissa on käyttäjien ja Unityn oman kehittäjäryhmän valmistamia ominaisuuksia, joissa on sekä ilmaisia että maksullisia ohjelmia, jotka helpottavat pelin työstämistä.

2.1 Pelisuunnittelun yleiset käytänteet

Nykyään varsinkin tietokone- ja konsolipelien tuotanto muistuttaa yhä enemmän elokuvatuotantoa kuin ohjelmistotuotantoa. Pelit ovat vuorovaikutteisia ja sisältävät pelin edetessä videoita, joissa monesti tarvitaan oikeita näyttelijöitä. Audiovisuaalinen suunnittelu vaatii resursseja pelistä riippuen, jopa huomattavasti enemmän kuin tekninen suunnittelu ja siihen liittyvä ohjelmointi, kuten kuvan 1 suunnitelmasta voidaan nähdä. Tietokonepelien sekä konsolipelien elinkaaresta siis varsin suuri osa on muuta kuin koodaamista. (3.)



KUVA 1. Esimerkki pelisuunnittelusta (3)

Pelin suunnittelu on täytynyt aloittaa ideoimalla uusi peli-idea, ellei tehdä valmiille pelille jatko-osaa. Peli-idean voi saada lähes mistä vain. Usein pelin idea saadaan inspiroitumalla muista peleistä, joista saadun kokemuksen perusteella yhdistellään uusi peli, johon lisätään omia visioita. Ideoita voi saada myös aivan arkisesta elämästä, elokuvista, kirjallisuudesta tai esimerkiksi urheilusta. Kun pelaamiansa pelejä pelaa analyyttisesti, saa monesti inspiraation uusiin omiin peleihin. (3.)

Kohderyhmän valinta on tärkeää. Liian suuri kohderyhmä on usein huono asia. Kohderyhmässä käytetään yleensä karkeaa luokittelua ”HC-pelaajien” (hardcore) ja ”sunnuntaipelaajien” välillä. HC-pelaajat käyttävät paljon aikaa pelien pelaamiseen ja voivat viettää aikaa pelissä saman ominaisuuden tai kohdan kanssa huomattavan kauan. Nämä pelaajat ovat valmiita opettelemaan monimutkaisenkin käyttöliittymän päästäkseen sisälle pelin vaikeuksiin ja ominaisuuksiin. Tähän kohderyhmään kuuluvat ne, jotka keskustelevat peleistä foorumeilla ja uppoutuvat muuallakin yhteisöissä pelimaailman saloihin syvästi. Sunnuntaipelaajan taas karkottaa liian vaikea käyttöliittymä ja he kyllästyvät helposti, mikäli peli on jossakin kohdassa liian vaikea. Nämä käyttäjät eivät uhraa päivästään paljoa aikaa pelien pelaamiseen. Sunnuntaipelaajat eivät useinkaan osaa arvostaa pelin teknisyyttä, vaan heille pelattavuus on tärkeämpi ominaisuus. (3.)

Mobiilipuolella suomalaiset pelifirmat ovat onnistuneet lyömään läpi muutamia pelejä suurelle yleisölle. Sunnuntaipelaajille suunnatut Rovio Entertainmentin

Angry Birds (4) sekä Fingersoftin *Hill Climb Racing* (5) ovat hyviä esimerkkejä. Supercellin *Clash of Clans* (6) on taas saanut koukutettua HC-pelaajiakin pelaamaan peliä tosissaan, vaikkakin pelin käyttöliittymä miellyttää sunnuntaipelaajiakin.

Pelin tyypillä (genrellä) on suuri vaikutus kohderyhmään. Idea toimii samalla tavalla kuin elokuvissakin. Esimerkiksi toimintapeleillä on omat käyttäjänsä, jotka pitävät nopeatempoisista, usein pelaajan motoriikkaan ja koordinaatioon perustuvista haasteista. Juoni on usein pienemmässä roolissa, mutta visuaalinen näyttävyys on sitäkin tärkeämpi. Genreen kuuluvat esimerkiksi urheilupelit, autopelit, pelihahmon silmistä kuvatut FPS-pelit (First Person Shooter) ja simulaattorit. Roolipeleissä taas pelaaja kehittää hahmoaan erilaisten pelimaailmaan perustuvien haasteiden kautta. Roolipeleissä suuri merkitys on kommunikoinnilla joko muiden pelaajien tai pelin sisäisten hahmojen kanssa, kun taas visuaalisuus ja pelimekaniikka eivät ole niin tärkeitä kuin toimintapeleissä. (3.)

Pelaajat keskittyvät tietyn genren peleihin syystä. He haluavat peliltä joko hyvän juonen, hienoa grafiikkaa, paljon räiskintää, taktisia haasteita tai muita vastaavia ominaisuuksia. Toisaalta pelaajat taas haluavat välttää tylsiä juonen seuraamisia, aivotonta räiskintää tai liian monimutkaisia käyttöliittymiä. Tässä voidaan taas verrata elokuviin: katsojalla on selkeä kuva siitä, mitä hän kokee, kun katsoo tiettyyn genreen kuuluvan elokuvan. (3.)

2.2 Ohjelmistosuunnittelu

Ohjelmointi on muuttunut vuosikymmenien saatossa. Oleellinen ero menneeseen on se, että ohjelmien testaaminen voidaan suorittaa usein saman tien, kun koodi on saatu valmiiksi. Tästä syystä monet ohjelmointiin liittyvät metoditkin ovat muuttuneet.

Joskus ohjelmointi on vaivatonta ja vaikeatkin ongelmat ratkeavat mukavasti. Koodaaminen sujuu kuin ajatus ja se on helppoa sekä hauskaa. Tilanne on myöskin hyvin usein aivan päinvastainen. Ohjelmointi ei luonnistu laisinkaan, vaan koodaamisen kanssa joutuu taistelemaan. On paikkoja, joita joutuu koodaamaan moneen kertaan uudestaan, ennen kuin löytää itseään tyydyttävän ratkaisun.

Selkeää ja tehokkaasti toimivaa koodia ei siltikään saatu luotua. Jotta ohjelmoija välttyisi näiltä ikäviltä tilanteilta, jossa koodista on tullut sotkuisen näköistä ja tehotonta, on ohjelmistosuunnittelulla tärkeä rooli. (7.)

2.3 Ohjelmistosuunnittelun käytänteet

Kun tuotetaan tietokoneohjelmistoja tai ylipäätään ohjelmia, käytetään työnteon ja työnjohdon menetelmistä yleisnimitystä ohjelmistotuotanto. Ohjelmistotuotanto kattaa kaiken ohjelmistojen valmistukseen liittyvän prosessienhallinnan sekä niihin liittyvät valmistamisen menetelmät. Siihen kuuluu siis periaatteessa mikä tahansa toiminta, joka tähtää ohjelmistojen valmistukseen. Tarve valmistukseen tulee yleensä asiakkaalta tai vaihtoehtoisesti ohjelman tuottaja päättelee tarpeen olevan olemassa kyseiselle ohjelmistolle. (8.)

Ohjelmistosuunnittelu koostuu kahdesta eri vaiheesta, toiminnallisesta määrittelystä sekä teknisestä määrittelystä. Ohjelmoinnin kannalta toiminnallinen määrittely on se tärkeä osa. Toiminnallisessa määrittelyssä kuvataan kaikki toiminnot sekä liitännät, jota ohjelma tuottaa järjestelmän ulkopuolelle. Vaiheesta tehdään määrittelydokumentti, joka kuvaa kaikkea, mitä järjestelmällä voi tehdä sekä miten käyttäjä voi niitä hallita. Se ei kuvaa sitä, miten ne tulee toteuttaa. Esimerkiksi tekstinkäsittelyohjelmassa se kuvaisi sen eri näytöt ja tasot, valikot ja niissä olevat asetukset sekä käyttöliittymän muut kontrollerit. (8.)

Tyypillinen ongelma ohjelmistoalalla on se, että määrittelydokumentteja ei jakseta kirjoittaa. Jonkinlainen analyysi siitä, mitä ohjelma pitää sisällään, saatetaan tehdä, mutta sen jälkeen aletaankin heti koodata itse ohjelmaa. Ajatellaan, että dokumentin kirjoittamiseen tuhlattava aika käytetään paremmin hyödyksi koodaamisessa. Asia on kuitenkin juuri päinvastoin. Asioita on paljon helpompi muuttaa kertovasta tekstistä, kuin että lähdetään etsimään muutettavia kohtia koodista. Varsinkin isoista ohjelmista muodostuu rakenteellisesti hyvin vaikeaselkoisia ja virhealttiita rakennelmia, jos dokumenttia ei tehdä. (8.)

Suunnittelun hallinta helpottuu, jos ohjelmisto jaetaan erilaisiin kerroksiin. MVC-malli (Model View Controller) (8) on yksi tunnetuimmista malleista, jossa käyttöliittymä erotetaan muista osista. MVC-arkkitehtuuria käytetään etenkin graafisten

käyttöliittymien suunnittelussa ja ohjelmoinnissa. Mallin etuna on se, että käyttöliittymän tai muiden osien muuttaminen on helpompaa, kun muutokset tulevat tiettyyn kohtaan jo suunniteltua ohjelmaa. Mallin haittapuolena voi olla tietokannan muuttuminen raskaaksi, jos suunnittelua ei ole toteutettu huolella. Kolmitasomalli on toinen tunnettu malli, jossa tasoina ovat käyttöliittymä, liiketoimintakerros ja tietokantakerros. Hyvin toteutettuna kolmitasomalli voi olla järkevämpi vaihtoehto, varsinkin suurissa ohjelmistoissa. (8.)

2.4 Mobiililaitteiden huomiointi suunnittelussa

2.4.1 Näyttö

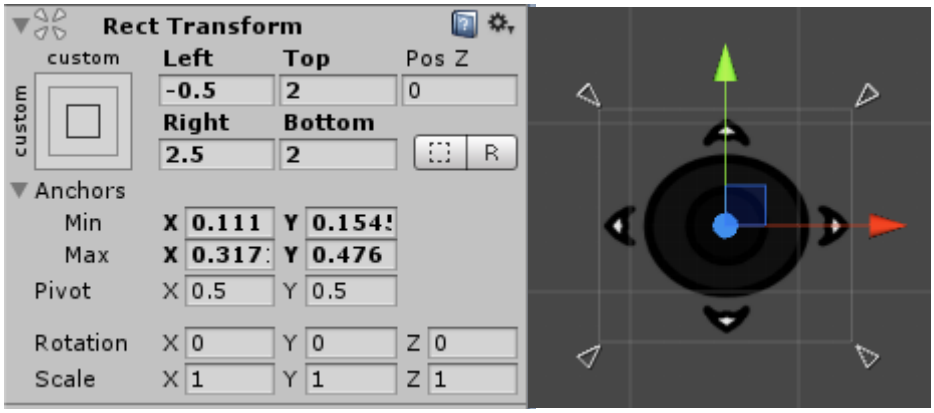
Mobiililaitteiden tekniset ominaisuudet vaihtelevat runsaasti. Pelikehityksen kannalta yksi merkittävimmistä ominaisuuksista on näytön koko ja samalla näytön resoluutio eli erottelukyky. Käytännössä resoluutio tarkoittaa sitä, kuinka paljon pikseleitä tuumaa kohden näytössä on. Koko taas vaihtelee aina pienistä 3 tuuman näytöistä isoihin tabletteihin, joissa voi olla yli 13 tuumankin näyttöjä. Esimerkiksi Apple iPhone 4s -puhelimen 3,5 tuuman näytössä pikseleitä on 960 x 640, mikä tarkoittaa 326 pikseliä tuumalle. Tyypillisissä pöytäkoneen näytöissä pikseleitä tuumaa kohti on vain 96. (9.)

Suunnittelun ja koodauksen kannalta näytön koko on merkittävä tekijä. Näyttöjä ja niiden tarkkuuksia (resoluutioita) on niin monta erilaista, että peliä ei voi suunnitella pelkästään tietyille tarkkuuksille. Huomioon täytyy siis ottaa koodauksessa ja suunnittelussa näytön korkeus ja leveys. Käytännössä kaikki käyttäjälle näkyvät painikkeet, ohjaimet ja pysyvästi paikallaan olevat kuvakkeet, jotka sijoitetaan pelinäkymän päällä olevalle UI-tasolle (user interface -taso), tulee skaalata näytön leveyden ja korkeuden suhteen. UI-taso on paikallaan pysyvä 2D-taso, joka tulee itse pelinäkymän päälle.

Unityssä näytön skaalaus on uusimmassa 5-versiossa tehty kehittäjälle mukavaksi ja suhteellisen helpoksi. Aiemmissa versioissa painikkeiden sijainti ja skaalaus täytyi toteuttaa täysin koodaamalla ohjelmakoodeissa (skripteissä), jos sen halusi saada mukautumaan eri näytöille UI-tasossa. Uudessa versiossa tilalle on

tullut uusi menetelmä (Canvas) (10), jossa ohjelma luo abstraktiin 2D-tilaan uuden UI-tason, joka skaalautuu aina näytön mukaan automaattisesti. Nykyisessä versiossa käyttäjän tarvitsee vain lisätä Canvas-objektin alle omat painikkeensa. Kaikkien käyttöliittymän elementtien on oltava alistettuja pääelementin alle, jossa on Canvas-komponentti liitteenä. (11.)

Käytännössä painikkeiden skaalaus tapahtuu Unityssä siten, että lisätään uusi elementti, esimerkiksi painike sen pääobjektin alle, jossa on Canvas liitettynä. Sen jälkeen painiketta voi venyttää ja siirtää editorin työkalulla. Painikkeiden siirtäminen ja venyttäminen haluttuun kokoon eivät kuitenkaan vielä riitä, jotta ne saataisiin skaalautumaan kaiken kokoisille näytöille. Kun painikkeen valitsee aktiiviseksi, avautuu Unityn käyttöliittymään sen kuvan 2 kaltainen Rect Transform -työkalu (laatikon sijainti- ja skaalaustiedot). Sen alavalikoista löytyy kohta ankkurit (Anchors), jossa valitaan tai säädetään painikkeille haluttu käyttäytymisen malli. Jotta kuvan tai painikkeen saa skaalautumaan kaikille näytöille, täytyy ankkurit asettaa UI-tasossa kuvan 2 kaltaisesti painikkeiden äärireunojen kulmiin.



KUVA 2. Rect Transform ja Anchors

2.4.2 Suorituskyky

Toinen huomioon otettava tekijä nimenomaan peliä mobiililaitteille suunniteltaessa on niiden suorituskyvyn rajallisuus. Vaikka mobiililaitteiden onkin povattu muutamien vuosien saatossa ohittavan suorituskyvyllään PS3- ja Xbox360-pelikonsolit, tosiasia on se, että vielä pitkään aikaan ei suurella yleisöllä tule olemaan sellaisia laitteita taskuissaan. (12.)

Tässä kohtaa tullaankin tärkeään seikkaan suunnittelun kannalta. Haasteena on saada reaaliaikainen ja vuorovaikutteinen peli, jossa on paljon tekoälyä ja objekteja sisältäviä pelinäkymiä, toimimaan keskitason älypuhelimilla. Monet peliyritykset ovat päätyneet ratkaisuun, jossa peleistä tehdään vuoropohjaisia, jolloin laitteelle ei tule mittavaa kuormaa. Toisaalta taas voidaan karsia rasitusta vähentämällä objektien määrää tai optimoimalla koodia siten, että vain kaikista välttämättömin on aina yhtä aikaa toiminnassa.

Esimerkkinä on kuvan 3 *Magic Legion* (13), jossa on saatu yhdistettyä 3D-maailma näyttävillä HD-tason (High Definition) grafiikoilla mielenkiintoiseen reaaliaikaiseen taisteluun. Pelissä ei ohjata itse hahmoja, vaan keskitytään painamaan taitopainikkeita, joilla pelihahmot saadaan tekemään haluttuja taisteluliikkeitä. Pelissä optimointia on toteutettu siten, että itse taistelukohtaukset ovat pieniä ja sisältävät vain kaikkein välttämättömimmän graafisen toteutuksen, mutta säilyttävät silti visuaalisen näyttävyyden. Lisäksi kerrallaan yhdessä pelinäkymässä ei ole liiallista määrää tekoälyä, vaan se on optimoitu siten, että tekoäly toimii pienimmällä mahdollisella koodimäärällä yhtäaikaisesti. Kun koodi saadaan optimoitua tehokkaaksi, voidaan pelinäkymässä pitää jopa yli 10:tä 3D-hahmoa, jotka sisältävät jonkinasteista tekoälyä. Miellyttävä graafinen toteutus, hienot erikoistehosteet ja animaatiot sekä käyttäjäystävällinen käyttöliittymä luovat pelikokemuksesta ainutlaatuisen.

KUVA 3. Magic Legion

3 PELIN OPTIMOINTI

Pelin optimoinnilla haetaan mahdollisimman sujuvaa ja tehokasta pelikokemusta loppukäyttäjälle, tinkimättä kuitenkaan liiaksi audiovisuaalisesta näyttävyydestä. Ohjelmoijan kannalta tarkoituksena on saada mahdollisimman hyvä näytön ja pelimoottorin päivitysnopeus pelille kaikissa sen kohdissa. Päivitysnopeutta voidaan parantaa jokaisella pelin osa-alueella. Ohjelmakoodin tulisi olla sujuvaa ja tehokasta, grafiikan pitäisi olla hyvin pakattua ja tehokkaasti käytettyä, sekä pitäisi tietää, kuinka käytössä olevalla pelinteko alustalla saadaan tehonkulutusta vähennettyä optimaaliseksi.

Kuten suunnitteluvaiheessa käytiin läpi, ohjelmakoodin optimoinnin kannalta olennaisinta on sen hyvä suunnittelu. Jokaiseen projektiin tulee omanlaisensa skriptit ja ne tulee suunnitella sitä käyttöä varten optimaaliseksi. Myöhemmässä vaiheessa käydään kuitenkin läpi sitä, kuinka koodia toteutettiin projektissa *Tales From The North*.

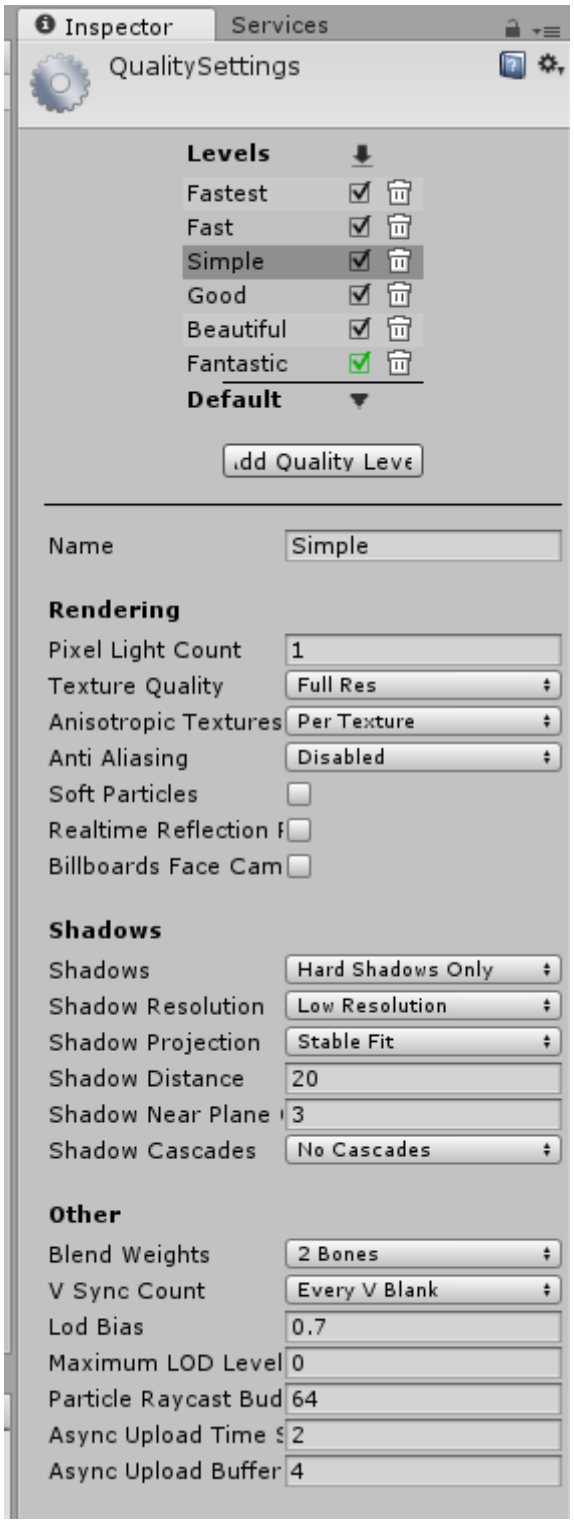
3.1 Grafiikan optimointi Unityssä

Unityssä tehonkulutus ja sen seuraaminen on tehty mukavaksi. Editorista löytyy painike, jonka alle on laitettu suorituskykyyn liittyvät tilastotiedot. Nämä tiedot päivittyvät reaaliaikaisesti, joten pelintekijä voi seurata, kuinka paljon mikäkin paikka pelissä syö tehoja. (Kuva 4)

KUVA 4. Statistiikka-ikkuna Unityn editorissa

Tärkeimmät kohdat tilastoikkunassa mobiilipelin kehittämisen kannalta ovat FPS- (frames per second), Tris- (Triangles) ja Verts (Vertex) -kohdat. FPS näyttää näytön päivitysnopeuden. Samalla se näyttää sen, kuinka monta kertaa ohjelmakoodien Update()-funktiot ajetaan läpi sekunnissa. Update()-funktiosta myöhemmässä vaiheessa lisää. Tris- ja Verts-kohdat ovat tärkeitä varsinkin 3D-mobiilipelejä tehdessä. Niillä tarkkaillaan sitä, kuinka paljon ruudulla kyseisellä hetkellä on kolmiotasoja ja pisteitä, jotka taas vaikuttavat pelin muihin osa-alueisiin, kuten varjostuksiin, kiiltoihin ja niin edelleen. Kolmiotasot muodostuvat pisteiden väliin jäävistä alueista, joka täytetään pinnalla (Face). Esimerkki tällaisesta on kuvan 6 pelihahmossa. Mitä enemmän tietyssä kohdassa peliä pisteitä ja kolmioita on, sitä enemmän ne kuluttavat tehoja. (14.)

Unityn editorissa on myös suoraan asetukset laadulle. Siitä pystyy valitsemaan mahdolliset käytettävissä olevat valmiit asetukset tai voi optimoida itse projektiinsa sopivat. (Kuva 5) Valikon asetukset on jaoteltu kolmeen suurempaan pääteemaan: kuvien luomiseen (Rendering), varjostuksiin (Shadows) ja muihin kohtiin. Valikosta voi säätää kuvien ja varjojen laatua, tarkkuutta sekä erikoistehosteiden fysikaalisia ominaisuuksia.



KUVA 5. Laadun optimointia Unityn editorissa

3.2 2D-grafiikan optimointi

Jos tehdään 3D-peliä, kuten *Tales From The North* on, 2D-grafiikka on pääsääntöisesti pelin käyttöliittymän painikkeita, logoja ja niin edelleen. Perussääntönä 2D-grafiikkaa suunniteltaessa on, että grafiikan tulee olla kahdella jaollinen (power of two). Sääntöä käytetään sekä mobiili- että PC-peleissä, sillä kaikki tietokoneet, konsolit, älypuhelimet ja pelimoottorit käyttävät tätä samaa tekniikkaa prosessoidessaan tietoa. Koska dataa ei prosessoida kerralla vaan osissa, on koneen helpompi prosessoida kahdella jaollisia osia. Kaiken graafisen materiaalin, mitä viedään peliin, tulee siis olla kooltaan kahdella jaettavissa. Tässä on hyvä pitää mielessä perinteisiä tarkkuuksia, kuten 512 x 512 tai 256 x 512 kuva-alkiota (pixel) tuumaa kohti ja niin edelleen. Jos tuotu data ei ole kahdella jaollinen, pelimoottori ”korjaa” koon aiheuttaen turhaa prosessointitehon kuluttamista ja kuvan laadun huonontumista (15.)

Grafiikan kokoa ei kannata laittaa liian suureksi. Pääsääntönä on, että grafiikasta ei tule tehdä sen suurempaa, kuin se pelissä tulee näkymään. Muutoin sen lataaminen syö turhaan laskentatehoja. Yksi helpoimmista tavoista optimoida on grafiikan uusiokäyttö. Jos mahdollista, käytetään samaa kuvaa monessa paikkaa. Esimerkkinä ovat erilaisten logojen taustat. Jos tilanne sen sallii, käytetään samaa kehystä kaikilla logoilla, eikä piirretä jokaiselle logolle omaa kehystä. (15.)

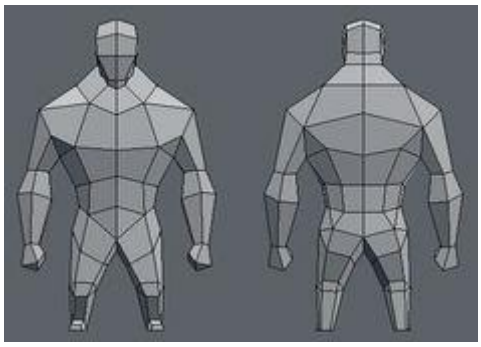
3.3 3D-grafiikan optimointi

3D-grafiikkaan liittyy objektien tai hahmojen mallinnus ja niiden tekstuurien piirtäminen. *Tales From The North* -projektissa käytimme Blender-mallinnusohjelmaa tehdessämme 3D-objekteja. Tyypillisesti heikkotehoisille laitteille mallinnettaessa tehdään niin sanottuja lowpoly-malleja.

Lowpoly-malli tarkoittaa 3D-mallinnuksessa polygoniverkoista koostuvaa mallia, joka sisältää mahdollisimman vähän pisteitä (Verts) (kuva 6). Lowpoly-hahmon mallintaminen on haastavaa, koska mallinnettaessa on pyrittävä luomaan pelimoottorissa toimiva 3D-hahmo pienellä polygonimäärällä. Mallintamisella tarkoitetaan sitä, kun hahmosta luodaan 3D-objekti erillisellä mallinnusohjelmalla.

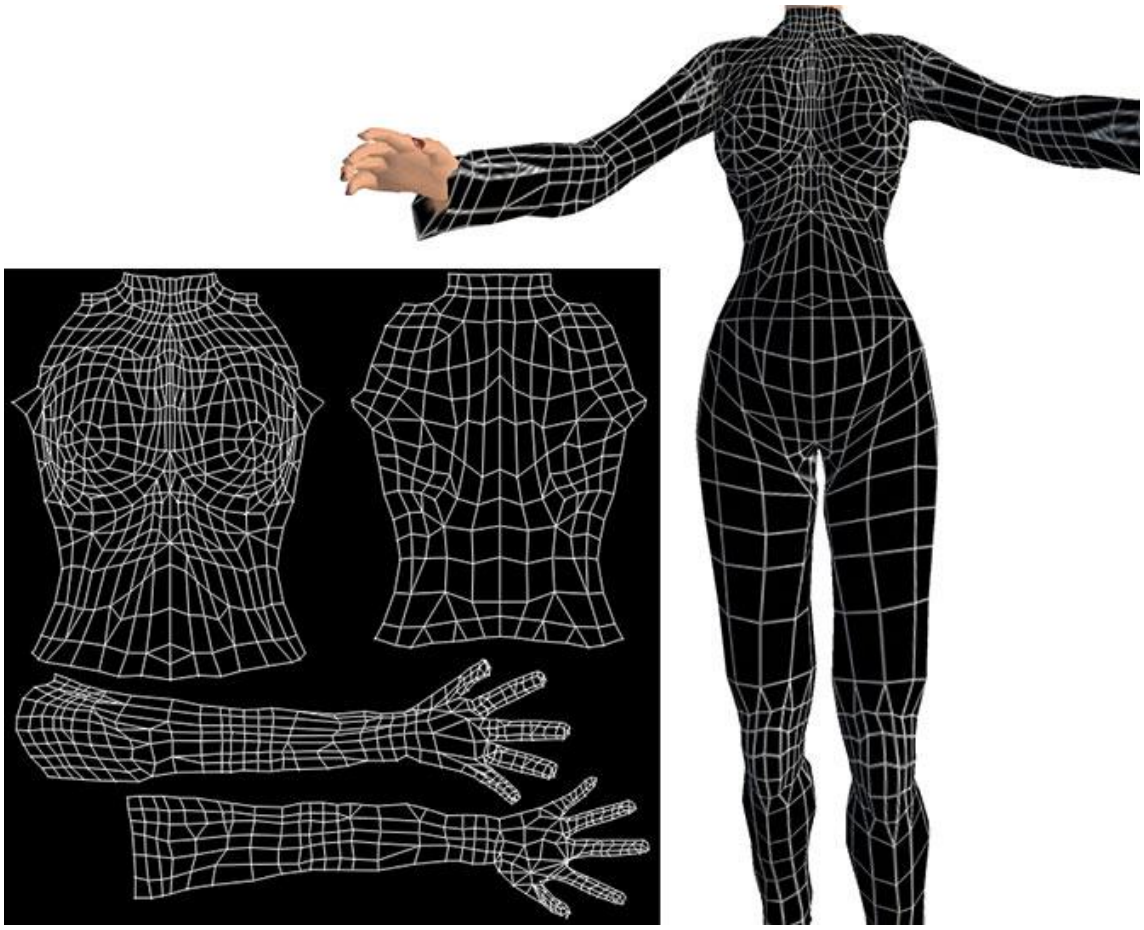
Apuna voidaan käyttää hahmosta piirrettyjä 2D-luonnoskuvia esimerkiksi etuprofiilista ja sivuprofiilista. Lowpoly-malleista pyritään harvoin tekemään realistisia, ja tunnusomaista on, että malleissa esiintyy teräviä kulmia. (16.)

Kun malli on lowpoly, siihen vaikuttavat luonnollisesti vähemmän esimerkiksi erilaiset valaistukset ja varjot jne. Yleistä on myös se, että lowpoly-grafiikkoja käytettäessä varjoja ja kiiltoja ei oteta käyttöön ollenkaan, vaan ne piirretään itse tekstuurikarttaan (kuva 7), joka asettuu mallin päälle.



KUVA 6. Lowpoly malli

Mallinnetuille 3D-hahmoille tai objekteille luodaan pinta tai iho. Pinnan tekemiseen on erilaisia työkaluja. Ammattilaiset käyttävät yleensä kallista lisäosaa, jonka avulla pinta voidaan piirtää suoraan mallin päälle. Yleisempää on tehdä objektista niin sanottu UV-kartta, jonka kirjaimet u ja v tulevat akseleista x ja y. Objekti siis hajotetaan pienempiin osiin, jotka asetetaan 2D-tasolle. (kuva 7). Tasolle sitten joko suoraan piirretään osien päälle haluttu tekstuuri tai tarkempaa työskentelyä varten kartta voidaan siirtää johonkin piirto-ohjelmaan, jonka jälkeen se sitten ladataan takaisin mallinnusohjelmaan. (17.)



KUVA 7. UV-mapping

Optimoinnin kannalta oleellista on, että jos mallissa käytetään esimerkiksi vain yhtä tiettyä väriä osa-aluetta kohti, tulisi kuvan koko asettaa todella pieneksi. Muutenkin tulee tarkastella, kuinka pienellä kuvakoolla tekstuurikartta näyttää vielä miellyttävän hyvältä pelimaailmassa.

3.4 Unityn optimointimahdollisuudet mobiililaitteelle

Unityn sivuilta (18) löytyy ohjeet siihen, miten peliä kannattaa optimoida, kun tuote on tarkoitettu suorituskyvyltään heikommille laitteille, kuten mobiililaitteet ovat. Sivuston muistilistan mukaan ohjeet ovat vapaasti suomennettuna seuraavan kaltaiset:

- Yritä pitää Verts- ja Tris -kohdat mahdollisimman pienissä lukemissa. Näytönohjaimesta riippuen alle 200 000 – 3 000 000. (Oman kokemuksen mukaan lukemien täytyy olla jopa alle 50 000 mobiililaitteilla).
- Jos käytät sisäänrakennettuja varjostustoimintoja, valitse ne Mobile- tai valaisematon (Unlit) -kohdista.
- Käytä mahdollisimman vähän eri materiaaleja yhdessä näkymässä ja jaa samat materiaalit niin monelle eri esineelle kuin mahdollista.
- Aseta liikkumattomat kohteet staattisiksi. (Jos objekti ei tarvitse fysiikoita, esimerkiksi painovoimaa, niitä ei kannata laittaa sille tehoa kuluttamaan.)
- Laita mieluiten vain yksi suunnattu valaistus (Directional light), jos näkymä ei muita välttämättä vaadi.
- Käytä pysyvää valaistusta (Bake-lightning) ennemmin kuin toiminnallista valaistusta (Dynamic-lightning). Pysyvä valaistus lasketaan pelinäkymään heti alussa, kun uuteen näkymään siirrytään, eikä sitä enää sen jälkeen lasketa uudestaan. Toiminnallinen valaistus taas lasketaan jokaisella näytön päivityskerralla uudestaan.
- Käytä pakattuja tekstuuritiedostojamuotoja, jos mahdollista (esimerkiksi .png) ja käytä 16-bittisiä tekstuureja ennemmin kuin 32-bittisiä tekstuureja.
- Vältä sumun käyttämistä, jos mahdollista.
- Rajoita näkyvien objektien määrää ja niiden näyttämistä (Occulsion Culling) kameran asetuksista. Suunnittele kentät siten, että mahdollisimman vähän objektien näyttämistä tulisi kerralla näytölle.
- Käytä ”Skybokseja” ”huijaamaan” kaukaisia objekteja pelaajalle. Tämä tarkoittaa sitä, että laitetaan taustalle pysyvä tai hieman liikkeitä mukaileva koko näytön peittävä taustakuva esimerkiksi vuoristosta tai taivaasta.

Grafiikan optimointi on oleellisen tärkeää, kun peli halutaan saada pieneen ja kevyeen kokoon. Grafiikka vie pelin koosta suuren osan ja samalla grafiikkatiedostojen koko vaikuttaa pelin laskentatehoon.

4 TOTEUTUS PROJEKTISSA TALES FROM THE NORTH

4.1 Suunnittelu projektissa Tales From The North

Projektin alussa suunnitteluun käytettiin noin 40 tuntia. Suunnitteluvaiheessa käytiin läpi mm. pelin sisältöä, ulkoasua ja mekanismeja sekä koodin jäsentämistä ja nimeämistä.

Pelin genreksi päädyttiin valitsemaan suosituksi noussut toimintaroolipeli (action role-playing game). Toimintaroolipelit ovat nopeatempoisuutensa sekä hahmon kehittämisen kannalta koukuttavia pelejä. Peliominaisuudet on toteutettu omalla tyylillä, vaikkakin niihin on haettu inspiraatiota samankaltaisista peleistä. Toimiviksi todistettuja mekanismeja, kuten satunnaislukugeneraattorilla luotuja arvoja varusteisiin, käytettiin yhtenä pelin ominaisuuksista. Näihin tarvittavat kaavat tutkittiin ja luotiin Excelillä. Tämä tarkoittaa sitä, että kun pelaaja avaa arkun, sen sisältä arvotaan eri vaihtoehdoista ensin varuste. Sen jälkeen varusteelle arvotaan muutamasta eri harvinaisuusasteesta (rarity) sen harvinaisuus. Harvinaisuusaste lisää aina varusteesta saatavia hyötyjä. Kun vihreällä värillä merkitystä varusteesta saa kahteen ominaisuuteen lisäpisteitä, sinisestä taas saa kolmeen ja niin edelleen (kuva 18).

Peli on tarkoitettu ensisijaisesti HC-pelaajille, jotka haluavat viettää aikaa hahmonkehityksen, erilaisten taitojen, varusteiden ja taitopisteiden kombinaatioiden pyörittelyssä. Pelin oletetaan kiinnostavan 15–25-vuotiaita poikia ja miehiä, joilla on kokemusta roolipelien pelaamisesta jo aiemminkin.

Suunnittelun ja optimoinnin kannalta oleellista oli alussa miettiä, miten paljon reaaliaikaisen 3D-hahmon ohjaaminen ja taitojen käyttäminen syövät resursseja mobiililaitteilta. Kun siihen vielä lisätään 3D-maailman fysiikkaominaisuuksilla varustetut objektit sekä reaaliajassa toimivat ja liikkuvat vastustajat, täytyy optimointia todella miettiä. Pelin päivitysnopeus pitää saada säilymään tarpeeksi korkeissa lukemissa.

Pelimaailma päädyttiin rakentamaan pienemmistä pelinäkymien kokonaisuuksista. Idea on, että peli koostuu muutamasta isommasta tarinasta, jotka jokainen

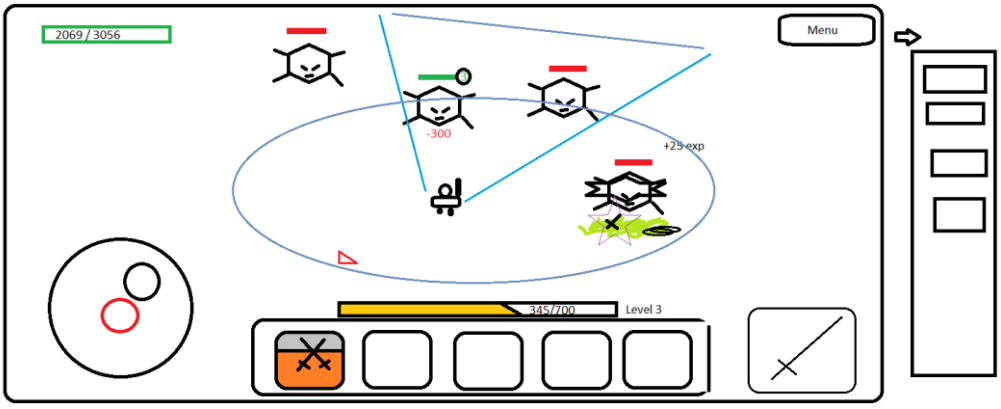
on jaoteltu noin viiteen eri karttaan. Yksi kartta sijoittuu aina erilaiseen maastoon, esimerkiksi metsä, kaupunki tai luolasto. Jokainen kartta on taas jaettu 4–10 pienempään pelinäkymään, joissa edetään portaalien välityksellä. Yksi tällainen pelinäkymä on pyritty optimoimaan tilan ja grafiikan suhteen siten, että mahdollisimman vähän suoritustehoa kuluisi niiden alustamiseen ja päivittämiseen.

Optimointia pelinäkymän sisällä on toteutettu siten, että jokaista tekoälyä sisältävää vihollista ei heti alkuvaiheessa laiteta kuluttamaan suorituskykyä peliin. Viholliset synnytetään (spawnataan) peliin määritettyyn paikkoihin vasta, kun päähahmo menee karttaan laitetun laukaisimen (triggerin) ylitse. Kun päähahmo on kävellyt laukaisimen ylitse, synnytetään 1–3 vihollista määritetylle paikalle. Kun vihollinen kuolee, se luonnollisesti poistetaan tehoa syömästä. Näin jatkuu läpi pelinäkymän, kunnes saavutaan portaalille, josta matka jatkuu seuraavaan näkymään.

4.2 UI-tasojen suunnittelu projektissa Tales From The North

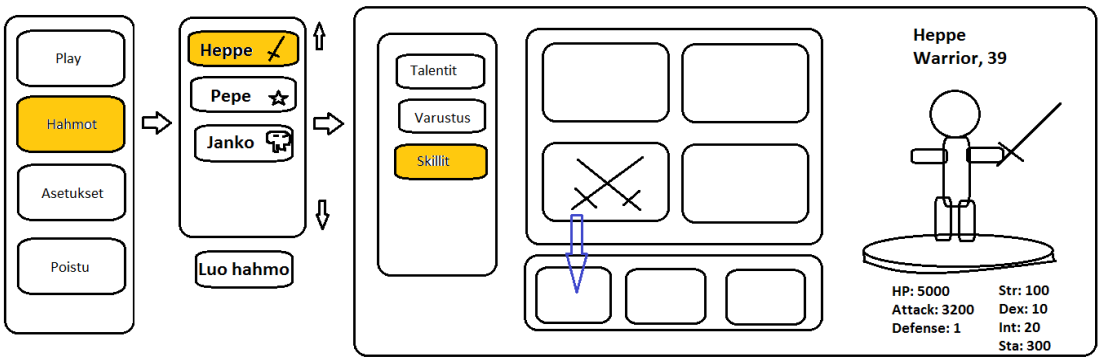
Kuten olettaa saattaa, vaikka olemassa on hienoja suunnitteluun tarkoitettuja ohjelmia ja metodeja, tekee monesti mieli työskennellä omilla toimintatavoilla. Kuvien 8, 9 ja 10 suunnitelmat toteutettiin Microsoftin piirtotyökalulla MS Paint.

Kuvassa 8 on suunnitelma pelinäkymän UI-tasosta ja hahmotelma pelimekaniikasta. Kuvassa näkyy keskellä olevan pelihahmon ympärille piirretyt kolmio ja ympyrä, jotka kuvaavat taitojen ja lyöntien vaikutusalueita. Vasemmassa alareunassa on piirrettynä ohjaimen hahmotelma, vasemmassa yläreunassa elämäpisteiden palkki, oikeassa yläreunassa pelinäkymän menuvalikon painike ja siitä avautuva lisävalikko, josta pääsee katsomaan pelaajan tietoja tai takaisin päävalikkoon ja alhaalla on taitojen ja taisteluominaisuuksien painikkeet, joita painamalla pelihahmo tekee haluttuja liikkeitä ja iskuja. Taisteluominaisuuksien yläpuolella on palkki kokemuspisteiden seuraamiseen. Aina kun palkki täyttyy, pelihahmon taso kasvaa yhdellä.



KUVA 8. Pelin UI-tason suunnittelu

Kuvassa 9 on hahmoteltu päävalikkoa, jossa vasemmalla olevat valikkoalueet päivittyvät näytön vasemmassa reunassa nuolien kaltaisesti. Samalla näytön keskiosa ja oikea reuna päivittyvät käytössä olevalle valikkoalueelle tarkoitetuille tilastoille.



KUVA 9. Päävalikon suunnittelu

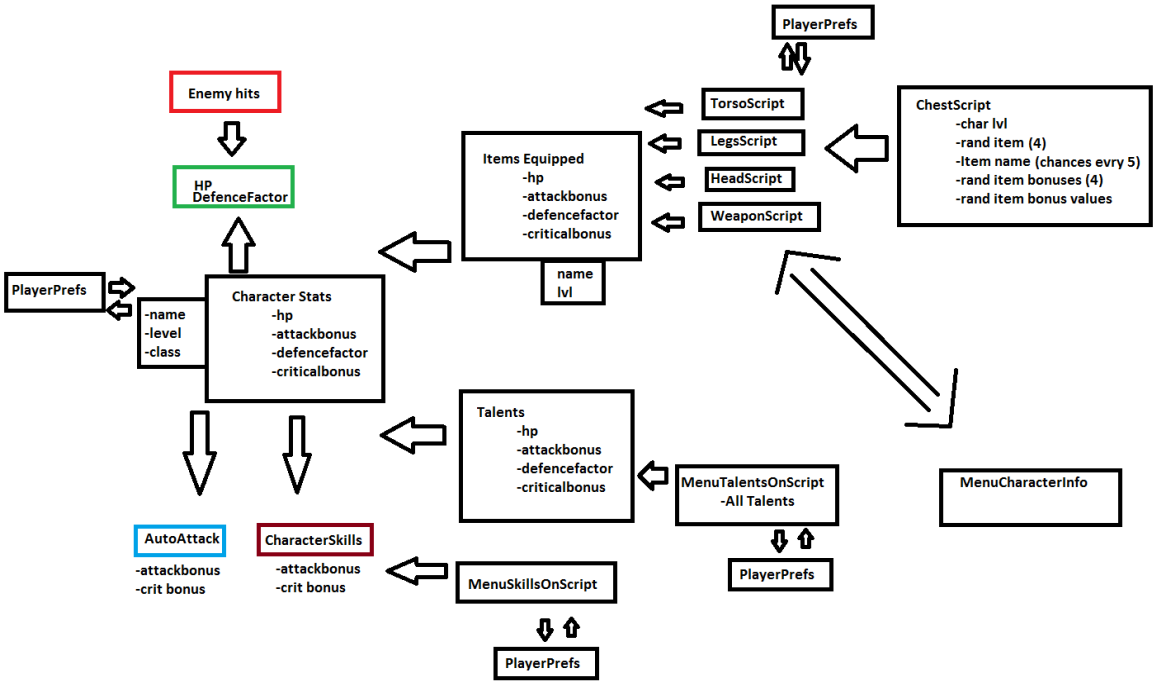
4.3 Koodin rakentaminen

Koska ohjelmoijalle on ensisijaisen tärkeää ymmärtää itse kirjoittamansa koodi, päädyttiin rakentamaan myös sellainen koodikaavio, jonka ymmärtää itse helposti (kuva 10). Luokat ja muuttujat pyrittiin nimeämään kategorioittain, pääluokat ja alaluokat samankaltaisilla nimillä. Tarkoituksena sillä oli helpottaa koodin kirjoittamista ja lukemista. Esimerkiksi päämuuttujana on *AttackPower* ja sen alamuuttujina on mm. *AttackPowerMain* ja *AttackPowerFromItems*.

Kuvan 10 ja dokumentin suunnitelmien pohjalta päähahmon koodia pelimoottorissa lähdettiin rakentamaan päähahmon kannalta pienempiin osa-alueisiin jaettuna. Päähahmo jaoteltiin kuuteen suurempaan luokkaan: *CharacterStats*, *CharacterItemsEquipped*, *CharacterTalents*, *CharacterSkills* ja *CharacterAttack*. Esimerkiksi *CharacterStats*-skriptissä on koottuna yhteen hahmon kaikki perustiedot, joihin sisältyy arvoja erilaisille ominaisuuksille, joita hahmolla on.

Hahmojen ominaisuuksiin kuuluvat mm. elämäpisteet, lyöntivoima ja puolustus. Koodissa on tehtynä päämuuttuja, johon haetaan tiedot kaikista alaluokista. Esimerkiksi hahmon elämäpisteet haetaan jokaisesta puetusta varusteesta ensin *CharacterItemsEquipped*-skriptiin, josta tiedot haetaan tiedot kokoavaan *CharacterStats* -skriptiin.

PlayerPrefs on Unityn tallennusluokka, joka ajaa integer-, float-, string- ja boolean-tyyppiset muuttujat tallennustiedostoon, josta niitä voidaan kutsua ja tallentaa. Kuvassa 10 on hahmoteltu skriptien välisiä suhteita sekä tallennusmekanismia.



KUVA 10. Koodikaavio

4.3.1 Muuttujien laskukaavat

Pelissä hahmoilla on monenlaisia ominaisuuksia, joille on täytynyt laskea pelaajan tasoon suhteuttaen kaavat, joiden mukaan arvot skaalautuvat. Kaavoihin sisältyy lineaarisia, eksponentiaalisia ja juurifunktiollisia lausekkeita.

Esimerkiksi vahinkoa joka pelaajalle koituu, voidaan vähentää pelissä saatavilla puolustuspisteillä. Taulukossa 1 puolustuspisteet (*DefenceValue*) muutetaan prosenteiksi (*DamageReduction*), jotka kertovat paljonko alkuperäisestä pelaajalle tehdystä iskusta vähennetään määrää prosentteina. Puolustuspisteistä saatava hyöty vähenee prosentilla aina kun hahmon taso nousee yhdellä, joka näkyy selkeästi kuvan 11 kuvaajassa. Lausekkeesta otetaan juurifunktio, jolloin lausekkeen kuvaaja saadaan käyttäytymään logaritmisesti halutulla tavalla.

TAULUKKO 1. DefenceValuen laskukaava ja taulukko

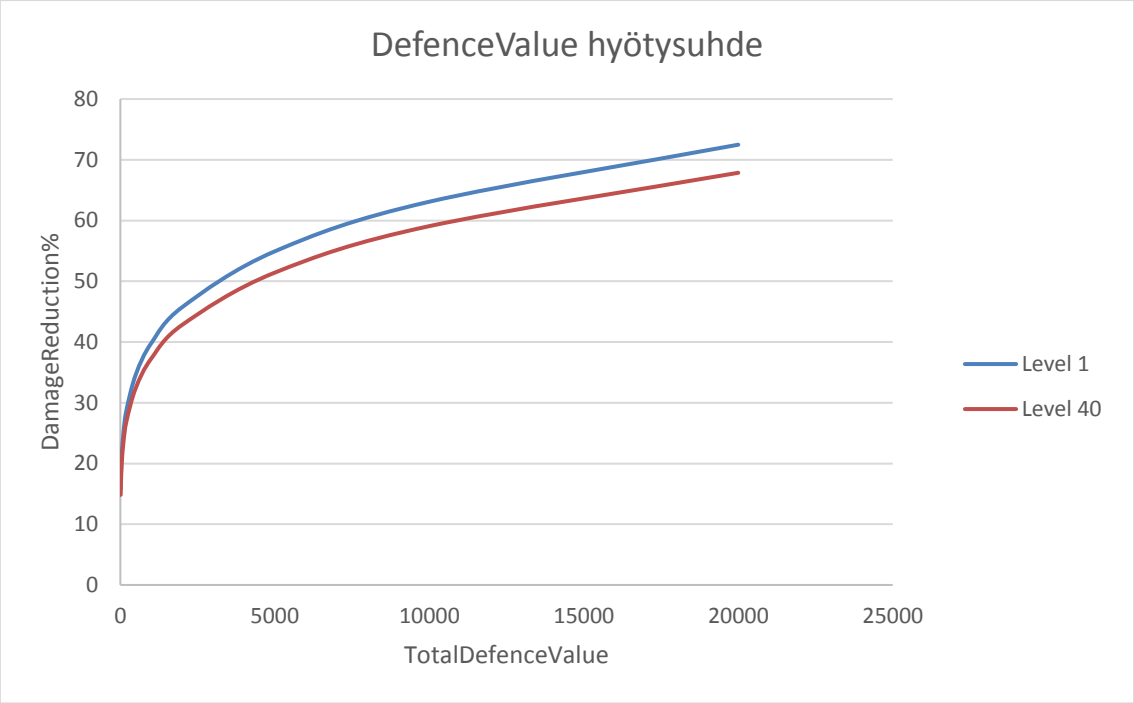
Totaldefense skaalaus damagereductioniin levelin kasvaessa

(DefenseFactorFromClass * TotalDefenseValue / (1+ %ReductionPerLevel * CharacterLevel - %ReductionPerLevel)) ^Root * 10

Root	a	b	a/b
	1	5	0,2

DefenseFactorFromClass	1
%ReductionPerLevel	0,01

	Level	1	2	3	4	5	10	15	20	25	30	40
Total%Red.PerLevel		1	1,01	1,02	1,03	1,04	1,09	1,14	1,19	1,24	1,29	1,39
TDF	10	15,84893	15,81742	15,78629	15,75551	15,7251	15,57811	15,439	15,30702	15,18153	15,06198	14,83874
	20	18,20564	18,16945	18,13368	18,09833	18,06339	17,89455	17,73475	17,58315	17,439	17,30167	17,04524
	30	19,7435	19,70425	19,66546	19,62713	19,58924	19,40613	19,23283	19,06843	18,91211	18,76317	18,48508
	40	20,91279	20,87121	20,83013	20,78952	20,74939	20,55544	20,37188	20,19773	20,03215	19,8744	19,57984
	50	21,86724	21,82377	21,78081	21,73835	21,69638	21,49358	21,30164	21,11955	20,94641	20,78146	20,47345
	100	25,11886	25,06893	25,01958	24,97081	24,9226	24,68964	24,46916	24,25999	24,06111	23,87163	23,51782
	200	28,854	28,79663	28,73995	28,68392	28,62855	28,36095	28,10768	27,86741	27,63896	27,4213	27,01488
	500	34,65724	34,58834	34,52025	34,45296	34,38645	34,06502	33,76082	33,47223	33,19783	32,93639	32,44823
	1000	39,81072	39,73157	39,65336	39,57606	39,49966	39,13044	38,781	38,44949	38,13429	37,83398	37,27323
	2000	45,73051	45,63959	45,54975	45,46095	45,37319	44,94907	44,54767	44,16687	43,8048	43,45983	42,8157
	5000	54,92803	54,81883	54,71091	54,60426	54,49885	53,98942	53,5073	53,0499	52,61501	52,20067	51,42698
	10000	63,09573	62,97029	62,84634	62,72383	62,60274	62,01756	61,46375	60,93834	60,43878	59,96282	59,07409
	20000	72,47797	72,33387	72,19148	72,05076	71,91166	71,23947	70,6033	69,99977	69,42592	68,87919	67,85831



KUVA 11. DamageReductionin ero tasojen 1 ja 40 välillä

Kaikista kaavoista, joita pelimoottorissa käytetään, löytyvät Excel-taulukot ja kuvaajat. Kun kaavat saatiin valmiiksi Excelillä, ne siirrettiin koodiin. Koodauksessa täytyi tietenkin ottaa huomioon se, miten kyseisellä ohjelmointikielellä (C#) toteutetaan tällaiset lausekkeet. Aluksi joidenkin vaikean näköisten kaavojen kohdalla ei uskottu että ne voisivat toimia ohjelmakoodissa, mutta pienen tutkiskelun ja opettelun päätteeksi kaavat saatiin toimimaan halutulla tavalla koodissa sekä itse pelissä.

4.3.2 Mobiililaitteiden huomiointi koodin toteuttamisessa

Kun koodia lähdetään toteuttamaan mobiililaitteille, tai mille tahansa muutakaan laitteelle, oleellisen tärkeää on saada koodi mahdollisimman kevyeksi. Kevyeksi saamisella tarkoitetaan sitä, että koodin ajaminen kuluttaa mahdollisimman vähän tehoa laitteelta.

Unityllä koodatessa muutamat pääfunktiot ovat tärkeitä ymmärtää, kun koodia lähdetään toteuttamaan. *Awake()*, *Start()*, *FixedUpdate()*, *Update()* ja *LateUpdate()* ovat määrättyjä funktioita, joita toteutetaan tietyssä järjestyksessä.

Awake()-funktio ajetaan läpi ennen ainoatakaan muuta funktiota. Se suoritetaan vain kerran heti, kun peliobjekti syntyy pelinäkymään tai pelinäkymä käynnistyy. *Start()*-funktio suoritetaan myöskin kerran, mutta juuri ennen ensimmäisen kuvan (framen) päivittymistä pelissä. Yleinen sääntö on, että *Awake()*-funktiossa alustetaan kaikki kyseiseen peliobjektiin liittyvät muuttujat ja *Start()*-funktiossa alustetaan ne muuttujat, jotka ovat riippuvaisia toisista luokista ja objekteista. Tarkemmin funktioiden suoritusjärjestyksestä ja niille suositelluista käyttökohteista on tietoa Unityn manuaalissa. (19.)

Update()-funktiot suoritetaan jokaisella pelin kuvalla. Jos *Update()*-funktioon laitetaan paljon raskasta laskutoimitusta vaativaa koodia, pelin päivitystaajuus (FPS) laskee huomattavasti. Tämän huomiointi on erityisen tärkeää, kun koodataan mobiilialustalle. Koodi tulisi toteuttaa siten, että vain välttämättömimmät laskurit suorittavat laskutoimintoja samanaikaisesti. Näin säilytetään mahdollisimman pieni laskenta yhtä kuvaa kohti ja FPS säilyy korkeana. Esimerkiksi *Tales From The North* -projektissa olemme tehneet suurten laskukaavojemme kanssa siten, että laskemme *Start()*-funktiossa kaavan muuttujaan, jota käytämme sitten kertoimena sitä vaativissa kohdissa. Tällä vältetään se, että laite joutuisi jokaisella kuvalla laskemaan saman kaavan, joka taas söisi huomattavasti tehoa.

Kun koodataan ajastimen varaan jotakin toimintoa, on tärkeää muistaa myöskin sammuttaa se suorituskykyä kuluttamasta. Laitetaan ehtolauseke ajastinta varten niin, että jos ajastin ylittää tietyn raja-arvon, se ei enää sen jälkeen saa toteuttaa ajan laskemista. Ajastimen arvo voidaan taas palauttaa normaaliin muuta kautta, esimerkiksi nappia painamalla, jolloin se taas lähtee laskemaan alusta.

Yksi käteväksi havaittu keino on, että kun jossakin koodissa ei tarvitsekaan toteuttaa kuin kerran tietyllä hetkellä jokin toiminto, alustetaan sitä varten *Awake()*- tai *Start()*-funktioissa ”*doOnce”* -muuttuja todeksi (*doOnce = true*). Sitten asetetaan se ehtolausekkeeseen siten, että jos *doOnce* on tosi, toteutetaan toiminto ja sitten määrätään *doOnce* epätodeksi (*doOnce = false*). Näin säästyy taas laskentatehoa, kun kyseisen kohdan yli hypätään koodissa, kun se on jo käytetty. (kuva 12).

```
    private bool doOnce;

    void Start()
    {
        doOnce = true;
    }

    void Update()
    {

        if (doOnce == true)
        {
            // do the action here
            doOnce = false;
        }

    }
```

KUVA 12. Esimerkki doOnce -muuttujan käytöstä

5 TULOKSENA MOBIILIPELI TALES FROM THE NORTH

Opinnäytetyön perustana oli mobiilipohjainen 3D-toimintaroolipeli. Pelin nimi on *Tales From The North*. Peliä pelataan ns. isometrisestä kuvakulmasta, eli pelaajaa kuvataan takayläviistosta. Ajatuksena on liikkua ja taistella pelihahmolla siten, että käytetään erilaisia taitoja ja taisteluominaisuuksia käyttöliittymään tehdyltä kontrollerilta, jotka näkyvät kuvan 13 vasemmassa ja oikeassa alareunassa. Kamera on kiinnitetty pelaajaan. Pelissä saa valita hahmon pääluokan, jonka ominaisuuksia kehitetään erilaisilla pelialueilla. Tehtävät määräytyvät oman pelaajan taitotason mukaan. Hahmoluokissa on pitkän kantaman tyyppisiä sekä lähitaisteluun erikoistuneita luokkia.

KUVA 143. Pelinäkymä jousimiehen taistelusta

5.1 Talentit

Hahmonkehityksessä taitopuuhun (talent tree) saa hahmon tason (level) perusteella pisteitä, joilla voidaan muokata pelihahmoa (kuva 14). Erityistaitoja pelissä on 144 kappaletta. Hahmon erityistaitoja muokataan päävalikon tai kylästä löytyvän kirjan kautta. Taidot vaikuttavat hahmon ominaisuuksiin pelissä halutulla tavalla. Esimerkiksi taitojen kautta voi lisätä elämäpisteitä tai lisätä voimaa haluamaansa iskuun. Taitoja kertyy yksi hahmon tasoa kohti.

KUVA 14. Näkymä taitopuusta

5.2 Päävalikko

Päävalikko koostuu hahmonvalinnasta, johon sisältyy hahmon pääluokan ja alaluokan valinta. Pelissä on 3 eri pääluokkaa, Warrior, Magician ja Thief. Näiden alta valitaan vielä kolmesta eri vaihtoehdosta erilaisilla taidoilla ja varusteilla varustettu alaluokka. Käyttäjä saa vertailla eri vaihtoehtoja ja luoda hahmonsa mieleisensä luokan ja alaluokan mukaan.

Kuvassa 15 on näkymä hahmonvalinta-valikosta, jossa vasemmassa yläreunassa on valikkoalue pelihahmoille. Pelihahmon valinta-alueen alapuolella näkyvät hahmon rahat ja poletit, joilla voi ostaa kaupungissa olevalta sepältä uusia varusteita. Oikeassa yläreunassa on tiedot hahmon taitopisteistä. Taitopisteiden alapuolella on painikkeet varusteiden tiedoille. Painiketta painamalla keskelle näyttöä avautuvat tiedot hahmon varustekohtaisista taitopisteistä. Keskellä alhaalla on painike hahmon muokkausta varten. Tästä painikkeesta pääsee taitopuuhun ja vaihtamaan hahmon alaluokkaa. Oikeassa alareunassa on painike, jolla päästään valitsemaan kartta, mitä halutaan lähteä pelaamaan.

KUVA 15. Hahmon valinta

5.3 Pelin käyttöliittymä ja ajatus

Pelin käyttöliittymä koostuu muutamasta eri osa-alueesta. Kuvassa 16 vasemmassa yläreunassa näkyvät pelaajan elämäpisteet ja kokemuspisteet. Oikeassa yläreunassa on menuvalikon kuvake (hammasratas), josta pääsee takaisin päävalikkoon tai katsomaan tarkemmin hahmon tiedot ja varusteet. Vasemmassa alareunassa on ohjain, jolla hahmoa liikutellaan kentällä. Alhaalla oikealla on taitojen ja iskujen valinta-alue, jossa on hahmon luokkaan ja alaluokkaan perustuvat taidot. Niitä käytetään taisteluissa vihollisia vastaan. Oikeassa reunassa keskellä näkyy automaattihyökkäyksen ”on/off” -painike, jolla saadaan automaattihyökkäys joko päälle tai pois päältä.

KUVA 16. Käyttöliittymä

Kenttien lopussa tai kovatasoiselta viholliselta saadaan arkku, josta arvotaan pelaajan tason perusteella uusi varustus tai ase. Arkun voi myös ostaa ja avata sepältä (kuva 17), joka löytyy aina kenttien alkupäässä olevasta turvapaikasta. Aseissa ja varusteissa on viisi eri tasoa. Mitä harvinaisempi ja parempi ase tai varustus on, sitä pienempi todennäköisyys arkusta on saada kyseinen varuste. Todennäköisyyttä saada parempi varuste voidaan kuitenkin kehittää pelin edetessä loppupuolelle.

KUVA 17. Uuden varusteen osto ja arkun aukaiseminen sepältä

LOPPUSANAT

Työn tarkoituksena oli saada kokonaiskuva isomman projektin suunnittelusta ja toteutuksesta sekä oppia matkan varrella niitä pieniä nyansseja, jotka ratkaisevat ohjelman ja pelin järkevän toimimisen. Tuloksena saatiin testiversio 3D-toiminta-roolipelistä, joka jatketaan aina demo- ja kokoversioihin.

Projektin aikana on opittu paljon siitä, kuinka ohjelmasta tehdään mobiililaitteita varten kevyt. Ennen projektin alkua tietämys 3D-mallinnuksesta ja pelin tekemisestä 3D-muotoisena oli lähes olematonta. Tässä vaiheessa projektia kokonaiskuva alkaa olla hyvin hallussa. Optimoinnin kannalta omassa projektissamme on onnistuttu kohtalaisesti. Työnsarkaa löytyy vielä kuitenkin tiedostokokojen minimoimisen ja koodin läpikäymisen parissa.

Peli tullaan julkaisemaan ensin Androidille Google Play -kauppaan ja myöhemmin iOS-käyttöjärjestelmälle Apple Storeen. Androidille testiversio pelistä löytyy tästä linkistä:

https://play.google.com/store/apps/details?id=com.NoblesGameStudio.TalesFromTheNorth_Demo.

LÄHTEET

1. Unity, 2017. Hakupäivä 23.3.2017. Saatavissa: https://unity3d.com/

2. Asset Store 2017. Unity. Hakupäivä 23.3.2017. Saatavissa: https://www.assetstore.unity3d.com/en/

3. Soininen, Teppo 2006. Peliohjelmointi. Cloud Solutions CS Oy. Hakupäivä 15.11.2016. Saatavissa: http://www.cs.tut.fi/kurssit/OHJ-2710/2006JaVanhemmat/Pelisuunnittelu2006_4.pdf

4. Angry Birds 2017. Google Play. Hakupäivä 23.3.2017. Saatavissa: https://play.google.com/store/apps/details?id=com.rovio.angrybirds

5. Hill Climb Racing 2017. Google Play. Hakupäivä 23.3.2017. Saatavissa: https://play.google.com/store/apps/details?id=com.fingersoft.hillclimb

6. Clash Of Clans 2017. Google Play. Hakupäivä 23.3.2017. Saatavissa: https://play.google.com/store/apps/details?id=com.supercell.clashofclans

7. Haaramo, Erkki. Koodauksen ylistys! Ohjelmistoyrittäjät Ry. Hakupäivä 1.12.2016. Saatavissa: https://ohjelmistoyrittajat.fi/fi/koodauksen-ylistys-0

8. Haikala, Ilkka — Mikkonen, Tommi 2011. Ohjelmistotuotannon käytännöt. 12. uudistettu painos. Helsinki: Talentum.

9. Paussu, Jari 2012. Web-sivujen sovittaminen mobiililaitteille. Opinnäytetyö. Turku: Turun ammattikorkeakoulu, tietotekniikan koulutusohjelma, ohjelmistotuotannon suuntautumisvaihtoehto.

10. Canvas 2017. Unity manuals. Hakupäivä 23.3.2017. Saatavissa: https://docs.unity3d.com/Manual/UICanvas.html

11. Class-canvas 2016. Unity manuals. Hakupäivä 21.11.2016. Saatavissa: https://docs.unity3d.com/Manual/class-Canvas.html

12. Graziano, Dan 2012. NVIDIA's mobile GPUs to surpass Xbox 360 performance by 2014. BGR Media. Hakupäivä 1.12.2016. Saatavissa: http://bgr.com/2012/04/20/nvidias-mobile-gpus-to-surpass-xbox-360-performance-by-2014/

13. Magic Legion 2017. Google Play. Hakupäivä 23.3.2017. Saatavissa: https://play.google.com/store/apps/details?id=com.xgg.ml&hl=fi

14. Rendering Statistics Window 2017. Unity manuals. Hakupäivä 6.3.2017. Saatavissa: https://docs.unity3d.com/Manual/RenderingStatistics.html

15. Jurvelin, Hanna 2016. 2D-mobiilipeligrafiikan optimointi. Opinnäytetyö. Oulu: Oulun ammattikorkeakoulu, viestinnän tutkinto-ohjelma, kuvallisen viestinnän suuntautumisvaihtoehto.

16. Jauhiainen, Aku 2015. 3D-HAHMOJEN MALLINTAMINEN MOBIILIPELIIN –case Northbound. Opinnäytetyö. Turku: Turun ammattikorkeakoulu, tietotekniikan koulutusohjelma, mediatekniikan suuntautumisvaihtoehto.

17. Mullen, T 2009. Mastering Blender. 1st ed. Indianapolis, Indiana: Wiley Publishing, Inc.

18. Optimizing graphics performance 2017. Unity manuals. Hakupäivä 6.3.2017. Saatavissa: https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html

19. Execution Order of Event Functions 2017. Unity manuals. Hakupäivä 6.3.2017. Saatavissa: https://docs.unity3d.com/Manual/ExecutionOrder.html