**OPINNÄYTETYÖ**

**JANNE REMES 2011**

# PELIKEHITYS UNITY 3D-YMPÄRISTÖSSÄ

Rovaniemen
ammattikorkeakoulu
University of Applied Sciences
LUC

**TIETOTEKNIIKAN KOULUTUSOHJELMA**


ROVANIEMEN AMMATTIKORKEAKOULU

TEKNIIKKA JA LIIKENNE

Tietotekniikan koulutusohjelma

Opinnäytetyö

# PELIKEHITYS UNITY3D-YMPÄRISTÖSSÄ

JANNE REMES

2011

Ohjaaja Kenneth Karlsson

Hyväksytty _____ 2011 _____

Työ on kirjastossa lainattavissa

Rovaniemen ammattikorkeakoulu
University of Applied Sciences
LUC

Tekniikka ja liikenne
Tietotekniikan koulutusohjelma

Opinnäytetyön tiivistelmä

**Tekijä** Janne Remes Vuosi 2011

**Työn nimi** Pelikehitys Unity 3D-ympäristössä
**Sivu- ja liitemäärä** 70 + 2

Tämän opinnäytteen tarkoituksena on suunnitella ja kehittää 3D-kauhuseikkailupeli Unity 3D-pelimoottorilla. Samalla perehdytään pelituotannon osa-alueisiin, kuten 3D-mallinnukseen, animointiin, ohjelmointiin, sekä projektin- ja versionhallintaan. Projektissa käytetään tekijän omistamaa Unity 3D Pro -lisenssiä. Ilmaisia avoimen lähdekoodin sovelluksia suositaan muiden tarvittavien työkalujen valinnassa. Opinnäytteen tuloksena syntyy pelisuunnitelmaan pohjautuva prototyyppi, jolle on tehty asiakaskysely ja käytettävyystestaus.

Asiakaskyselyn tavoitteena on saada palautetta pelistä ja määrittää pelin ikäryhmä. Käytettävyystestauksen ja asiakaspalautteen perusteella peliin suunnitellaan jatkokehityssuunnitelma. Testaajia pidetään pelin mahdollisina asiakkaina ja heistä kerätään tarvittavia tietoja liiketoimintasuunnitelmaa varten. Opinnäyte alkaa tutustumalla pelikehityksen liittyviin teorioihin ja käsitteisiin. Seuraavaksi esitellään projektissa käytetyt työkalut ja tämän jälkeen käydään läpi projekti vaiheittain. Lopuksi esitellään testitapaukset ja saadut tulokset.

Projektin jatkuva laajeneminen pidettiin kurissa hyvällä projektin hallinnalla ja dokumentoinnilla. Ideat jotka eivät päätyneet prototyyppiin, tulivat hyvin dokumentoiduiksi myöhempää käyttöä varten. Positiivista palautetta ja uusia ideoita saatiin pelitestauksen aikana. Kokonaisuudessaan projekti oli onnistunut.

Opinnäytteen valmistuttua prototyypistä jatkokehitetään kaupallinen versio, joka julkaistaan PC:lle, Mac:lle ja selaimelle. Opinnäytteen lopullisena päämääränä on aloittaa tekijän oma indie-pelistudio.

Avainsana(t) pelit, pelisuunnittelu, kenttäsuunnittelu, pelinkehitys, unity 3d, videopelit, 3D-mallinnus
Muita tietoja Työhön liittyy esittely video.

Rovaniemen ammattikorkeakoulu
University of Applied Sciences
LUC

School of Technology Information Technology Programme

Abstract of Thesis

**Author** Janne Remes Vuosi 2011

**Subject of thesis** Game development in unity 3d environment
**Number of pages** 70 + 2

The aim of this thesis was to design and develop a 3D horror adventure game with the Unity 3D game engine. At the same time the author will learn game production areas, such as 3D modelling, animation, programming, project management and version control. Author's own Unity 3D Pro licence is used in this project. Free open source applications are favoured in the selection of other needed tools. As a result of this thesis, a prototype based on game design was developed, which has passed usability testing and customer survey.

The goal with customer survey was to gain feedback of the game and determine a suitable age group for the game. Based on the usability testing and customer feedback further development plans were made. The testers are seen as potential customers and the relevant information was collected for business plan. The thesis began by introducing game development related theories and terms. Secondly is presented the tools used in the project and after that project is gone through step by step. In the end will be present the test cases and the gained results.

The continuous expanding of the project was kept in control with good project management and documentation. Ideas that couldn't make it to the prototype were well documented for later use. Positive feedback and new ideas were received during test playing. In overall the project was a success. The prototype will be further developed into a commercial version, which will be released for PC, Mac and web browsers. The ultimate goal of this thesis was to start the author's own indie game development studio

Key words games, game design, level design, game development, unity 3d, video games, 3d modeling
Special remarks Thesis includes a multimedia presentation.

**Alkusanat**

Tämä opinnäytetyö tehtiin Rovaniemen ammattikorkeakoululle. Haluan kiittää ohjaavaa opettajaa Karlsson Kennethia, pelin testaukseen osallistuneita henkilöitä, sekä opiskelijatovereitani Aku Shemeikkaa ja Samu Poikajärveä. Haluaisin myös kiittää lapsuudentoveriani Sami Tuomasta ja elämänkumppaniani Ana Lorenaa. Suuret kiitokset tuestanne ja kärsivällisyydestänne.

# SISÄLLYSLUETTELO

# TAULUKKOLUETTELO



# KUVIOLUETTELO

# KÄSITELUETTELO

**Alustariippumattomuus**
Sovellusta voidaan käyttää useammalla alustalla kuin yhdellä.

**Asset**
Assetteja ovat 3D-mallit, animaatiot, tekstuurit, skriptit ja äänitiedostot.

**CLR**
Common Language Runtime on ajonaikainen ympäristö. Se lataa sovellukset, suorittaa niille JIT-käännöksen, huolehtii suoritusoikeuksista, hallitsee muistin ja kutsuu käyttöjärjestelmän palveluja. ( Haukilehto 2003, 9.)

**Coroutine**
Vuorottelualiohjelma. Vuorotteluohjelmien suoritus jatkuu, kun aikaisempi on suoritettu loppuun asti tai keskeytetty.

**CPU**
Tietokoneen osa, joka suorittaa konekielellä käännettyjä käskyjä.

**Draw Call**
Tarkoittaa renderöintipyyntöä grafiikkapiirille.

**Frame rate**
Kuvataajuus, jonka arvo kuvaa, montako kertaa sekunnissa kuvaa pystytään näyttämään.

**GNU**
GNU Not Unix on Unixin kaltainen käyttöjärjestelmä

**GNU GPL**
GNU General Public License on ilmaisten sovellusten lisenssi, jonka on kirjoittanut Richard Stallman GNU-projektille.

**GPU**
Tietokoneen osa, joka on erikoistunut grafiikkojen renderöintiin.

**IDE**
Integrated Development Environment. Integroitu kehitysympäristö on sovellus, jossa voidaan ohjelmoida.

**Iteroida**
Iterointi on yleinen nimitys menetelmille, joissa samoja työvaiheita toistetaan kunnes haluttu tulos on saavutettu (Wikipedia 2010a).

**jit**
Just in time tarkoittaa ajonaikaista ohjelman kääntämistä eli ohjelmaa tai sen osaa ei ole käännetty konekielelle etukäteen.

**Komentosarja**
Ohjelmointikielen lause (Script). Tiedostopääte riippuu käytetystä ohjelmointikielestä.

**Käyttöliittymä**
Sovellusta käytetään käyttöliittymän kautta. Käyttöliittymä on se, mitä käyttäjä näkee ja minkä avulla käyttäjä vaikuttaa sovellukseen.

**OSI**
Open Source Initiative. Organisaatio, jonka tavoitteena on tukea avointa lähdekoodia.

**Prefab**
Prefab on kokonaisuus, joka voi sisältää esimerkiksi 3D-mallin ja komentosarjoja. Kun Prefabeistä tehdään useita kopioita, voidaan kaikkiin päivittää uudet asetukset, muuttamalla alkuperäistä prefabiä. Prefabeistä voidaan luoda ilmentymiä pelinaikana.

**Raycasting**
Raycasting on tietokonegrafiikassa käytetty menetelmä, jolla voidaan tehdä laskelmia käyttäen sädettä. Raycasting menetelmällä voidaan esimerkiksi laskelmoida varjoja objekteihin, käyttämällä valonlähdettä lähtöpisteenä.

**Skriptikieli**
Skriptikieli (Scripting language) on yleisesti ottaen ohjelmointikieli, jota käytetään muokkaamaan, käsittelemään ja automatisoimaan toimivan järjestelmän tarjoamia palveluita (Peltomäki 2001, 7).

**Suodin**
Suodin (shader) on renderöintiin vaikuttava osa, jolla tehdään renderöintiefektejä.

**Suoratoisto**
Suoratoisto (streaming) on tiedonsiirtotapa, jonka avulla voidaan näyttää käyttäjälle ladattu tieto ennen kuin koko tiedosto on ladattu.

**Valokartta**
Valokartat ovat pintojen ennalta laskettuja kirkkaus arvoja, joiden avulla voidaan nopeuttaa staattisten objektien renderöintiä.

**VRAM**
Video Random Access Memory, on GPU:n omaa muistia.

# 1 JOHDANTO

Tässä opinnäytetyössä suunniteltiin ja toteutettiin peli Unity 3D-ympäristössä. Aiheen valinnan taustalla on kiinnostukseni pelialasta, joka on kasvanut lapsuudesta asti. Tutustuin pelimoottoreihin toiseksi viimeisenä opiskeluvuotenani ja olin alustavasti päätynyt Torque3D-pelimoottoriin, mutta onnekseni minulle suositeltiin Unity 3D-pelimoottoria. Unity 3D tuki tuttuja ohjelmointikieliä ja sisälsi mielestäni parempia ominaisuuksia.

Aloitin pelimoottoriin tutustumisen hyvissä ajoin. Käytin siihen kolmannen vuoden kevään ja kesälomani. Tässä vaiheessa en ollut vielä päättänyt, millaisen pelin tekisin. Keväällä vielä ajattelin että tekisin pienen kännykkäpelin. Kunnes pelimoottori alkoi tulla tutuksi, aloin tuntea, että voisin tehdä jotain suurempaa. Kun koulu jatkui syksyllä, opiskelijoita pyydettiin pohtimaan opinnäytetyön aiheita. Sillä kovinkaan monella ei ollut aavistusta, mistä aiheesta tekisivät opinnäytetyönsä. Tällöin päätin, että otan ideavihostani aiheen ja teen siihen pohjautuen pelisuunnitelman ja prototyypin pelistä. Tämä aihe oli yksi kunnianhimoisimmistani, mutta tunsin että nyt on sille sopiva aika. Tiesin että projekti onnistuisi, kunhan aloitan sen tekemisen välittömästi ja hyödynnän mahdollisimman paljon kesällä keräämiäni prefabejä sekä komentosarjoja.

Työssä tutustutaan pelikehityksen teoriaan ja käytäntöön Unity 3D-ympäristössä. Teoriassa tutustutaan pelisuunnitteluun, kuten hahmonsuunnitteluun ja kenttäsuunnitteluun. Tämän jälkeen tutustutaan pelikehityksen olennaisiin työkaluihin, kuten 3D-mallinnusohjelmaan, kuvankäsittelyohjelmaan, sekä pelimoottoriin. Työkalujen valinnassa pyritään suosimaan ilmaisia ja avoimen lähdekoodin -sovelluksia.

Työn päämääränä on toteuttaa toimiva prototyyppi suunnitellusta pelistä ja tämän jälkeen testata ja iteroida peliä. Kun peli on hiottu oman arvion mukaan paremmaksi, järjestetään pelille kaksi testitapahtumaa RAMK:ssa. Ensimmäinen testaus tulee olemaan käytettävyystestaus, jonka tavoitteena on löytää pelistä käytettävyysongelmia ja hioa peliä eteenpäin. Käytettävyystestauksessa kerätään ryhmä opiskelijoita testaamaan peliä,

jonka jälkeen he täyttävät suunnittelemani lomakkeet käytettävyydestä. Tämän jälkeen järjestetään pelattavuustestaus, jossa yksi henkilö kerrallaan testaa peliä. Testauksen aikana seuraan pelin kulkua ja pelaajan kehonkieltä, sekä esitän muutamia kysymyksiä ja teen muistiinpanoja. Pelattavuustestauksen aikana pyrin olemaan neuvomatta pelaajaa ja pysymään vaiti, paitsi silloin kun esitän hänelle kysymyksiä pelistä. Muistiinpanoista kirjoitan peliä varten jatkokehityssuunnitelman, jossa pyrin listaamaan hyvät ja huonot ideat, joita pitää karsia tai jatkokehittää.

Opinnäytteen prosessi alkaa ideoinnilla. Pohdin mikä voisi toimia pelissä ja tämän jälkeen suunnittelen miten ja milloin ideat toteutetaan. Suunnittelun tuloksena syntyy synopsi ja suppea pelisuunnitelma, joita jatkokehitetään kolmen version verran. Tämän jälkeen peliä aletaan toteuttaa suunnitelmien pohjalta. Ensimmäiseksi luodaan pelin kenttä, jonne kaikki tapahtumat sijoittuvat. Maisema tulee olemaan pohjoinen ja vuodenaikana on talvi. Kentän pinta-alasta suurin osa käytetään tunturiin, metsiin, sekä keskellä sijaitsevaan järveen. Pelin kentän tekemisvaihe on yksi tärkeimmistä, sillä se luo suuren osan pelin vaikutelmasta.

Tämän jälkeen aloitetaan ohjelmointi ja 3D-grafiikan luonti. Peliä pyritään tekemään iteroiden ja nopeasti prototypoimalla. Tärkeimmät 3D-mallit ovat järvi, talot ja pelaajan hahmo. Komentosarjoista tärkeimmät ovat pelaajan liikkumiseen vaikuttavat skriptit, ovien aukaiseminen, sekä kaupan toiminnot. Tarkoituksena on saada perustoiminnot peliin, jonka jälkeen peliä on helppo jatkokehittää suunnitelmien mukaisesti. Suunnitelmiin kuuluu muun muassa kauhuelementtien lisääminen peliin, mutta todennäköisesti ne jätetään jatkokehitykseen.

Opinnäyte jakaantuu viiteen osaan: Ensimmäisessä osassa käydään läpi pelikehitykseen liittyviä teorioita, sekä tutustutaan ohjelmistoalalle tuttuihin käsitteisiin, kuten käytettävyyteen ja avoimeen lähdekoodiin. Seuraavassa osassa esitellään peliprojekti ja kerrotaan sen vaiheista ja siinä käytetyistä työkaluista. Kolmanneksi käydään läpi testitapahtuma, joka pohjautuu peliprojektiin ja pohditaan saatuja tuloksia. Tämän jälkeen esitellään valmistunut prototyyppi ja, käydään läpi sen sisältöä.

# 1 PERUSKÄSITTEITÄ

## 1.1 Pelisuunnitelma

Pelisuunnitelma on dokumentti, jonka tarkoituksena on nopeuttaa tuotteen toteutusta ja välttää sudenkuoppia, jotka ovat etukäteen nähtävissä suunnittelemalla. Pelisuunnitelmista on dokumenttipohjia, mutta niissä tulee ottaa huomioon, että ne ovat pelikohtaisia ja eivät välttämättä sovi juuri tähän projektiin. Tällöin on parasta laatia oma dokumenttipohja ja pohtia mikä on hyödyllistä kirjoittaa ylös ja mikä ei. Kuten elokuvateollisuudessa suunnitelmista on olemassa lyhyt ja pitkä versio. Synopsi on lyhyt tiivistelmä, josta saa nopeasti käsityksen millaisesta tuotteesta on kyse. Tämän lisäksi on suppeapelisuunnitelma, joka on laajempi dokumentti, jossa käydään läpi kaikki oleelliset yksityiskohdat.

### 1.1.1 Gameplay

Gameplay on subjektiivinen termi, mutta minulle se tarkoittaa mitä pelaaja kokee pelatessaan peliä ja miten pelaaja voi vaikuttaa peliin. Gameplay usein rinnastetaan pelattavuuteen, mutta itse pelattavuus ei kuvaa kaikkea mitä gameplay on. Pelattavuus sisältää useita määritteitä, kuten kuinka helppo uuden pelaajan on oppia pelaamaan peliä ja kuinka peli motivoi pelaajaa, sekä millaisia tunteita peli herättää pelaajassa. Monet näistä käsitteistä tulevat tarkemmin tutuksi, kun käsitellään käytettävyyttä. Pelialalla on todettu, että hyvällä gameplayllä voidaan korvata huonoa tarinaa, mutta hyvällä tarinalla ei voida korvata huonoa gameplaytä.

### 1.1.2 Tarinansuunnittelu

Mikäli tarina liittyy tuotteeseen, on tärkeää suunnitella, kuinka tarina kerrotaan. Tarinan suunnittelussa käytetään useasti kuvakäsikirjoitusta, joka on kuin sarjakuva. Kuvakäsikirjoituksessa edetään pelin alusta pelin päämäärää kohti. Yksittäinen kuva voi esittää yhtä kenttää tai vaikkapa tilannetta, joka tapahtuu tässä vaiheessa peliä. Tarinan kirjoittamisessa kuvataan siihen liittyvät henkilöt, tapahtumapaikat ja päätapahtumat.

### 1.1.3 Kenttäsuunnittelu

Kenttäsuunnittelussa kirjoitetaan tai piirretään, missä kukin esine sijaitsee. Suunnittelun tarkoituksena on saada pelaaja ohjattua oikeaan suuntaan pelissä ja saada kaikki tarvittava sopimaan kenttään ilman, että se näyttäisi ahtaalta tai tyhjältä. Kun käytetään nopeaa prototypointimenetelmää, kuten Scrumia tai eXtreme Programmingia, kenttäsuunnittelu helpottuu ja iterointi nopeutuu. Testaamalla ideoita paperilla karkeasti on nopeampaa kuin niiden toteuttaminen 3D-ympäristössä. Suunnittelulla vältetään lähes aina sudenkuoppia, kuten tilan loppumista kentästä tilanteessa, jossa sinne pitäisi sijoittaa vielä muutama rakennus tai alue. Tämä tuli minulle vastaan tehdessäni kentästä prototyyppejä. Ensimmäisissä iteraatioissa kenttäsuunnittelu oli vanhentunut muuhun pelisuunnitelmadokumenttiin nähden. Uuden pelisuunnitelman mukaan kentässä tuli sijaita enemmän rakennuksia, kuin kenttäsuunnitelmassa oli varattu niille tilaa. Nopeasti kuitenkin huomasin, että säästän useita tunteja piirtämällä ensin paperille ja vasta tämän jälkeen mallintamalla kenttää Unity 3D:ehen.

### 1.1.4 Tavoitteiden suunnittelu

Peliin suunnitellaan tavoitteita, joita pelaajan tulee suorittaa. Mikäli pelaajan ei haluta juoksentelevan vapaasti koko kentässä pelin alkaessa, voidaan peliin asettaa niin kutsuja lukkoja. Lukot rajoittavat pelaajan liikkumista. Yksinkertaisimmillaan tämä voi olla lukko ovessa, johon pelaajan on etsittävä avain tiloista, joihin hän pääsee ennen ovea. Mikäli lukkoja väärinkäytetään esimerkiksi sijoittamalla liian monta samankaltaista lukkoa peliin, saatetaan pilata pelin tunnelma ja tehdä pelistä toistava. Toisaalta parhaimmillaan lukot parantavat pelin tunnelmaa ja tekevät siitä mielenkiintoisemman. Lukko voi olla tällöin tilanne, jossa pitää esimerkiksi ohittaa vartija tulematta huomatuksi. Avain voi olla jotain hyödyllisempää kuin pelkkä avain, kuten työkalu, jota voi käyttää muuhunkin kuin pelkkään lukon avaamiseen. Lukko mekanismi voi olla voimakas ja elintärkeä työkalu pelin suunnittelulle. Mutta kuten kaikkia työkaluja, niitä tulee käyttää järkevästi ja luovasti ollakseen aidosti tehokkaita. (Schilpp 2001.)

### 1.1.5 Tapahtumien suunnittelu

Tapahtumat ovat tilanteita, jotka aktivoituvat peliin esimerkiksi pelaajan saapuessa alueelle, jossa on törmäyksentunnistuslaatikko. Törmäyksentunnistus rekisteröi pelaajan astuneen laatikon sisään. Tapahtumat ovat pelikohtaisia, ne voivat olla esimerkiksi vihollisten sijoittamista alueelle tai välikohtauksen näyttämistä videona. Mikäli tapahtumat toistavat aina samaa kaavaa ja ovat ennalta arvattavia, voi tämäkin ominaisuus pilata pelin tunnelman. Joissakin peleissä näytetään enemmän välivideoita, kuin pelataan itse peliä ja tämäkään ei mielestäni peliä paranna. Mutta kun välivideoita käytetään oikein ja ne tehdään huolella, antavat ne uuden tavan kertoa tarinaa. Yleisemmin kuvattuna tapahtumat ovat pelin perustilanteita, jotka käynnistetään tietyllä hetkellä.

### 1.1.6 Hahmojen suunnittelu

Hahmojen suunnittelulla kuvaillaan, miltä hahmo näyttää ja kuulostaa. Kerrotaan hänen taustansa ja miten hän liittyy tarinaan, mikäli sellainen on. Hahmoja suunnitellaan yleensä luonnostelemalla heidät paperille ja kirjoittamalla dokumenttiin heidän ominaisuuksiaan. Mikäli kyseessä on 3D-peli, voidaan hahmon luonnoksia käyttää taustakuvana 3D-mallinnusohjelmassa ja luoda hahmo mahdollisimman paljon luonnoksensa näköiseksi.

## 1.2 Peliohjelmointi

Peliohjelmoinnilla määritellään kaikki pelin toiminnot, kuten pelaajan liikkuminen. Peliohjelmoinnissa, kuten ohjelmoinnissa yleensäkin pyritään kevyeen uudelleen käytettävään koodiin. Mikäli komentosarjaa kutsutaan useita kertoja sekunnissa ja se sisältää useita toistorakenteita, pelistä tulee raskas ja kuvataajuus laskee. Raskaissa peleissä pelaaja tarvitsee paremman tietokoneen kestääkseen tarpeettomia komentojonojen kutsuja. Peliohjelmointi on mielestäni pelin kehityksessä hauskinta ja sen vuoksi valitsin ohjelmistoalan koulutuksen itselleni. Vaikkakin peliohjelmointi on mielekästä, on se myös lähes kaikkein turhauttavinta ja aikaa vievintä. Toisaalta se on kenties kaikkein palkitsevinta.

## 1.3 Käytettävyys

Käytettävyydellä maksimoidaan tuloksellisuus, tehokkuus ja käyttäjän tyytyväisyys. Käytettävyyttä tutkitaan järjestämällä testitapahtumia ja analysoidaan tuloksia. Tuloksellisuudella laskelmoidaan tehtävän onnistumisprosenttia ja kuinka monesti tiettyä funktiota käytetään. Tuloksellisuuden avulla saadaan selville, kuinka moni testikäyttäjistä on suoriutunut tehtävistä ja mitä funktioita ei välttämättä tarvita suoriutumiseen. Tehokkuudessa mitataan, kuinka kauan käyttäjällä menee suorittaa tietty tehtävä ja kuinka monta toimintoa hänen tulee suorittaa päästäkseen päämääräänsä. Tehokkuudessa myös mitataan kuinka paljon käyttäjä käyttää aikaa help-dokumenttien lukemiseen ja kuinka paljon aikaa menee käyttäjällä virheistä palautumiseen. Käyttäjän tyytyväisyyttä voi olla vaikea mitata, mutta usein sitä pyydetään arvioimaan väliltä yhdestä viiteen. Käytettävyyttä pidetään hyvänä tyytyväisyyden osalta, mikäli yli 90 prosenttia käyttäjistä arvioi tyytyväisyytensä tuotteeseen välille neljästä viiteen. (Faulkner 2000, 35—38.)

Muita käytettävyyden mittareita ovat opittavuus, joustavuus ja asenne. Opittavuudella tarkoitetaan, kuinka nopeasti ja vaivalloisesti käyttäjä oppii käyttämään toimintoja joita mitataan. Opittavuudessa huomioidaan myös muistettavuus eli, kuinka nopeasti käyttäjä uudelleen oppii käyttämään tuotetta pitkän tauon jälkeen. Hyvä opittavuus on myös sitä, että käyttäjän ei tarvitse erikseen oppia käyttämään laitetta, vaan pystyy käyttämään tuotetta suoraan. Joustavuudella tarkoitetaan, kuinka hyvin tuote sopii suunnitellun ympäristön ulkopuolelle esimerkiksi toiselle laitteelle tai toisenlaisiin työtehtäviin. Joustavan ohjelman tulee sopeutua ympäristöön ja tehtäviin, eikä toisinpäin. Tuotetta käytettäessä tulee käyttäjällä pysyä yllä hyvä asenne, eikä ohjelma saa ärsyttää käyttäjää, varsinkaan turhaan. Käytettävyydessä mitataan myös usein virheiden määrää ja toistuvuutta. Virheet yleensä lajitellaan niiden vakavuuksien mukaan, pieni virhe voi häiritä käyttäjää ja saada tämän tekemään enemmän virheitä. Vakavammat virheet häiritsevät enemmän ja kaikkein vakavimmat virheet estävät käyttäjää tekemästä tehtäviään täysin. (Faulkner 2000, 130—131.)

Hyvä käyttöliittymä on yksinkertainen ja selkeä. Painikkeet tulevat hyvin esiin taustastaan ja eroavat toisistaan. Tuotetta on helppo käyttää ja sen oppii lähes välittömästi. Peleissä usein arvostetaan haastetta, mutta haasteiden ei tarvitse tulla hankalasta käyttöliittymästä. Pelaajien on todettu olevan herkkiä käytettävyydelle, koska tarjolla on niin paljon pelejä joista pelaaja voi valita. Pelaaja voi lopettaa pelin ja vaihtaa toiseen nopeasti ja vaivatta, mikäli huono käytettävyys häiritsee peliä.

Ymmärtämällä käyttäjää voidaan tuotetta parantaa mukavammaksi ja käyttäjä todennäköisemmin käyttää tuotetta ja tuote voi siten menestyä paremmin. Työn teko tehostuu, eikä työntekijöitä tarvitse kouluttaa yhtä paljon ja asiakastukeen voidaan panostaa vähemmän. Käytettävyyden suunnittelulla voidaan projektissa välttää sudenkuoppia ja säästää aikaa.

## 1.4 Avoin lähdekoodi

### 1.4.1 Määritelmä

Avoin lähdekoodi on kehitysmenetelmä ohjelmistoille jotka kaipaavat hajautetun vertaisarvioinnin ja avoimen prosessin vahvuuksia. Avoimella lähdekoodilla saadaan parempaa laatua, korkeampaa luotettavuutta, lisää joustavuutta, sekä vähemmän kustannuksia. (OSI 2011.) Ollakseen avoimen lähdekoodin sovellus, tulee ohjelman täyttää OSI-organisaation (2011) asettamat vaatimukset. Vaatimukset ovat seuraavat:

1. Ohjelman täytyy olla vapaasti levitettävissä. 2. Ohjelman tulee sisältää lähdekoodi. 3. Lisenssin tulee sallia lähdekoodin muokkaaminen ja muokatun lähdekoodin levittäminen 4. Lisenssi voi vaatia, että muokattu lähdekoodi tulee nimetä eri tavalla kuin alkuperäinen, jos tiedosto on versioitu päivitystiedosto. 5. Ketään ei saa syrjiä. 6. Käyttötarkoituksia ei saa rajoittaa. 7. Sama lisenssi pätee kaikkiin tästä johdettuihin ohjelmiin. 8. Lisenssin tulee päteä vaikka ohjelma irrotettaisiin toisesta tuotteesta. 9. Lisenssi ei voi kieltää muiden ohjelmien käyttämistä tämän ohjelman kanssa. 10. Lisenssin tulee olla teknologisesti riippumaton.

### 1.4.2 Hyödyt ja haitat

Avoin lähdekoodi on tarjolla kaikille ja sen kehitykseen voi osallistua kuka tahansa. Ohjelman käyttäjät voivat kehittää yhdessä tuotetta paremmaksi ja keskustella alkuperäisen kehitystiimin kanssa. Mikäli ohjelmassa on jotakin parannettavaa, pystytään parannukset tekemään itse, eikä tarvitse ehdottaa ohjelman kehitystiimille ja toivoa että he tekisivät haluamasi päivityksen jonakin päivänä. Avoin lähdekoodi on myös lähes aina ilmaista ja avoimeen lähdekoodiin perustuvia sovelluksia on paljon. Esimerkkisovelluksia mainitaan seuraavassa kappaleessa, työvälineet osiossa.

# 2 Prototyyppi

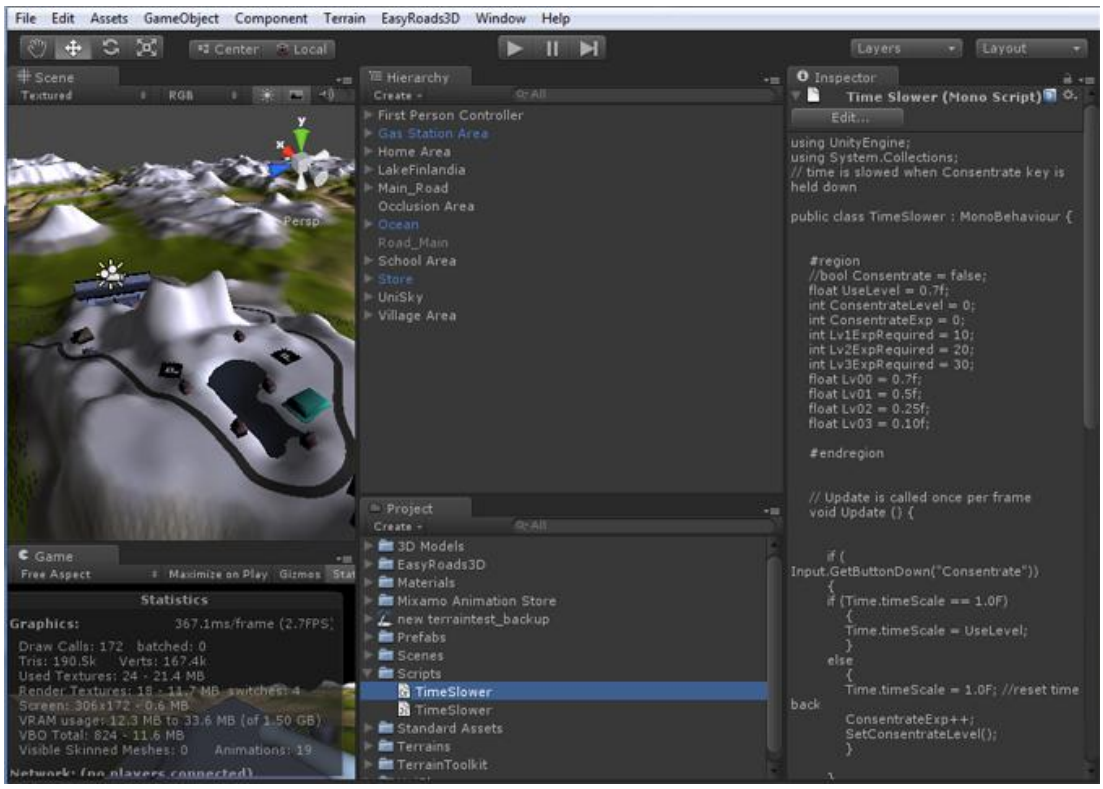
## 2.1 Projektin taustat

Kiinnostukseni pelialaan ajoi minut tähän projektiin, jonka itse suunnittelin alusta asti. Projektin ohjelmistokehitysmenetelmänä oli kahden ensimmäisen kuukauden ajan Rational Unified Process. Ensimmäisinä kuukausina pyrin suunnittelemaan ja dokumentoimaan mahdollisimman paljon. Tämä kuitenkin vei paljon aikaa ja kun lopulta aloin toteuttamaan peliä huomasin, että suunnitelmat eivät toimineet niin hyvin kuin olisin halunnut. Tämän jälkeen suunnittelin uudelleen ja yritin uudelleen, sama toistui useita kertoja. Lopulta luovuin etukäteen suunnitelemisesta ja aloin käyttämään nopeaa prototypointia. Tutustuin eXtreme Programming -menetelmään ja sen avulla pystyin etenemään projektissani tehokkaammin. Varsinkin kentän toteutuksessa jouduin aloittamaan useasti puhtaalta pöydältä ja XP-menetelmä tuki tätä paremmin kuin RUP. Usean prototyypin jälkeen päätin lopulta päivittää suunnitelmadokumenttini ja tässä vaiheessa tiesin, että voisin toteuttaa testaamani ideat peliin. Projektin puolivälissä minulle alkoi hiljalleen selvitä pelituotannon laajuus ja minun tuli varata toteutukseen rutkasti enemmän aikaa, kuin mitä olin alun perin suunnitellut.

## 2.2 Työvälineet

### 2.2.1 Unity 3D

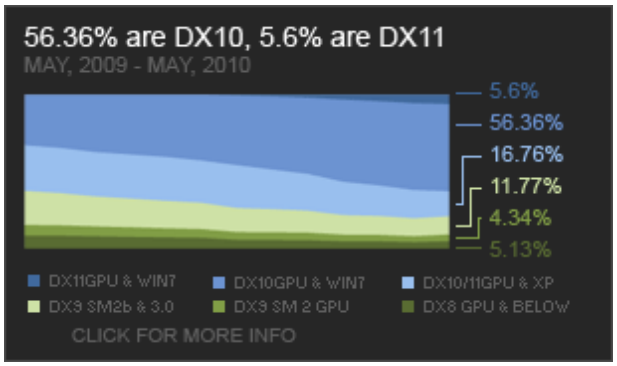
Unity 3D on alustariippumaton 3D-pelimoottori, jolla voidaan kääntää valmis projekti seuraaville alustoille: PC, Web, Mac, Xbox 360, Ps3, Wii, iPhone, iPad, iPod touch, sekä Android. 3D-pelien lisäksi Unityllä voidaan tehdä myös 2D-pelejä, mutta niitä varten tulee hankkia yhteisön tekemiä liitännäisiä. Unity 3D on visuaalinen työkalu, jossa käyttäjä näkee peli-ikkunasta, mitä pelaaja tulee näkemään. Pelin tekeminen tapahtuu vetämällä projektikansion listasta objekteja, kuten 3D-malleja scene-ikkunaan. Tämän jälkeen objekti näkyy hierarkialistassa, jossa näkyy kaikki scene-ikkunan objektit.




Kuva1. Kuva Unityn käyttöliittymästä.

Nyt objekti on lisätty peliin ja sille voidaan antaa toimintoja liittämällä komentosarja-tiedosto siihen. Pelilogiikka toimii .NET alustalla nimeltä mono, jonka tuettuja ohjelmointikieliä ovat C#, UnityScript ja Boo. UnityScript on JavaScriptin kaltainen kieli ja Boon syntaksi on Python-kielen inspiroima. Unitya käyttäessä on kuitenkin suositeltavaa keskittyä yhteen skriptikieleen. Saman kielen käyttö helpottaa varsinkin, kun koodien tulee pystyä kommunikoimaan keskenään. Olen pyrkinyt käyttämään C#-kieltä ja kääntämään JavaScript-esimerkit C#-kielelle. Kuitenkin toisinaan omat komentosarjaliitännäiseni jäävät lyhyemmiksi, kuin mitä valmiit tarjolla olevat komentosarjat ovat ja tällöin kirjoitan oman koodini JavaScript-kielellä.

Unity 3D käyttää Nvidian PhysX-fysiikkamoottoria, joka on standardi alalla. Grafiikat renderöidään DirectX9 tai vastaavaa OpenGL-versiota käyttäen. Unity Technologies ei ole päivittänyt uudempaan versioon DirectX:stä, koska se katsoi että resursseja on parempi käyttää toistaiseksi muualla, kunnes DirectX11 yleistyy ja täten päätti hypätä yli DirectX10-versiosta.




Kuva2. Steam julkaisualustan teettämä DirectX-kysely.

Kuten edellä olevasta kuvasta näkyy vain 5,6 prosenttia Steam-julkaisualustan käyttäjistä omistaa DirectX11-yhteensopivan grafiikkapiirin, mutta toisaalta 56,36 prosenttia käyttäjistä pystyy hyödyntämään DirectX10-versiota. Tässä tulee kuitenkin ottaa huomioon, että Unityllä voidaan julkaista myös selaimille, joissa tulee vastaan myös vanhempia tietokoneita. Kuitenkaan ei tule tehdä sitä johtopäätöstä, että Unityllä voidaan tehdä vain pieniä selainpelejä, jotka toimivat kymmenen vuotta vanhalla tietokoneella. Unity Technologies on pyrkinyt optimoimaan pelimoottoria mahdollisimman paljon, jotta korkealaatuisetkin pelit voisivat toimia vanhemmalla koneella. Optimoitu peli saavuttaa mahdollisimman suuren yleisön, eikä synny kuilua pelaajien välille, jossa syrjittäisiin niitä, jotka eivät päivitä konettansa aina kun tulee uudempi versio tarjolle.

Unityn uusin versio tukee Deferred Renderer-tekniikkaa, jonka avulla rajaton määrä valonlähteitä voi vaikuttaa samanaikaisesti objektin valotukseen. Sen avulla saadaan korkealaatuinen valaistus peliin ja voidaan käyttää huoletta useita valonlähteitä. Valojen määrän lisäksi tekniikka tukee myös reaaliaikaisia varjoja ja valolla voidaan heijastaa tekstuurin muotoja esimerkiksi taskulamppuefektiä varten. Tulee kuitenkin ottaa huomioon, että varjojen laatu vaikuttaa paljon pelin suorituskykyyn. Varjot voivat olla pehmeitä tai kovia ja niille voidaan asettaa piirtoetäisyys, sekä resoluutio. Pehmeät varjot ovat huomattavasti raskaampia renderöidä, mutta ne näyttävät paljon realistisimmilta. Tehokkuuden kannalta tulee ottaa huomioon lopullisen julkaisualustan suorituskyky. Mutta on myös mahdollista laittaa pelin asetuksiin, kumpaa tekniikkaa käytetään. Unityssä grafiikka saadaan loistoonsa sadalla Unityn mukana tulevalla suotimella ja mikäli nämä eivät riitä, käyttäjä voi muokata niitä tai kirjoittaa kokonaan uusia suotimia. Suotimet vaikuttavat siihen miten objektien materiaalit renderöidään.

Materiaalit voidaan tehdä läpinäkyviksi käyttämällä Transparent Shadereitä, jotka lukevat tekstuurin alpha-arvot ja asettavat ne läpinäkyviksi. Unityssä käytetään materiaaleja, kun halutaan asettaa tekstuureja objekteihin. Materiaali sisältää siis tekstuurin ja suotimen. Samaa materiaalia voidaan käyttää useissa objekteissa samanaikaisesti. Tämän jälkeen materiaalia voidaan muokata, ja se päivittyy automaattisesti kaikkiin objekteihin. Materiaaleja ei tarvitse kuitenkaan tehdä manuaalisesti, mikäli 3D-objekti sisältää materiaalin tuotaessa Unityyn. Unityn mukana tulee myös laaja paketti jälkiprosessointiefektejä, joita voisi yksinkertaisemmin kutsua kameraefekteiksi. Näiden avulla saadaan elokuvaefektejä, joita videoihin lisätään jälkeenpäin. Kamerat ovat pelaajan silmät pelissä ja on hienoa, että efektejä voidaan esittää reaaliajassa. Esimerkiksi auringonsäteet saadaan aina näkymään oikeassa kulmassa pelaajaan. Kameran tarkennusefektillä voidaan sumentaa tietyn alueen ympäriltä ja kiinnittää pelaajan huomiota.

Unityn maastoeditori on sisäänrakennettu, jolloin kenttää tehtäessä tulokset näkyvät välittömästi scene-ikkunassa. Kenttää tehdään käyttämällä seuraavaa työkalupalkkia, joissa lähes kaikissa on asetus siveltimen koolle ja tehokkuudelle.

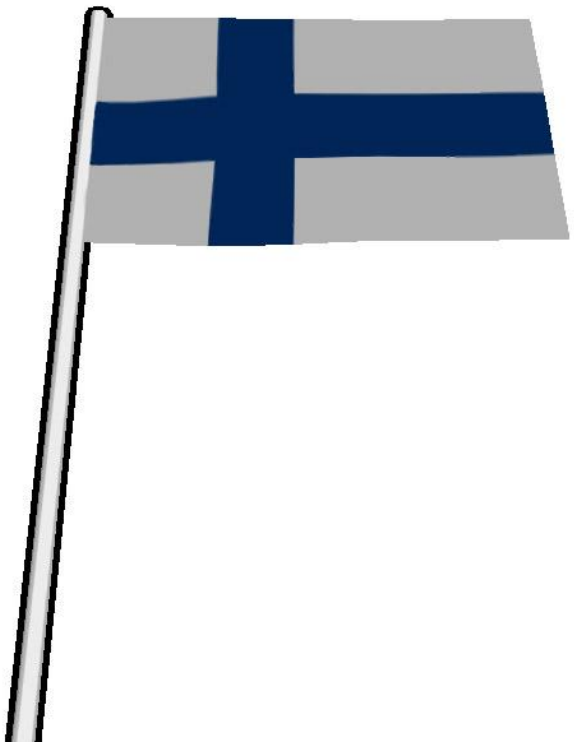
Kuva3. Terraineditorin pääpalkki

Edellä olevan työkalupalkin toiminnot ovat seuraavat. Ensimmäisellä vasemmalta katsoen voidaan nostaa tai laskea maastoa. Toisella voidaan nostaa tai laskea maastoa halututtuun korkeuteen asti. Kolmannella työkalulla tasoitetaan maaston korkeuseroja. Siveltimen kuvaa, joka on neljäntenä, käytetään tekstuurien maalaamiseen maastoon. Siveltimessä on uutena asetuksena läpinäkyvyysarvo, jonka avulla voidaan maalata useita kerroksia eri tekstuureilla. Kaksi seuraavaa toimii samalla tavalla, näihin asetetaan yksi tai useampi objekti, joita voidaan maalata maastoon kiinni. Ensimmäinen näistä on puita varten. Siihen voidaan asettaa arvoja, joilla voidaan tehdä puista erikokoisia ja näköisiä. Tällöin puut eivät tule olemaan saman puun näköisiä. Viimeisenä on hammasrattaan kuva, jota käytetään usein asetuksia varten, kuten tässäkin tapauksessa. Asetuksissa voidaan

asettaa maastoon vaikuttavia arvoja, kuten piirtoetäisyyksiä tai alueen puita heiluttava tuuli.

Nvidian PhysX on integroituna Unityyn ja täten tarjolla on kaikki PhysX-version fysiikkakirjastot. Fysiikka toiminnoista yleisin käyttämäni on Rigidbodies, joka liitetään objektiin ja tämän jälkeen tälle voidaan antaa fyysisiä arvoja, kuten massa, ilmanvastus ja se vaikuttaako painovoima objektiin. Rigidbodiesin liittämisen jälkeen fysiikkasimulointi vaikuttaa objektiin, mikäli tämä lakkaa olemasta levossa. Rigidbodiesin avulla saadaan aikaan hienoja törmäyksiä objektien välille. Fysiikkoihin pääsee käsiksi myös komentosarjojen kautta, jolloin voidaan asettaa esimerkiksi työntövoima pelaajan eteen ja tämän jälkeen pelaaja voi alkaa työntämään tarpeeksi kevyitä objekteja. Muita fysiikkatoimintoja kuin Rigidbodies ovat Joints, Soft Bodies, Ragdolls, Cars ja Cloth. Nivelten avulla saadaan niveliä esimerkiksi kettinkiin tai pelaajan raajoille. Soft Bodiesin avulla voidaan tehdä pehmeitä objekteja, kuten vajaa rantapallo, jonka fysiikkamoottori pitää oikeassa muodossa riippuen törmäyksistä. Ragdolls-velhon avulla voidaan muuttaa täysin animoitu hahmo hetkessä räsynukeksi. Räsynukkeja käytetään useasti peleissä, joissa halutaan saada hahmo kaatumaan realistisesti. Cars-fysiikoilla saadaan auto kulkemaan tiellä.

Tämän opinnäytetyön aikana Unityn yhteisöllä oli kilpailu, jossa palkittiin kaksi parasta autofysiikkaprototyyppiä. Prototyyppien tuli olla helposti siirrettävissä muihin projekteihin ja muokattavissa eri ajoneuvoille. Yhteisö palkitsi kaksi parasta, joille jaettiin yhteensä 6150 Yhdysvaltain dollaria. Kilpailun päätteeksi kaikki, jotka olivat osallistuneet tukemaan projektia, saivat lähdekoodit dokumentoituna ja esimerkkiprojektit haltuunsa. Olin mukana tukemassa tätä projektia ja seurasin sitä puolesta välistä loppuun asti. Tulokset ylittivät kaikki oletukseni, jotka perustuvat siihen, mitä olen nähnyt peleissä viime vuosina. Viimeisenä, mutta ei suinkaan vähäisimpänä Cloth, josta on tarjolla kaksi eri vaatefysiikkamallinnus-versiota. Ensimmäinen on interaktiivinen vaate, joka sopii esimerkiksi lippuihin tai roikkuviin lakanoihin. Toinen on optimoitu versio ensimmäisestä, joka sopii parhaiten animoidun hahmon vaatteisiin ja hiuksiin.



Kuva4. Vaatefysiikoiden simulointi pelissä

Unity 3-versiosta lähtien Unityssä on ollut integroituna tehokkain tarjolla oleva äänimoottori FMOD. Se tukee useimpia äänitiedostoformaatteja ja on alustariippumaton. Unityn selainpeleille on tärkeää pieni tiedostokoko, jolloin äänitiedostojen suoratoisto internetistä on tärkeä ominaisuus. Suoratoistolla voidaan pienentää selaimella pelattaessa, käynnistyksen yhteydessä tapahtuvaa tiedostojen lataamista. Äänitiedostot ladataan pelin aikana tarvittaessa. FMOD:lla on mahdollista kuunnella pelin ääniä käynnistämättä peliä. Tämä helpottaa, sekä nopeuttaa äänien säätämistä. FMOD sisältää paljon äänisuodattimia, joilla voidaan luoda nopeasti ääniefektejä, muokkaamatta ja kopioimatta äänitiedostoa. Äänisuodattimia on tarjolla muun muassa Echo, Reverb, Distortion, Chorus, sekä High/Low pass. Echo-suodattimella saadaan aikaan toistuva kaikuefekti. Echo-suodattimessa voidaan säätää kaiun viivettä, vaimenemisnopeutta, sekä paria muuttujaa, joilla muokataan jokaista kaikua erilaisiksi. Luonnossakaan ääni ei ole eristyksissä ympäristöstään, eikä sen tarvitse olla peleissä. Reverb-suodatin muokkaa äänen ympäristöön sopivammaksi. Äänen muokkausta ympäristöönsä auttaa Reverb Zones, joka on alue, jossa vaikuttaa tietty Reverb-suodatin. Reverb Zones asettaa Reverb-suodattimen tehokkuuden riippuen pelaajan Audio Listener -komponentin etäisyydestä alueeseen.

Distortion-suodatin vääristää ääntä ja saa sen kuulostamaan vähemmän studiomaiselta. Chorus-suodatin toistaa äänen useita kertoja pienillä muutoksilla ja saa aikaan kuoromaisen äänen. High/Low pass -suodatin muokkaa ääntä riippuen pelaajan sijainnista. Sillä saadaan aikaan etäisyydentunne äänilähteestä. High/Low pass -suodattimella voidaan myös suodattaa korkeita ääniä, jotta ääni kuulostaisi vaimeammalta esimerkiksi oven takaa. FMOD:in avulla pelissä voidaan säästää tiedostokokoa ja kaistanleveyttä muokkaamalla muutamaa äänitiedostoa useiksi ääniksi.

Unity on kirjoitettu C++-kielellä, mutta käyttäjän komentosarjat ja pelilogiikka toimii avoimen lähdekoodin .NET-kehitysalustalla Mono. Mono on alustariippumaton C#-kielen ja CLR:in kokoonpano, joka on binääri yhteensopiva Microsoft.NET:in kanssa. Unity tukee kolmea skriptikieltä, jotka ovat keskenään yhteensopivia ja yhtä tehokkaita. Silti on suositeltavaa keskittyä yhteen ohjelmointikieliin mikäli mahdollista. Unity kääntää kaikki komentosarjat .NET dll -tiedostoiksi, jotka jit-käännetään ajonaikana. Tämä tarkoittaa, että komentosarjat käännetään vasta tarvittaessa. Tästä tekniikasta johtuen, jotta komentosarjojen keskenään kommunikointi toimisi, tulee ohjelmoijan varmistaa ennen kuin komentosarjaa kutsutaan, että kutsuttava komentosarja on käännetty. Unity kääntää kaikki komentosarjat tässä järjestyksessä. Ensimmäisenä käännetään komentosarjat, jotka ovat kansioissa Standard Assets, Pro Standard Assets tai Plugins. Ensimmäisessä vaiheessa käännetyt komentosarjat eivät pysty suoraan kommunikoimaan muihin komentosarjoihin, mutta ne pystyvät kommunikoimaan näiden kansioiden sisällä olevien komentosarjojen kanssa. Ulkopuolisiin komentosarjoihin voidaan lähettää tietoa käyttämällä GameObjet.SendMessage-metodia. Seuraavaksi käännetään ”Editor” nimiset -kansiot, jotka sijaitsevat ensimmäisen vaiheen kansioissa. Editor-kansiossa sijaitsevat komentosarjat käyttävät UnityEditor-nimiavaruutta. Nämä tiedostot ovat työkaluja, joita käytetään Unityn sisällä. Unityä voidaan laajentaa kirjoittamalla omia työkaluja, joilla voidaan tehdä jotain uutta tai nopeuttaa yleisiä toimintoja. Kolmanneksi käännetään Editor-kansiot, jotka sijaitsevat muualla kuin edellämainituissa kansioissa. Standard Assets, Pro Assets ja Plugins ovat Unityn mukana tulevia paketteja. Selvyyden vuoksi voidaan tehdä oma kansio uudelle työkalulle ja tehdä sen sisään Editor-kansio, johon

tulee kaikki komentosarjat. Tällä tavalla pysyvät standardipaketit erillään muista tiedostoista. Neljäntenä ja viimeisenä käännetään kaikki muut komentosarjatiedostot. Kommunikointi tapahtuu kääntöjärjestyksessä aikaisempiin komentosarjoihin, eli komentosarjoja kutsuvan komentosarjan tulee sijaita vähintään yhtä korkealla kääntämisjärjestyksessä. Unityssä coroutinet ovat yksinkertaisia toteuttaa käyttämällä yield-metodia seuraavasti:

```
public class example : MonoBehaviour {
IEnumerator Do()
{
   print("Do now");
   yield return new WaitForSeconds(2);
   print("Do 2 seconds later");
}

IEnumerator Awake()
{
   yield return StartCoroutine("Do");
   print("Also after 2 seconds");
   print("This is after the Do coroutine has finished execution");
}
}
```

(Unity 2011).

Yield-coroutinen avulla voidaan viivästyttää ohjelman suoritusta, käyttämällä WaitForSeconds(arvo)-metodia, johon syötetään sekuntien määrä. Unity pystyy automaattisesti synkronoitumaan MonoDevelopin ja Visual Studion kanssa. Tämä tarjoaa automaattisen täydennysominaisuuden kumpaankin kehitysympäristöön. Unityyn on myös integroitu debuggaus ominaisuus, jolla peli voidaan pysäyttää ja koodin suoritusta tutkia. Esimerkkikoodeja esitän toteutusosiossa myöhemmin.

Unity tukee reaaliaikaisia verkko-ominaisuuksia, peliobjektin sijainti, nopeus, animointi ja kaikki muu voidaan synkronoida muiden pelaajien kesken käyttämällä deltapakkausalgoritmia. Clientien välinen funktioiden kutsuminen, tapahtuu ilman erillistä verkkoneuvottelua. Unityn web-alustalle julkaistut pelit

pystyvät kommunikoimaan täysin julkaisu web-sivunsa kanssa. Kommunikointiin voidaan käyttää JavaScriptiä, sekä AJAX-menetelmiä. Unity pystyy myös yhdistämään web-sivuihin ja web-palveluihin. Web-sivuihin yhdistäminen tukee synkronista ja asynkronista tapaa. Moninpelejä varten pelinkehittäjän tulee kuitenkin ymmärtää verkkopelaamisen peruskäsitteet. Verkkopelaamista varten tarvitaan client ja server -sovellus. Serveri voi olla erillisellä koneella, joka kommunikoi pelaajien kesken. Toinen mahdollisuus on, että yksi pelaajista toimii serverinä ja kaikki muut pelaajat yhdistävät tähän koneeseen. Client on käyttäjän sovellus, tässä tapauksessa peli, joka yhdistää serveriin.

Servereitä on kahdenlaisia, auktoritatiivinen ja ei-auktoritatiivinen. Molemmat lähestymistavat perustuvat clientin yhdistämistä serveriin ja näiden väliseen kommunikointiin. Samalla molemmissa tapauksissa clienteillä on yksityisyys suojattuna, clientit eivät koskaan kommunikoi suoraan keskenään. Auktoritatiivisessä serverissä serveri hoitaa kaiken simuloinnin ja sääntöjen noudattamisen, sekä pelaajien syötteiden käsittelyt. Peli on synkronoituna serverin kanssa. Client ei koskaan käsittele pelin logiikkaa. Client kertoo serverille, mitä halutaan tehdä ja serveri käsittelee pyynnöt ja toteuttaa käskyt. Tässä tulee ero sille, mitä pelaaja haluaa tehdä ja mitä lopulta tapahtuu.

Auktoritatiivisessä serverissä huijaaminen on paljon vaikeampaa clienteille, koska pelaajilla ei ole oikeuksia kertoa serverille mitä tehdä. Pelaajat voivat ainoastaan ilmoittaa niitä asioita, joita he pystyvät tekemään. Ongelmana tässä tekniikassa on, että pelaajien liikkeet eivät tapahdu ennen kuin serveri on käsitellyt syötteet. Tätä voidaan kuitenkin välttää käyttämällä Client side prediction -tekniikkaa. Tässä tekniikassa client tekee laskelmoinnit itse, jotka serveri korjaa ainoastaan, mikäli laskelmat ovat vääriä. Tällä tavalla hahmo voi liikkua koko ajan, myös serverin ollessa hidas ja pelaajan sijainti päivitetään heti kuin mahdollista. Tästä voi johtua pelaajien siirtyminen aikaisempiin askelmiinsa, jos serveri suorittaa komentoja peliä hitaammin. Ei-auktoritatiivinen serveri ei hallitse pelaajan syötteiden lopputulosta. Clientit prosessoivat omat syötteensä ja pelilogiikan omassa ympäristössään. Tämän jälkeen ne lähettävät lopputulokset serverille, jotka serveri synkronoi kaikille.

Tämä menetelmä on helpompi toteuttaa suunnittelun näkökulmasta, koska serveri vain välittää tietoa clientien kesken. Tässä menetelmässä ei myöskään tarvitse käyttää Cliet prediction -menetelmiä, mutta tietoturva on tässä heikompi. (Unity 2010a.)

Unity tarjoaa verkkotestaamista varten työkaluja, jotta esimerkiksi peliin yhdistäminen voidaan testata eri yhteysnopeuksilla. Servereiden ja clientien välillä tulee lähettää ainoastaan tärkeää dataa, jotta saadaan säästettyä siirtokaistaa. Siirtokaistaa voidaan myös säästää tekemällä clienteistä mahdollisimman itsenäisiä. Esimerkiksi serveri lähettää käskyn ladata uuden kentän ja client tekee latauksen jälkeen perustoimintoja, joita ei tarvitse käsitellä serverillä. Unity ei sisällä Massive Multiplayer Online -teknologiaa, mutta sitä varten on tarjolla useita väliohjelmistoja, jotka on integroitu toimimaan Unityssä. Väliohjelmistoja ovat muun muassa Electrotank Universe Platform, Photon Socket Server, sekä Smartfox Server. Näiden lisenssit riippuvat, montako käyttäjää tulee yhdistämään serveriin. Hinnat ovat kohtalaisia myös pienille studioille.

Aloitan optimoinnin käsittelyn seuraavassa järjestyksessä. Aluksi esittelen kuinka optimoida komentosarjojat ja tämän jälkeen kuinka optimoida renderöintiä. Unityssä koodi voidaan kirjoittaa kolmen funktion sisään, joita suoritetaan eri aikoina. Awake-funktio suoritetaan, kun komentosarjasta on luotu ilmentymä. Start-funktio suoritetaan Awake-funktioiden jälkeen, mutta ennen ensimmäistä Update-funktiota. Update-funktio kutsutaan jokaisessa kehyksessä. FixedUpdatea käytetään, kun vaikutetaan RigidBodyyn. FixedUpdate kutsutaan jokaisessa piirtokehyksessä, ellei tiheyttä muuteta. Komentosarjoja voidaan optimoida noudattamalla seuraavia ohjeita: Ei tule laittaa GameObject.Find(), eikä gameObject.GetComponent()-funktioita Update()-metodin sisään. Nämä ovat raskaita metodeja varsinkin kun käytössä on useita peliobjekteja. Näillä voidaan kommunikoida toisen peliobjektin kanssa. Tämä kommunikointi linkki kannattaa tehdä Start()-funktiossa. Toinen ohje on olla käyttämättä Unityn OnGUI-metodia itse pelissä. Se on hyvä GUI-metodi menujen tekemiseen, mutta liikkuvassa pelissä se kuluttaa paljon kuvataajuutta. Kolmas ohje on keventää automaattisen roskienkeruun taakkaa välttämällä ilmentymien luomista

Update()-funktion sisällä. On suositeltavaa ilmentää objektit start-funktiossa ja siirtää ne pelialueelle tarvittaessa, ilmentämisen sijaan. Sen jälkeen kun objektia ei enää tarvita, se siirretään takaisin kentän ulkopuolelle ja voidaan käyttää uudelleen. Eli luodaan mahdollisimman paljon etukäteen ja kierrätetään, mikäli objektia tarvitaan uudelleen. (Ante 2007).

Muita optimointivinkkejä on harventaa FixedUpdaten kutsumista Unityn projektiasetuksista. Käytä Structia Allocaten sijaan, koska struct ei varaa muistia vaan asettuu pinoon, jonka käsittely on tehokkaampaa. Mikäli joudutaan käyttämään muistin varaamista, on suositeltavaa kutsua automaattista roskienkeruuta esimerkiksi pelin latausaikoina. Automaattista roskienkeruuta kutsutaan kirjoittamalla System.GC.Collect()-metodi, tätä on hyvä oppia käyttämään oikein. Unity optimoi JavaScriptin konstruktorit automaattisesti staattisiksi, mutta on suositeltavaa että asetetaan arvojen tyypit itse. Aina Unity ei osaa muuttaa konstruktoria staattiseksi vaan se jää dynaamiseksi. Mikäli tätä halutaan välttää, voidaan kirjoittaa #pragma strict komentosarjan yläosaan, jolloin Unity ei muodosta lainkaan dynaamisia tyyppejä, vaan antaa käännösvirheen.

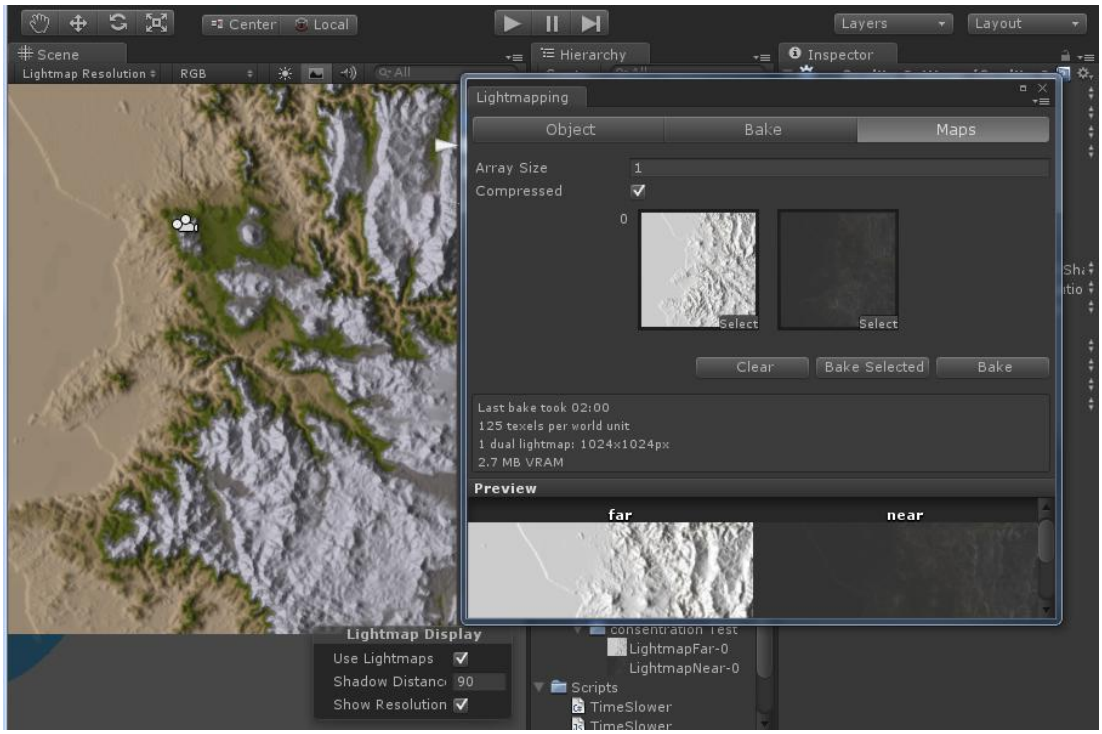
Komentosarjoja voidaan myös optimoida välttämällä dynaamisia taulukoita, jolloin tulee vain tietää taulukolle sopiva maksimi koko. Seuraava lähes itsestään selvä komentosarjojen optimointivinkki on olla kutsumatta funktioita, joita ei ole pakko. Tämä voidaan implementoida seuraavasti: tarkastetaan pelaajan sijainti objektista ja asetetaan objekti nukkumistilaan, kunnes pelaaja on lähettyvillä. Komentosarjojen optimoinnin lopuksi, esittelen vielä kuinka monta sykliä tarvitaan matemaattistenfunktioiden suorittamiseen. Ante (2007) kertoo, että plus, miinus ja kertominen vaativat vain muutaman syklin. Jakaminen tarvitsee noin 30–40 sykliä. Neliöjuuri, sini ja cosini tarvitsevat noin 60–100 sykliä, jotta ne saadaan laskettua. Normalize-funktion käyttöä kannattaa myös välttää, koska se käyttää neliöjuurta. Normalize sijaan on suositeltavaa käyttää magnitude-funktiota.

Esittelen ensin kohtia, joita on hyvä optimoida, tämän jälkeen optimointimenetelmiä. Raycasting käyttäminen mesh-objekteihin on raskasta. Raycastingiä käytetään muun muassa etäisyyksien mittaamisessa. Sitä

voidaan optimoida harventamalla sen kutsumistiheyttä tai käyttämällä Unityn Layer-kerrosominaisuutta. Asettamalla kerroksia voidaan valita mistä kerroksesta raycast etsii collider-objekteja. Tällöin raycast törmää vähemmän ei-tarkoitettuihin kohteisiin. Draw callsien määrä kasvaa kun käytössä on useita objekteja, tämä on raskasta CPU:lle. Mikäli objekteissa on paljon kolmioita, tämä on raskasta GPU:lle. Näitä voidaan optimoida yhdistämällä objekteja yhtenäiseen meshiin. Objektien tulee olla lähellä toisiaan, sekä niillä tulee olla sama materiaali ja tekstuuri. Objekteja jotka ovat kaukana toisistaan, ei tule yhdistää, koska yhdistetyt objektit renderöidään vaikka vain yksi niistä olisi näkyvissä. Objektien lisäksi tekstuurit voidaan yhdistää yhteen tekstuuriarkkiin.

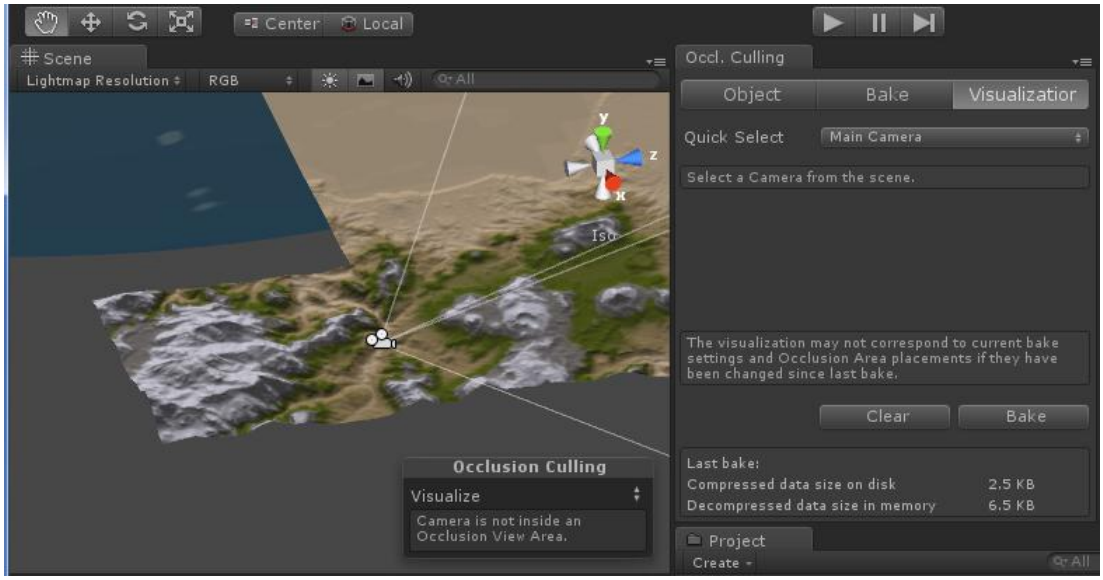
Ante (2007) kertoo, että nykyiset tietokoneet pystyvät käsittelemään noin 500 draw callsia yhdessä piirtokehyksessä. Meshien yhdistämisen kannalta on suositeltavaa yhdistää meshit siten, että yhteen meshiin tulee noin 1500–4000 kolmiota. Näin saadaan vähennettyä kutsuja grafiikkapiirille ja saadaan kutsuista sopivan kokoisia. On suositeltavaa käyttää vain yhtä Skinned Mesh Rendereriä hahmoa kohden. Mikäli rakentaa hahmon useista osista, joille tarvitsee usean Skinned Mesh Rendererin, renderöinnin määrä moninkertaistuu. Materiaalien ja luiden määrää tulee myös rajoittaa mahdollisimman vähäisiksi. Animointia varten usein alle 30 luuta riittää ja yhden hahmon materiaaleihin useasti riittää 2–3 kappaletta. Animointien määrää voi vähentää käyttämällä samaa animointia useiden mallien kesken. Unity ei tarvitse Inverse Kinematics solmuja, joten on suositeltavaa poistaa ne hahmosta. Mikäli hahmo sisältää IK-solmuja tuotaessa Unityyn, IK-solmut yhdistetään Forward Kinematicsiin. Hahmojen suositeltava kolmioiden määrä riippuu julkaisualustasta. Ante (2007) suosittelee mobiililaitteilla pysymään 300–1500 kolmion välillä ja PC-alustalla käyttämään 500–6000 kolmiota hahmoa kohden. PS3 ja Xbox360 -laitteilla voidaan käyttää 5000–7000 kolmiota yhtä hahmoa kohden. Esimerkkinä suosittu PC-peli Half-Life 2, joka julkaistiin vuonna 2004. Peli sisälsi uusia hahmon luontimenetelmiä ja peli palkittiin sen näyttävistä hahmoista. Half-Life 2 hahmoissa oli noin 2500–5000 kolmiota (Unity 2010b).

Versiosta 3.0 alkaen Beäst Lightmapping ja Umbra Occlusion Culling ovat olleet osa Unityn parhaimpia optimointiohjelmistoja. Beäst on ohjelma joka laskelmoi staattisten objektien valoisuuden ja varjoisuuden erilliseen uv-sarjaan. Tämän tekniikan avulla voidaan vähentää valo-objektien määrää ja vähentää valojen simulointia pelin aikana. Beäst Lightmappingin lisenssi maksaa normaalisti 90.000 dollaria julkaistua peliä kohden, mutta tulee ilmaiseksi Unity Pro -lisenssin käyttäjille (Unity 2011a). Beästin avulla saadaan myös realistisempi valaistus peliin käyttämällä sen lukuisia asetuksia, kuten varjojensyvyyttä. Beäst on integroitu suoraan Unity Editoriin ja se pystyy laskelmoimaan Unityn taustalla, eikä häiritse työnkulkua.



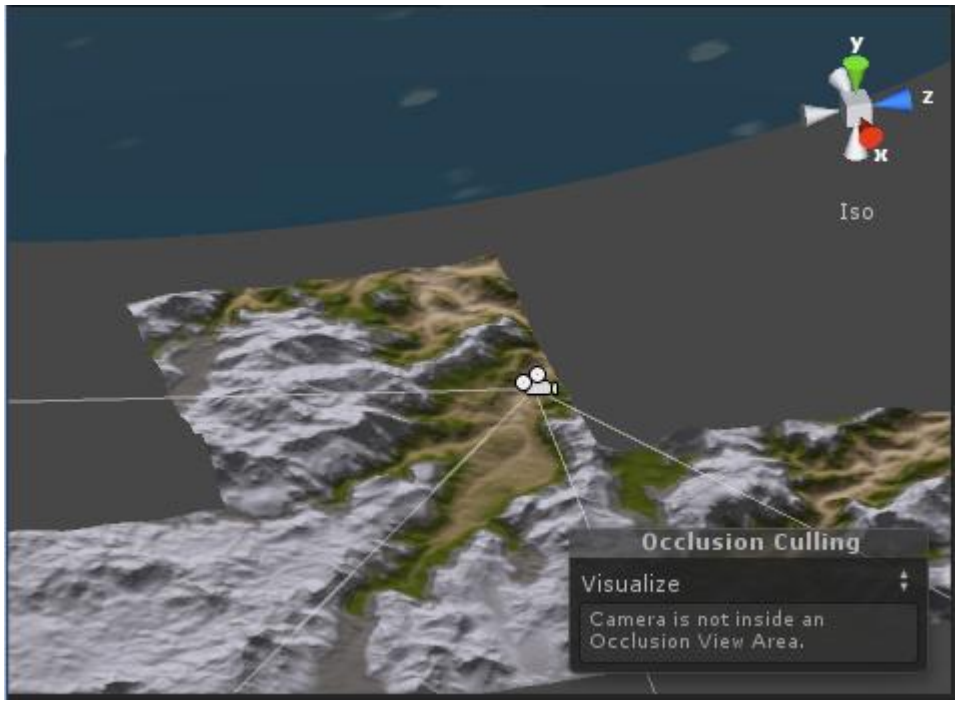

Kuva5. Beäst Lightmapping demonstraatio

Kuvassa on 20 km kertaa 20 km kokoinen kartta, joka on valokartoitettu käyttäen hyvän laadun asetuksia. Lightmap vie ainoastaan 2.7 megabittiä VRAM:ia koko kartalla. Kaikki valot on saatu karttaan käyttämällä niin vähän näytönohjaimen muistia. Vaikka Beäst laskelmoi staattiset valot peliin, se ei tarkoita ettei pelissä voitaisi käyttää myös realistisia varjoja samaan aikaan. Unityn Dual Lightmapping -tekniikan avulla staattiset varjot sulautuvat realististen varjojen kanssa. Tämä lisää paljon suorituskykyä käytettäessä realistisia varjoja, koska varjoja tarvitsee laskelmoida ainoastaan pelaajan lähettyvillä. Umbra Occlusion Culling on väliohjelmisto, joka laskelmoi mitkä objektit renderöidään.




Kuva6. Umbra Occlusion Culling demonstraatio 1

Occlusion Cullingin tehtävänä on renderöidä ainoastaan ne alueet ja objektit, jotka ovat näkyvissä kameralle. Kuvassa näkyy renderöitynä kameran näkemä alue ja mitään muuta ei ole renderöity. Käännettäessä kameraa, renderöidään alue, joka on sen edessä. Vertaa edellä olevaan kuvioon.

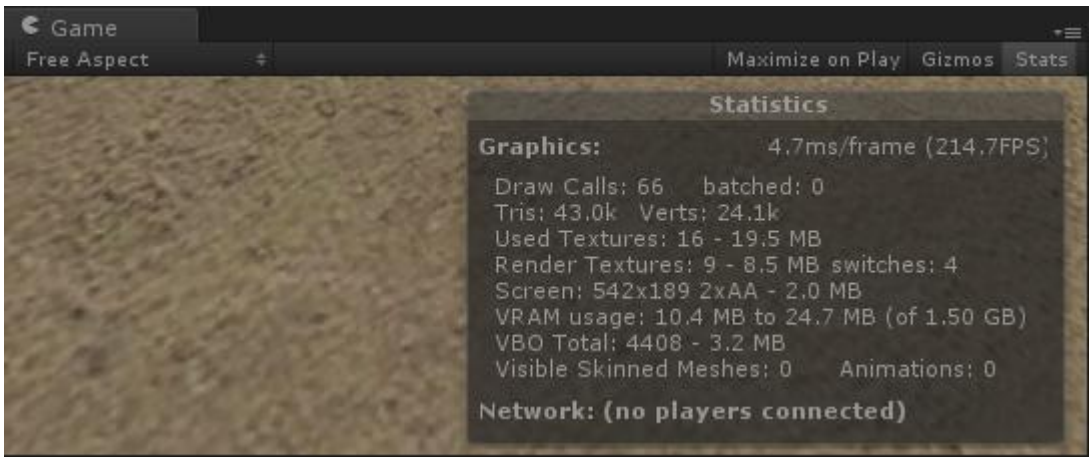



Kuva7. Umbra Occlusion Culling demonstraatio 2

Aikaisempi tekniikka renderöisi myös ne objektit, jotka ovat toisten objektien takana piilossa. Umbra Occlusion Culling sisältyy Unity Pro -lisenssiin ja on

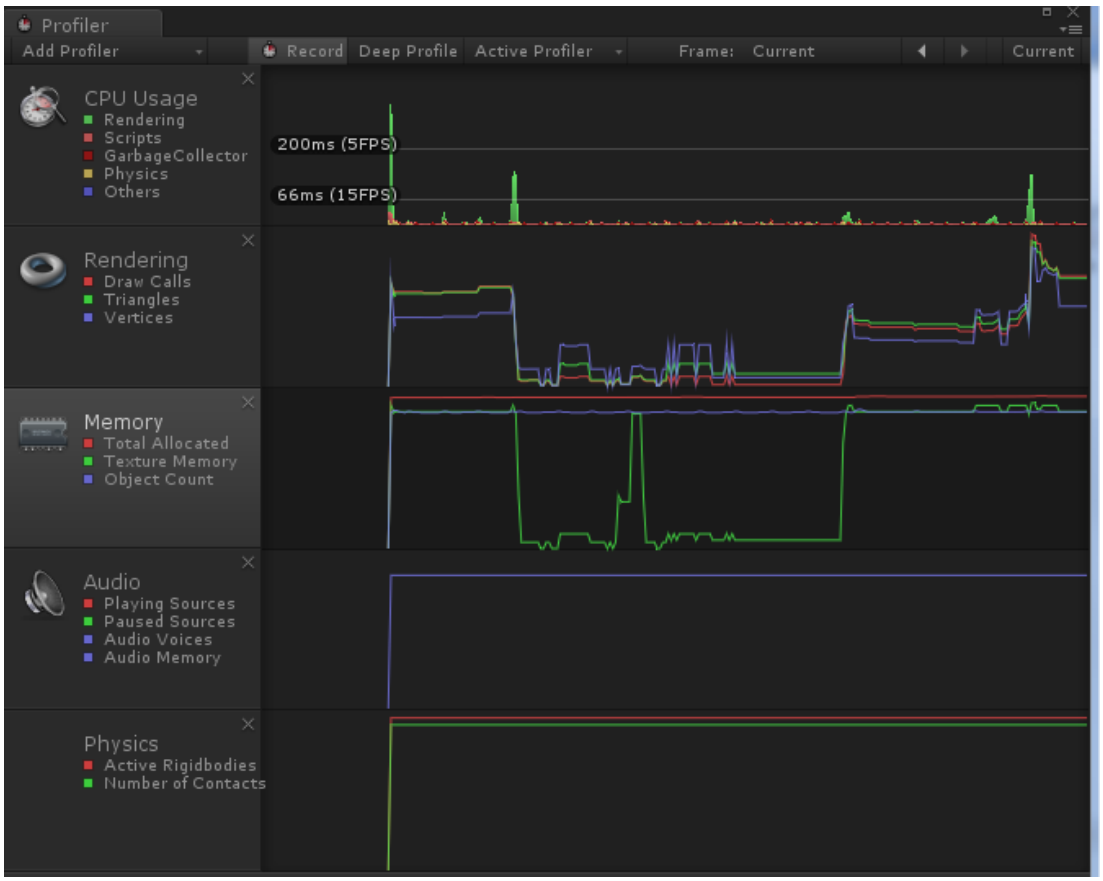
arvoltaan 30.000 dollaria. Umbran asettaminen peliin vaatii hieman säätämistä käyttäjältä, mutta sen avulla voidaan saada moninkertainen kuvataajuus. Umbra toimii asettamalla alueita kameraa varten, jolloin kamera renderöi vain ne objektit, jotka ovat alueen sisällä.

Unity Technologies päivitti renderöintimoottorinsa Unity 3.0 -versiossa ja saavutti jopa 50 prosenttia nopeamman renderöinnin. Toinen automaattinen optimointiominaisuus suoraan Unityssä on Batching-tekniikka, joka vähentää Draw Callsien määrää yhdistämällä geometriaa paketeiksi automaattisesti (Unity 2011b). Nämä paketit lähetetään grafiikkapiirille Draw Callsissa ja yhdistelemällä grafiikkaa niitä tarvitsee lähettää vähemmän. UT kirjoitti myös oman optimoidun version OpenGL Shading Languagesta. Tämän optimoidun version avulla saadaan kaksin- tai jopa kolminkertainen ruuduntäyttönopeus renderöidessä suotimia (Unity 2010c). Unityssä on kaksi suorituskykyyn liittyvää tilastotyökalua. Ensimmäinen näistä on renderöintitilastoikkuna (2010), joka sijaitsee peli-ikkunassa.




Kuva8. Renderöintitilastoikkuna

Tässä näkyy pelinaikana renderöintiin liittyviä tilastoja, kuten muun muassa Draw Callsien, kolmioiden, VRAM:in ja animointien lukumäärä. Siinä näkyy myös kuinka kauan aikaa kuluu renderöidä yksi ruutu ja mikä on kuvataajuus sekunneissa. Toinen tilasto-ohjelma on Profiler (2011), joka raportoi kuinka paljon suorituskykyä kuluu mihinkin osa-alueeseen.




Kuva9. Profiler-ikkuna

Osa-alueita ovat CPU:n käyttö, renderöinti, komentosarjat, roskienkeruu, fysiikkojen mallintaminen, sekä muut osio. Renderöinti osiosta näkyy erikseen Draw Callsien, kolmioiden, sekä lakipisteiden määrä. Kolmannessa osiossa näkyy muistinkäytön määrä ja neljännessä osiossa näkyy äänien käyttämän muistin määrä. Profiler piirtää reaaliaikaisen käyrän aikajanalla, josta voidaan seurata missä kohtaa peliä käytettiin eniten resursseja.

Unitystä on tarjolla ilmainen ja maksullinen versio. Ilmaisessa versiossa on rajoitetusti ominaisuuksia ja sillä voidaan julkaista ainoastaan PC:lle, Mac:lle ja selaimelle. Unityn maksullisella Pro-versiolla saa käyttöönsä kaikki ominaisuudet ja lisälisenssillä saa käyttöönsä eri julkaisualustoja. Unityn Pro-versio maksaa tällä hetkellä 1100 euroa, mobiilialustoista on tarjolla tavallinen ja Advanced-versio. Mobiilialustat ovat iOS ja Android. IOS -versiolla pääsee julkaisemaan pelejä iPod Touch:lle, iPhone:lle ja iPad:lle. Android alusta tukee myös Sonyn tulevaa Xperia Play:ta. Tavallisen version saa haltuunsa 300 eurolla ja Advanced-version 1100 eurolla. (Unity 2011c.) Konsoleita varten studion tulee rekisteröityä kehittäjäksi alustajalle ja maksaa

alustan lisenssöintimaksut ja tämän jälkeen ostaa julkaisualusta Unityyn.

Ilmaisesta-versiosta puuttuvat audio-suodattimet, suoratoistaminen verkosta, sekä videojen sisällyttäminen peliin (Unity 2011c). Julkaistaessa pelin ilmaisessa versiossa näytetään Unityn Splash Screen aina, mutta Pro-version käyttäjät voivat asettaa oman studion logonsa käynnistyksen yhteyteen (Unity 2011c). Grafiikkojen osalta ilmainen-versio tukee valokartoitusta, mutta ei globaalihohto-efektiä, jolla simuloidaan luonnon valoa ulkona (Unity 2011c). Ilmainen versio ei myöskään tue Umbraa lainkaan, eikä deferred renderöinti-ominaisuutta (Unity 2011c). Eikä reaaliaikaisiavarjoja ole lainkaan ilmaisessa-versiossa (Unity 2011c).

Unityn mukana tulee vakio asset-paketti, jossa sijaitsee kameraefektejä, komentosarjoja ja 3D-malleja. Pro asset-paketti sisältää enemmän kameraefektejä, komentosarjoja, sekä paremman version ilmaisesta vedenrenderöinti-tasosta. Optimointitilastosovelluksia ei ole kumpaakaan ilmaisessa versiossa (Unity 2011c). Kuitenkin ilmaisella versiolla pystyy tekemään ihan laadukkaita pelejä ja niiden julkaiseminen on myös ilmaista. Pro-versiosta on myös tarjolla 30 päivän kokeiluversio. Unity Pro:n käyttäjät saavat myös ilmaisia päivityksiä tasaisin väliajoin. Unityn 3.0 versiosta tuli kuitenkin maksullinen päivitys sen laajuuden vuoksi, mutta aikaisemmat Unity Pron käyttäjät saivat uuden version alennettuun hintaan.

Unityyn on tarjolla myös UT:in oma versionhallintasovellus, joka integroituu Unityn käyttöliittymään. Versiohallintasovellusken nimi on Unity Asset Server, joka on tarjolla Unity Pro-version käyttäjille (Unity 2011d). Version hallinta on myös maksullinen ja sen nykyinen hinta on 370 euroa, mutta se ei ole ainut versionhallintasovellus, jota voidaan käyttään. Unityn kanssa voi käyttää haluamaansa versionhallintasovellusta, kunhan valitaan oikeat tiedostot versionhallintaa varten. Unity Technologies on pyrkinyt hankkimaan pelialan standardi väliohjelmistoja, kuten Umbra ja Beäst. Uusimpien pelialanuutisten mukaan Unityyn integroidaan EKI One A.I-väliohjelmisto (Gamasutra 2011). EKI One A.I sisältää kaikki tarvittavat työkalut tekoälyn suunnitteluun ja toteutukseen. Toinen väliohjelmisto Allegorithmic liittyy tekstuurien pakkaus algoritmeihin ja kentän mallinnus työkaluihin (Gamasutra 2010).

Allegorithmiciä käytetään MMO-pelien tiedostokoon pienentämisessä (Gamasutra 2010). Unityllä on myös oma julkaisuportaali Union, jossa pelinkehittäjä saa 80 prosenttia voitoista (Unity 2011e). Muissa julkaisuportaaleissa kuten Steam-pavelussa, pelinkehittäjä saa 60 prosenttia digitaalisistä myynneistä, joka on silti paljon verrattuna perinteiseen julkaisijan kautta tapahtuvaan myyntiin (Steamreview 2005). Unityllä on myös verkkokauppa rakennettuna Unityn sisään, jossa pelinkehittäjät voivat myydä komentosarjojaan tai 3D-mallejansa. Verkkokaupan nimi on Unity Asset Store, jossa myyjä saa 70 prosenttia voitoista (Unity 2011f). Tämä on yleinen osuus voitoista, kun myydään 3D-malleja verkossa.

Taulukko 1. Pelimoottoreiden vertailu.

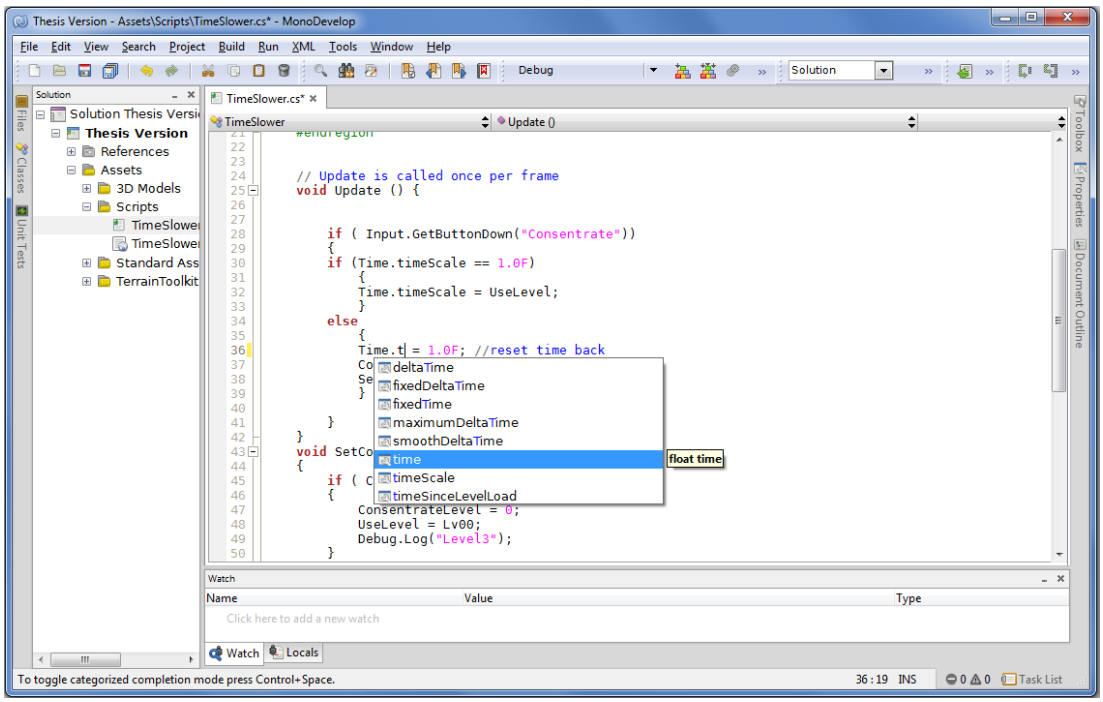
| **Nimi** | **Unity 3D** | **Torque 3D** | **Unreal Development Kit** | **SIO2 Engine** | **Essenthel** |
|---|---|---|---|---|---|
| **Renderöinti** | DirectX 9, OpenGL | DirectX 9, OpengGL | DirectX 11 | OpenGL | DirectX 11 |
| **Ääni-moottori** | FMOD | FMOD | FMOD | ??? | OpenAL |
| **Fysiikka-moottori** | PhysX | Vaatimaton, integroituu muiden fysiikka-moottorien kanssa | PhysX | Bullet | Bullet |
| **Skriptikieli** | C#, JavaScript, Boo | Torque-Script | UnrealScript | Lua | Lua |
| **Lähdekoodi saatavilla** | Erillisellä lisenssillä | Kuuluu lisenssiin | Ei ole | Kuuluu lisenssiin | Ei ole |
| **Tekoäly** | Yhteisön ilmainen liitännäinen | Yhteisön maksullinen liitännäinen | Unreal AI | Reitinhaku | Ei ole |
| **Hinta** | Ilmainen, Pro 1500$ | Normaalisti 1000$ nyt rajoitetun ajan 99$ | 99$ + 25% voitoista, mikäli tulot yli 50 000$ vuodessa | 364,99$ | Yksityinen 150$, yritys 750$ |
| **Huomioitavaa** | Umbra Occlusion Culling, Beäst Lightmapping | pureLight | Tuohoutuvat ympäristöt, Unreal Lightmass, Kasvojen animointi | PVR-pakkaus | Integroitu MMO-teknologia |

Valitsin Unity 3D:n pelimoottorin, koska se sopi mielestäni parhaiten tälle projektille. Harkitsisin Essenthel-pelimoottoria, mikäli olisin tekemässä MMO-peliä. Muilta ominaisuuksiltaan moottori oli keskinkertainen (Essenthel 2011).

SIO2-pelimoottorin kotisivut saivat pelimoottorin näyttämään ammattimaiselta ja hienolta. Lähemmän tarkastelun jälkeen huomasin, että sen ominaisuudet eivät ole sen erikoisemmat kuin Essenthelissäkään (SIO2-engine 2011). SIO2:sta oli vaikea löytää mainittavaa huomioitavaa-kohtaan, mutta se on varmasti riittävä pelimoottori ensimmäisiä projekteja varten. Lähdekoodin avulla siihen voisi liittää uusia ominaisuuksia. Unreal Development Kit perustuu Unreal 3-teknologiaan ja sen näyttävyys pelien laadussa on alan parhaita. UDK:sta löytyy myös A.I-ominaisuudet ja siitä ei näytä puuttuvat mitään (UDK 2011). Kuitenkin Unreal-pelimoottorit ovat tunnettuja pelien raskaudesta ja pelejä on vaikea saada toimimaan samantasoisena vanhemmilla laitteilla. Torque 3D oli Unityn alkuvuosina varteen otettava vaihtoehto ja se oli myös hintatasoltaan lähes sama. Kuitenkin Torque3D:stä puuttuu ominaisuuksia, jotka tulee itse implementoida tai integroida (Torque3D 2011). Nyt Torque 3D on hinnaltansa pelimoottoreista halvin, mutta ominaisuuksiltansa silti yksi heikoimmista. Myöskään ilman lähdekoodia pelimoottorista ei saanut paljoa irti ja ainakin vuosi sitten, Torquen sanottiin olevan erittäin buginen. Unity 3D on mielestäni näistä vahvin ja samalla kevein pelimoottori, jolla on tarjolla useita julkaisualustoja ja vahva yhteisön tuki.

### 2.2.2 MonoDevelop

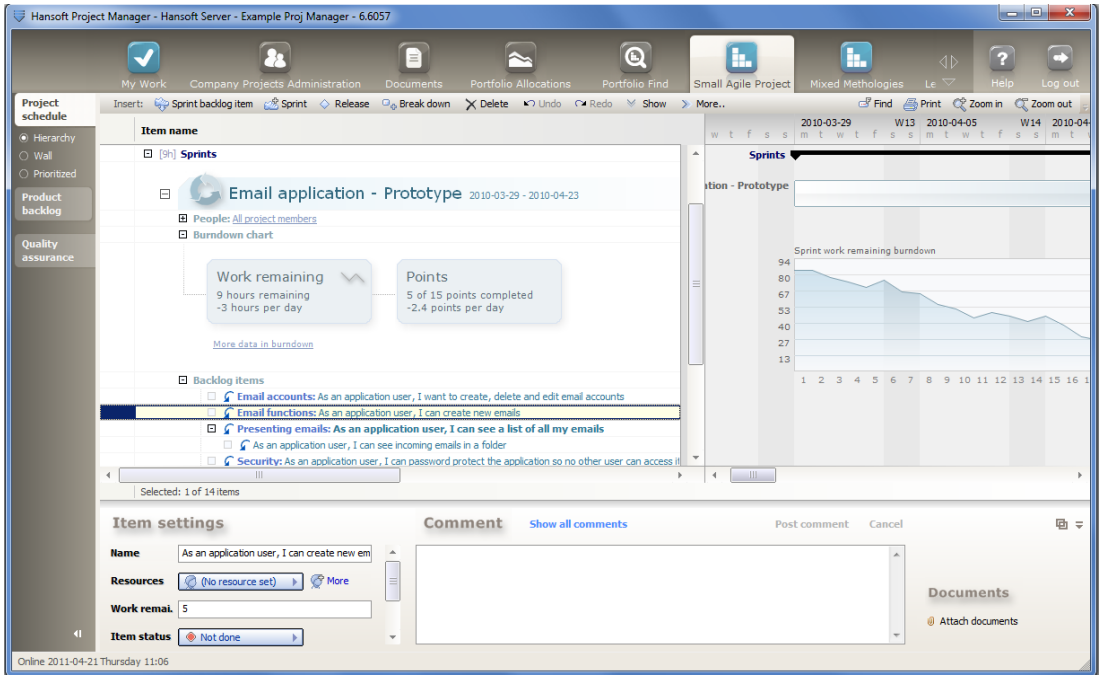
MonoDevelop on integroitukehitysympäristö .NET-kielille. Se pohjautuu avoimeen lähdekoodiin ja on alustariippumaton. Unityn kannalta MonoDevelop on parempi kuin Microsoftin Visual Studio, koska se tukee osittain UnityScriptiä, kun Visual Studio ei tue sitä lainkaan. Osittainen automaattinentäydennys on parempi kuin ei mitään. Automaattisella täydennyksellä voidaan nopeuttaa ohjelmointia ja välttää kirjoitusvirheitä. Jotta MonoDevelop saadaan käyttöön Unityssä, tarvitsee vain painaa Assets-valikosta Sync MonoDevelop Project ja vaihtaa asetuksista käytettävä External Script Editor MonoDevelopiksi. MonoDevelop on myös ainoa IDE, jossa voidaan suorittaa debuggaus, Unityn integroidun debuggerin lisäksi. MonoDevelopin käyttöliittymä muistuttaa Visual Studiota.




Kuva10. MonoDevelopin käyttöliittymä

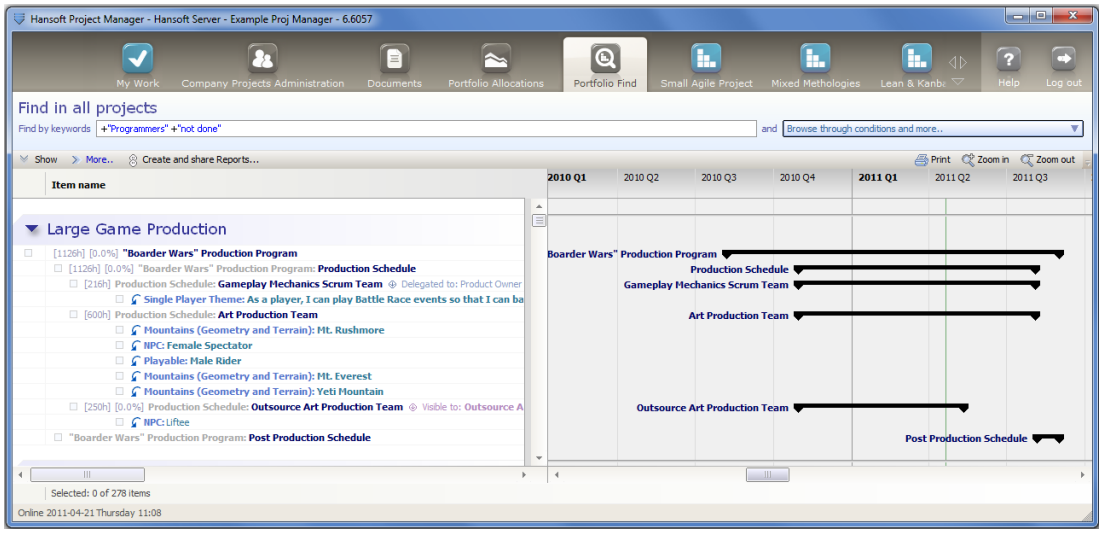
### 2.2.3 Hansoft

Hansoft on projektinhallintasovellus, joka sopii ketteriin ohjelmistokehitysmenetelmiin, kuten Scrumiin ja eXtreme Programmingiin. Sovelluksen käyttäjät saavat reaaliaikaista tietoa projektin etenemisestä ja jäljellä olevista työtehtävistä. Hansoftin avulla projekti pysyy aikataulussa ja tärkeät tiedot, kuten työmäärä- ja riskiarviot pysyvät ajantasalla.




Kuva11. Hansoftin Käyttöliittymä

Ohjelma soveltuu kaiken kokoisiin projekteihin ja käyttäjät voivat kirjautua ohjelmaan verkon kautta. Projektipäällikkö voi asettaa, että projektin jäsenet näkevät vain heille kuuluvat projektin osa-alueet. Projektipäällikkö pystyy myös asettamaan tarvittavat työmäärät eri työtehtäville ja työntekijä pystyy kirjaamaan tehdyt työtunnit.




Kuva12. Ohjelmoijan jäljellä olevat työtehtävät Hansoftissa

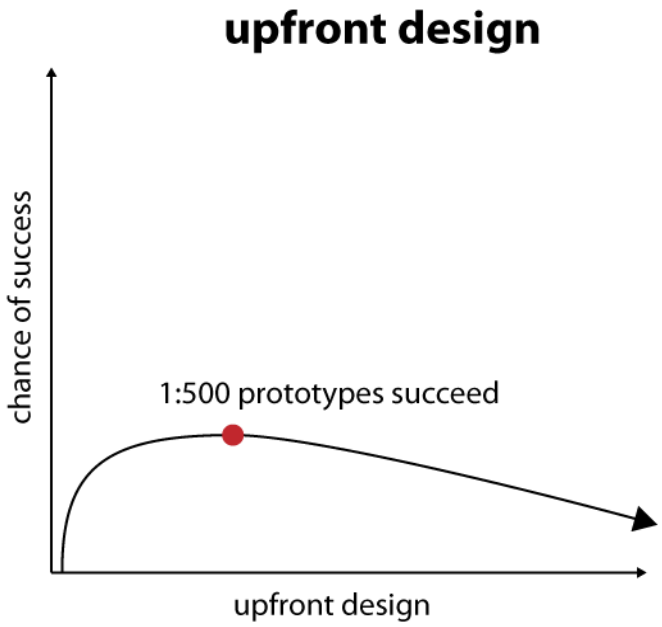
Tällöin projektinseurannassa näkyy paljonko ja mihin työtunteja vielä tarvitaan. Aikataulut pystytään myös esittämään graafisesti. Hansoftin käyttöönotto on nopeaa ja mielestäni se mukautuu käytettävään ohjelmistokehitysmenetelmään todella hyvin.

### 2.2.4 Ohjelmistotuotannon menetelmät

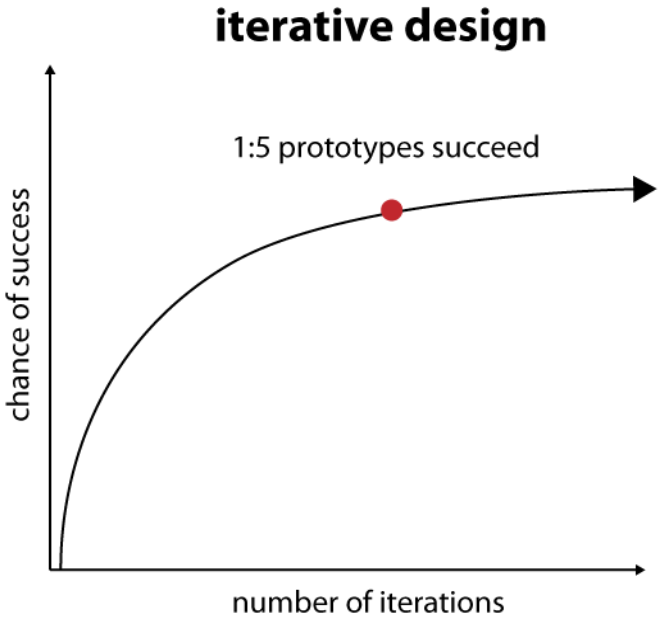
Rational Unified Process -menetelmää käytettiin projektin alkuvaiheessa. RUP-menetelmässä sovellusta toteutetaan iteroiden ja dokumentoiden. RUP:ssa dokumentointi ja laadunhallinta ovat suuremmassa osuudessa kuin eXtreme Programming-menetelmässä, mutta muuten ne ovat melko samankaltaisia, eivätkä ole toisiaan pois sulkevia menetelmiä. Kuitenkin projektissani dokumentoinnin päivittäminen vei paljon aikaa, jota tarvitsin muualla. Aloin tutustumaan XP-menetelmään, koska kuulin että dokumentointi olisi siinä vähäisempää. Dokumentointi on mielestäni tärkeää, mutta projektini aikataululla dokumentointi tuntui tarpeelliselta vasta suurien muutosten jälkeen.

Extreme Programming on ketterä ohjelmistokehitysmenetelmä, joka perustuu

testaukseen ja iterointiin. XP-menetelmä keskittyy muutoksien mukautumiseen, eikä niiden ennalta-arvaamiseen. Pelikehityksessä halutun ominaisuuden, kuten hauskuuden löytäminen harvoin onnistuu ensimmäisellä yrityksellä. Iteratiivinen kehitys nostaa projektin mahdollisuuksia onnistua, koska kaiken ennaltasuunnittelu voi olla erittäin vaikeaa.




Kuva13. Ennalta suunnittelevan projektin onnistumis mahdollisuus.



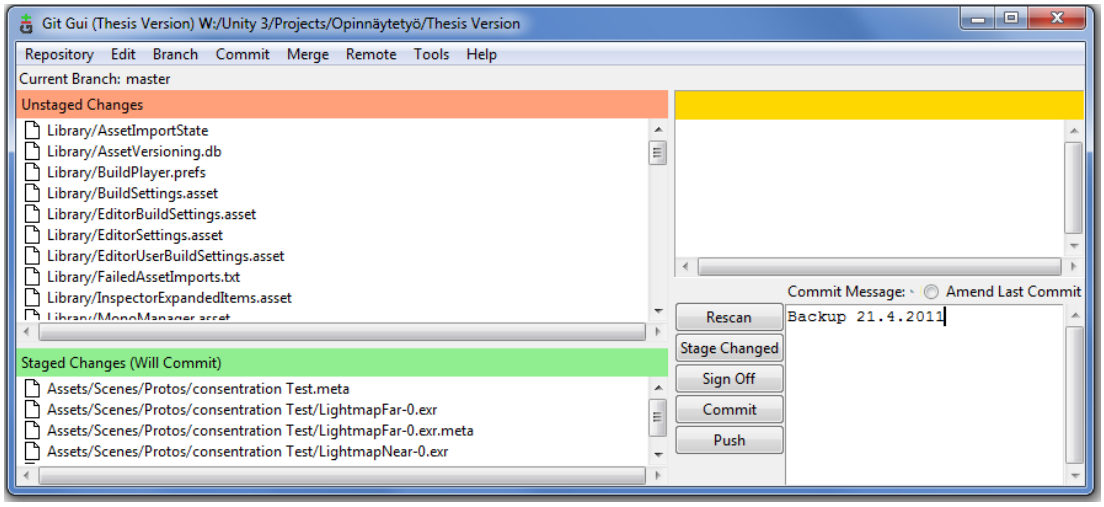

Kuva14. Iteroivan projektin onnistumis mahdollisuus.

Iteroimalla ideat voidaan helposti hylätä ja siirtyä uusiin ideoihin, kunhan nämä ongelmat havaitaan projektin alkuvaiheessa. Ohjelmoinnissa pyritään kokeilemaan ideoita käytännössä mahdollisimman pian ja samalla pitämään

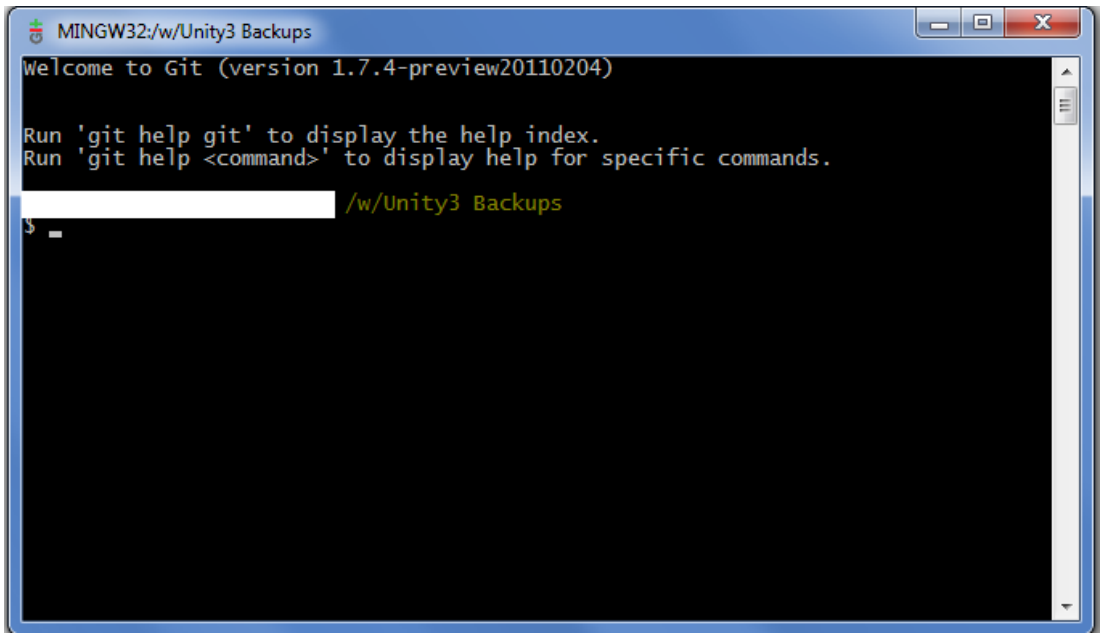
koodi yksinkertaisena ja helposti muokattavissa. XP-menetelmän avulla peli pysyy kokoajan ajantasalla ja siitä on näytettävissä uusin prototyyppi. Pelikehityksessä pyritään pitämään pelin ulkonäkö karkeana ja nopeana toteuttaa. Projektin loppuvaiheessa tehdään suurin osa visuaalisesta työstä ja hiotaan peliä. Testisuuntautuneessa kehityksessä testataan jokainen toiminto erikseen ja siirrytään seuraavan toiminnon kehitykseen, kun ensimmäinen toiminto läpäisee testit. Tämän avulla voidaan olla varmoja että koodi tekee mitä sen pitää tehdä ja että se toimii koko projektin kanssa. XP-menetelmällä on tapana toimia hyvin pienissä ja keskikokoisissa kehitystöissä, jotka ovat ympäristössä, jossa määritykset muuttuvat useasti ja lähes välitön kommunikointi on mahdollista. ( Myers 2004, 178).

### 2.2.5 Git-versionhallinta

Git on ilmainen, alustariippumaton ja avoimeen lähdekoodiin perustuva versionhallintajärjestelmä. Versionhallinta säilöö projektin versiot ja sen avulla voidaan palata haluttuun versioon. Se toimii myös projektin varmuuskopiona, mikäli alkuperäinen vahingoittuu tai halutaan siirtyä uudelle työasemalle. Vaikka ei käytettäisi Unityn Asset Serveriä, versionhallinta on silti vain Unity Pro -ominaisuus. Jotta Git saadaan käyttöön Unityn kanssa, tulee ruksata Unityn projekti-asetuksista Enable External Version Control. Tällöin Unity tallentaa .meta-päätteiseen tekstitiedostoon tarvittavat tiedot, jokaisesta assetista, joka pitää kopioida alkuperäisen tiedoston kanssa versionhallintaan.

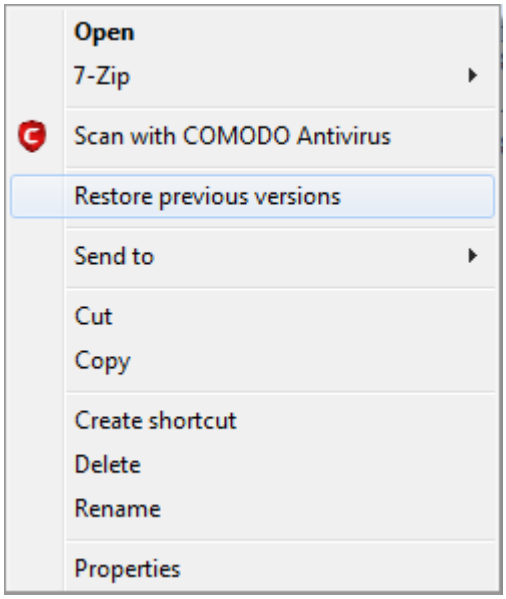



Kuva15. Git Gui -version käyttöliittymä




Kuva16. Git konsoli -version käyttöliittymä

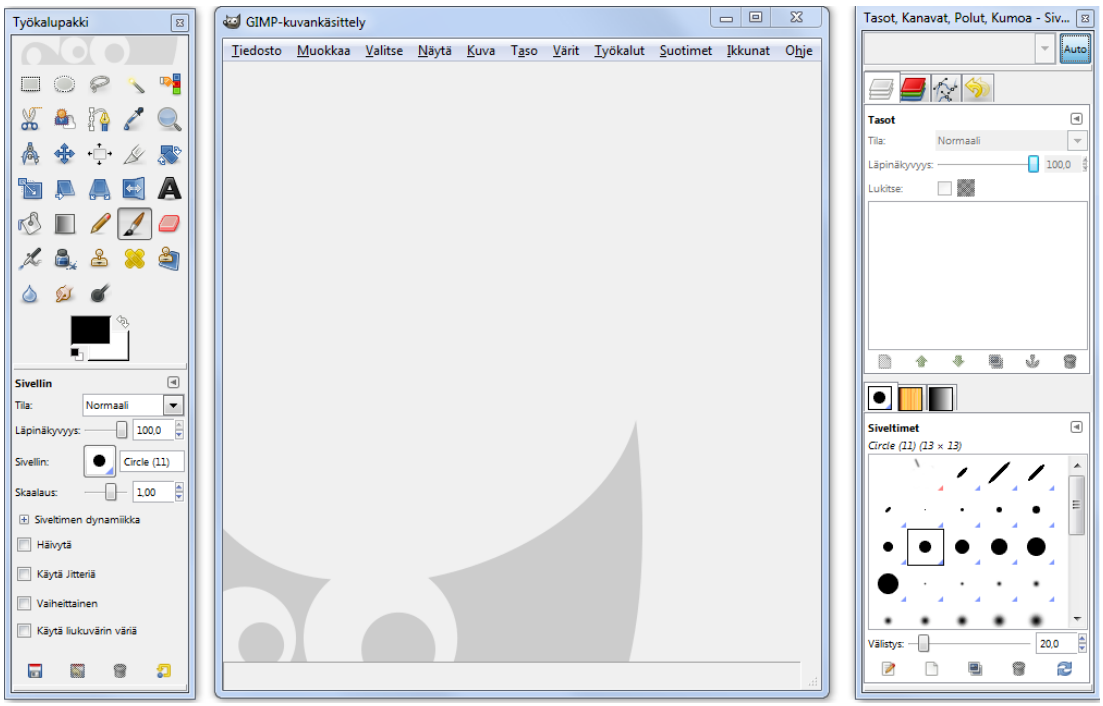
Git:llä saa nopeasti palautettua tiedoston aikaisempaan versioon seuraavasti:




Kuva17. Tiedoston nopea palautus aikaisempaan versioon

### 2.2.6 GIMP

GIMP on GNU kuvanmanipulointiohjelma, joka on työkalu kuvien muokkaamiseen ja saa ulkoasunsa ja käyttökokemuksen suositusta Macintoshista ja Microsoft Windows ohjelmasta nimeltään Photoshop, jonka on kehittänyt Adobe (Hammel 1999,1). GIMP on ilmainen alustariippumaton, kevyt, sekä avoimeen lähdekoodiin perustuva kuvankäsittelyohjelma.

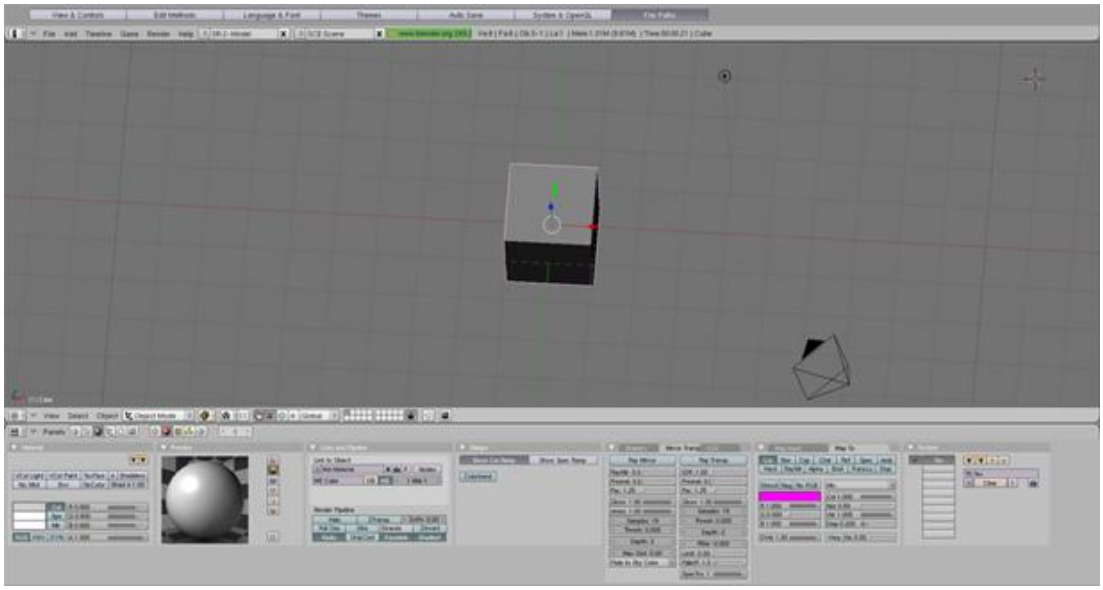



Kuva18. Gimpin käyttöliittymä

Ohjelman ominaisuudet ovat samankaltaisia kuin Adoben Photoshopissa, joka on yksi parhaista tarjolla olevista maksullisista kuvankäsittelyohjelmista. GIMP:iä voidaan käyttää yksinkertaisena piirto-ohjelmana, sekä ammattimaisena valokuvien muokkaus sovelluksena. Yksi GIMP:in vahvuuksista on että sille on helppo saada ja tehdä liitännäisiä. GIMP on ilmainen ja sitä suojaa GNU GPL -lisenssi, jotta ohjelma ja sen lähdekoodi pysyisivät ilmaisina.

### 2.2.7 Blender

Myös Blender on ilmainen ja alustariippumaton -sovellus. Blender on yksi suosituimmista avoimen lähdekoodin pohjautuvista 3D-mallinnusohjelmista. Blenderin tiedostokoko on pieni kuten GIMP:in, eikä Blenderikään tarvitse suuria tehoja toimiakseen. Blenderin käyttöliittymä on ainutlaatuinen, joka perustuu yksinkertaiseen ajatukseen. Mikään käyttöliittymän osa ei saa olla toisen käyttöliittymän osan edessä.



Kuva19. Blenderin Käyttöliittymä

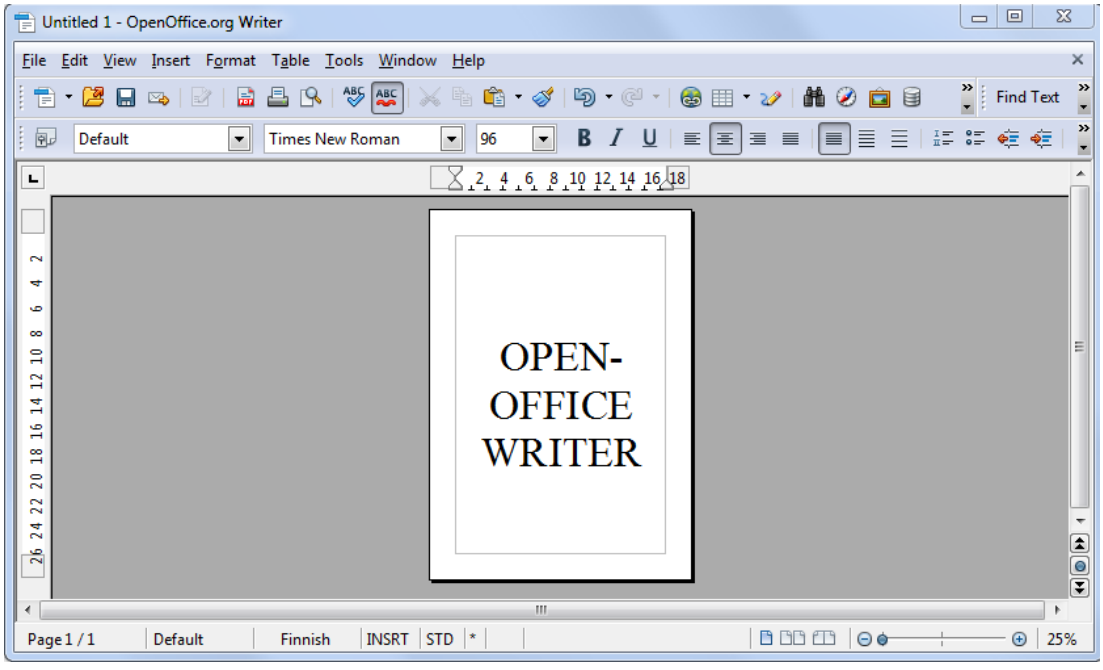
Tämä uusi tapa esittää käyttöliittymää voi vaatia hieman totuttelua joillekin, mutta itselleni sen käyttö selvisi hyvin nopeasti. Käyttöliittymän tavoite on nopeuttaa työskentelyä. Blenderi pystyy 3D-mallinnuksen lisäksi animointiin, simulointiin ja kaiken lisäksi sillä on oma integroitu pelimoottori. Blender on kirjoitettu käyttäen C-ohjelmointikieltä ja sen integroitu pelimoottori on kirjoitettu C++-kielellä. Komentosarjat ja osa käyttöliittymää on kirjoitettu Python-kielellä. Blenderin avulla saadaan korkealaatuisia 3D-malleja, sekä mielestäni se ei ole vaikeampi oppia kuin muutkaan 3D-mallinnusohjelmistot.

### 2.2.8 OpenOffice.org

OpenOffice.org on avoimen lähdekoodin yhteisöprojekti, jota on kehitetty kaksikymmentä vuotta. Projektin tavoitteena on kehittää alustariippumaton toimisto-ohjelmisto. Projektin omisti alun perin Sun Micrososystems, jolta omistajuus siirtyi Oraclelle, tämän ostaessa Sunin. Jonkin aikaa Oracle kehitti OpenOfficesta maksullista versiota, mutta palautti sen myöhemmin takaisin yhteisölle (Oracle 2011). OpenOffice.orgin käyttäminen tapahtuu valitsemalla osaohjelma käynnistysikkunasta.


Kuva20. OpenOfficen käynnistysikkuna.

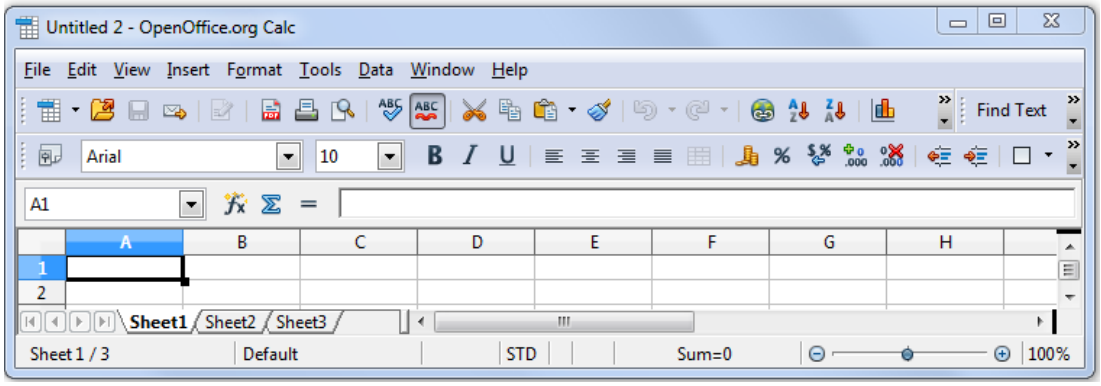
Osaohjelmia ovat Writer-tekstinkäsittelyohjelma, jolla voidaan tehdä tekstidokumentteja. Writer pystyy avaamaan yleisimmät tekstidokumentit, myös Microsoft Office Wordilla tallennetut.




Kuva21. Writerin käyttöliittymä

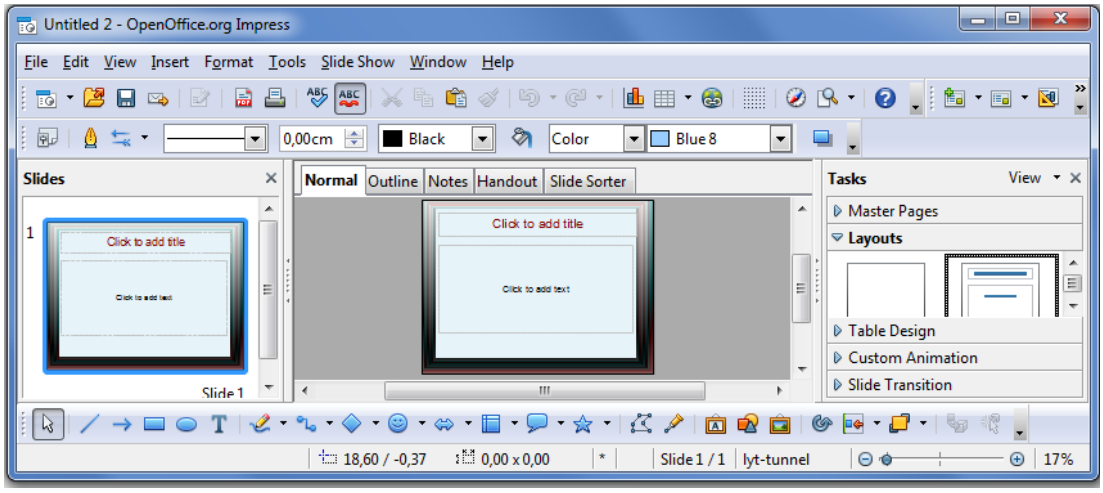
Toinen osa-ohjelma on Calc-taulukkolaskentaohjelma. Calc vastaa Microsoft

Officen Excel-taulukkolaskentaohjelmaa. Myös calc on yhteensopiva useimpien tiedostomuotojen kanssa ja pystyy avaamaan Excelillä tallennetut tiedostot.



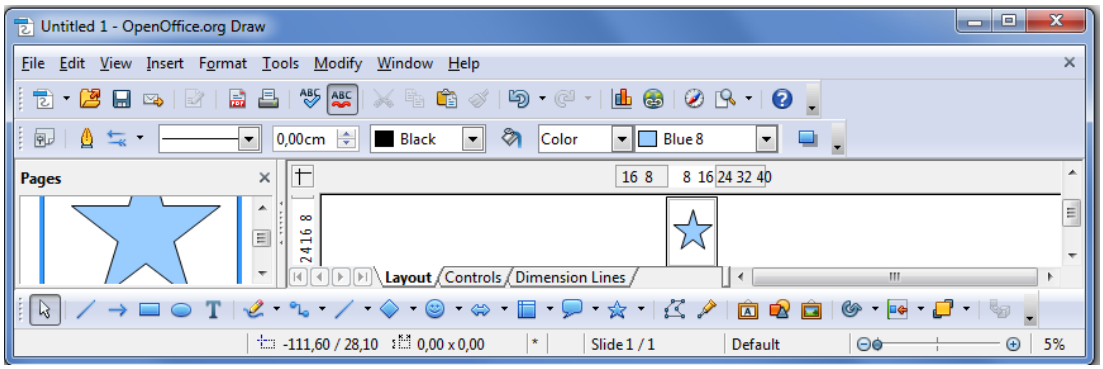

Kuva22. Calcin käyttöliittymä

Kolmas osa-ohjelma Impress-esitysgrafiikkaohjelma, joka vastaa Microsoft Office paketin PowerPoint-ohjelmaa.




Kuva23. Impressin käyttöliittymä

Neljäs osa-ohjelma on Draw-piirto-ohjelma, jolle ei ole Microsoft Office vastinetta. Draw on helppokäyttöinen ja sitä voi käyttää esimerkiksi ajatuskarttojen tekemiseen.




Kuva24. Drawin käyttöliittymä

OpenOfficeen siirtyminen Microsoftin Office-paketista oli helppoa, koska käyttöliittymät ovat samankaltaisia ja suurinosa painikkeista ovat samannimisiä. Tämä opinnäytetyö on kirjoitettu käyttäen OpenOffice.orgin Writeria. Valitsin OpenOfficen, koska halusin tutustua paremmin avoimen lähdekoodin vastine-sovelluksiin ja koska sen käyttäminen on ilmaista. OpenOfficessa on hienoa, että se pystyy avamaan Microsoftin Office-paketin tiedostot ja myös tallentamaan samoilla tiedostoformaateilla. Tämän lisäksi OpenOffice pystyy tallentamaan Open Document -formaateilla (Lammi – Malmirae 2003, 2 - 4.)

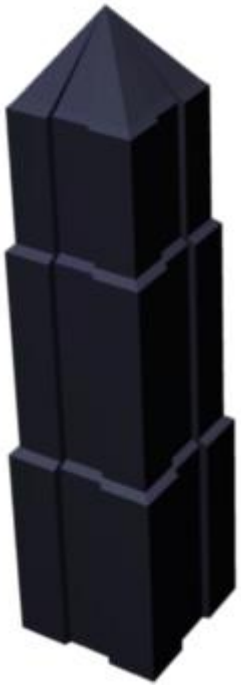
## 2.3 Projektin vaiheet

### 2.3.1 Projektin alkaminen

Aloitin Unity 3D:hen tutustumisen 2009 keväällä ja jatkoin sen harjoittelua koulun alkamiseen asti. Tilasin Unity 3D Pro ja Unity iPhone Advanced-lisenssit, kun Unity 3:sen ennakkotilaus alkoi. Heinäkuussa sain käsiini Unity 3 preview-version, jolla pääsin tutustumaan uuden version uusiin ominaisuuksiin. Unity 3-version beta jatkui koulun alkamiseen asti, jonka jälkeen siitä julkaistiin täysiversio. Etukäteen valmistelun ja harjoittelun tavoitteena oli oppia käyttämään Unity3D:tä etukäteen, jotta voisin käyttää opinnäytetyön ajasta suuremman osan tekemiseen, perusteiden harjoittelun sijaan. Tiesin että opinnäytetyöstä tulisi laaja ja se tulisi sisältämään paljon uutta, jonka tulisin oppimaan vain kokeilemalla.

Koulun jatkettua kesäloman jälkeen, menin keskustelemaan opettaja tuutorini kanssa. Esittelin alustavasti opinnäytetyöni aiheen, jossa tulisin kehittämään videopelin PC:lle käyttäen Unity 3D-pelimoottoria. Tuutorin mielestä aihe kuulosti hyvältä ja sovimme uuden tapaamisen. Seuraava tapaamista varten kirjoitin pelistä synopsin, projektisuunnitelman, sekä erillisen aikataulusuunnitelman. Seuraavalla tapaamisella sain aiheen virallisesti hyväksyttyä ja toimeksiantosopimuksen kirjoitettua. Opinnäytetyön ohjaavaksi opettajaksi pyysin Karlsson Kennethiä, koska hän oli aikaisemmilta kursseilta tuttu opettaja ja tiesin että hänen kanssaan olisi helppo keskustella pelistä. Ennen kuin aloin ideoimaan peliä, tutustuin

Blenderiin ja tein muutamia harjoituksia. Ensimmäinen harjoitukseni oli mallintaa pilvenpiirtäjä.



Kuva25. Pilvenpiirtäjä

Tähän projektiin en aikonut tehdä tämän tyylisiä rakennuksia, mutta harjoitus oli riittävän yksinkertainen, jotta pysyin mukana.

Harjoittelulla halusin varmistaa, että Blender olisi oikea vaihtoehto tälle projektille, ennen toteutuksen alkamista. Tärkein kriteeri oli nopea opittavuus. Käyttöliittymän tunteminen ja siinä liikkuminen tulisi tärkeäksi, jos haluaisin seurata esimerkkejä internetistä. Blenderin peruskäyttö osottautui helpoksi oppia ja jatko-opiskeluun löytyi paljon materiaalia. Muutaman esimerkin jälkeen aloin ideoimaan peliä.

### 2.3.2 Ideointi

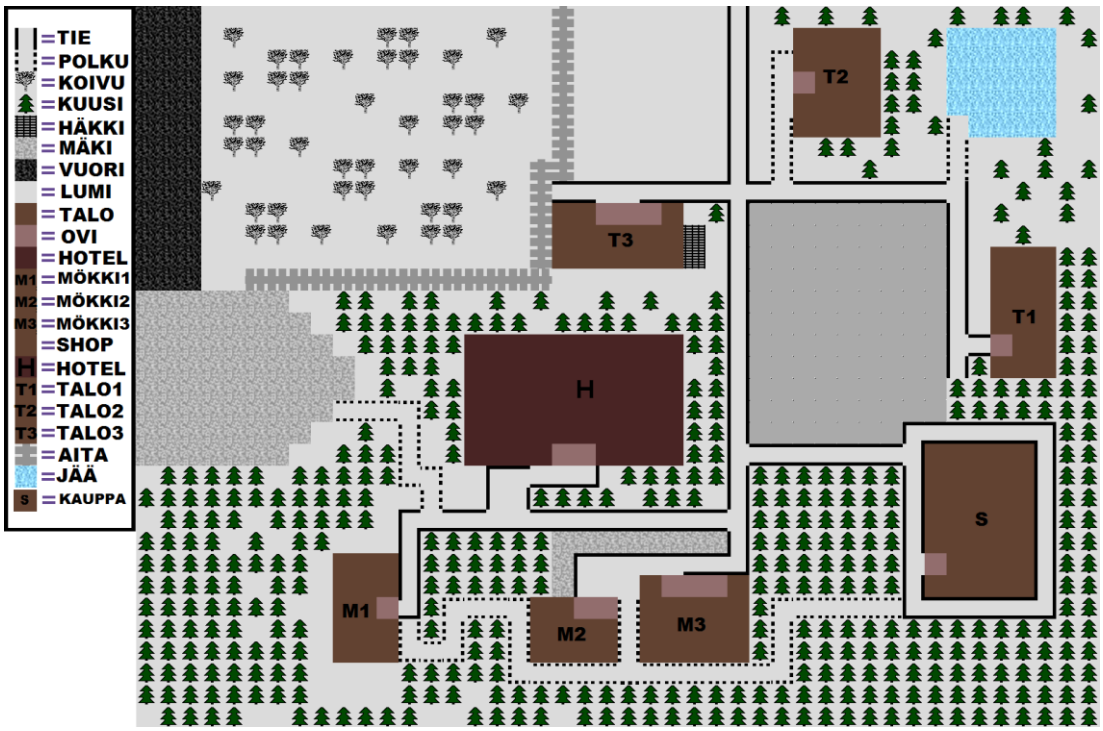
Kun olin varma työkaluista, olin valmis jatkamaan pelin ideointia. Olin ideoinut peliä jo lähes seitsemän kuukautta. Ideointia oli helppo jatkaa, koska minun tarvitsi vain avata ideavihkoni ja alkaa lisätä sekä karsia sisältöä. Pyrin jakamaan ideat eri lohkoihin. Osa ideoista olisi mahdollista toteuttaa aikataulussa ja tulisi olemaan vaikeaa tai mahdotonta toteuttaa yksin. Tämän jälkeen pyrin asettamaan ideat tärkeysjärjestykseen. Ideointivaiheessa pyrin

pitämään ideat yksinkertaisina suunniteluvaiheeseen asti. Ideointivaiheessa pohdin uudelleen aikaisempia ideoitani ja valitsin niistä tähän projektiin sopivimmat, tämän lisäksi mietin uutta sisältöä peliin ja kirjoitin ideoita ideavihkoon. Ideointivaihe ei tulisi loppumaan projektissa missään vaiheessa, vaan se tulisi kulkemaan kaiken muun rinnalla.

### 2.3.3 Suunnittelu

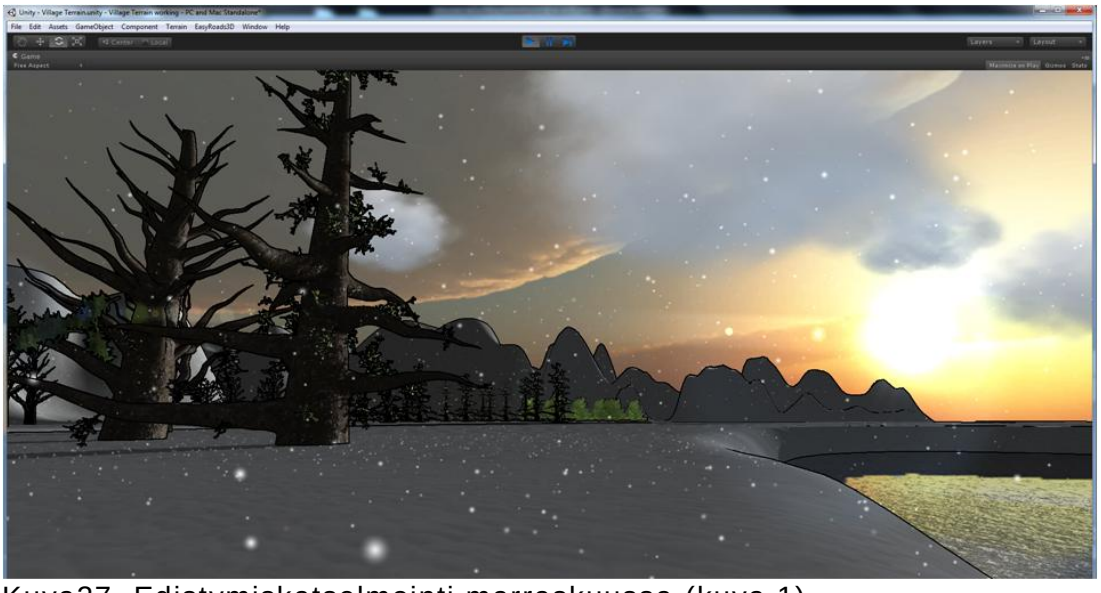
Suunnitteluvaihe alkoi 17. syyskuuta, jolloin luin läpi aikaisemmin kirjoittamani synopsin ja käänsin sen englanniksi. Tämän jälkeen aloin kirjoittamaan laajempaa pelidokumenttia. Suppea pelisuunnitelma ei tulisi olemaan lopullinen pelidokumentti tälle pelille, vaan sitä päivitettäisiin tarvittaessa. Suunnittelulla en halunnut lukita peliä mihinkään pysyvästi, vaan halusin että se toimisi ohjenuorana projektille ja pitäisi projektin kasassa kokonaisuutena. Suppeaa pelisuunnitelmaa kirjoittaessa jouduin vaihe vaiheelta tekemään päätöksiä, miten kukin asia toteutettaisiin pelissä. Suurin osa päätöksistä tehtiin tämän dokumentin kirjoitusvaiheessa ja niitä päätöksiä on rikottu ja muutettu useaan kertaan. Suppeassa pelisuunnitelmassa on kirjoitettu miten pelaaja pelaa peliä, miltä peli näyttää ja tuntuu, mitä siellä tapahtuu, sekä sisältää kenttäsuunnittelun. Suppean pelisuunnitelman ensimmäisessä versiossa tein kenttäsuunnittelun käyttämällä TileStudio-sovellusta.

TileStudio on tiilipohjainen grafiikkaohjelma, jolla on helpposuunnitella 2D-grafiikoita. TileStudion graafinen laatu oli parempi kuin mitä olin saanut sutattua vihkooni ja sitä oli paljon helpompi lukea jälkeenpäin. Kun aloin käyttämään TileStudiota minun piti ensimmäiseksi tehdä grafiikat, joita käyttäisin kenttäsuunnittelussa. Tämä vei jonkin verran aikaa ja oli ehkä hieman turhauttavaakin. Nyt kun ajattelen sovellusta näin jälkeenpäin, sitä olisi nyt tietysti nopeampi käyttää, kun grafiikat ovat valmiit. Ainoat ongelmat TileStudiossa olivat, että sillä on vaikea tehdä mutkia esimerkiksi teitä varten ja sillä suunnitellut kartat eivät ole lainkaan abstrakteja. Karttojen mittasuhteet tulivat ongelmaksi toteutuksessa, koska mittasuhteet esimerkiksi rakennuksista ovat melko epätarkat, ellei haluta tehdä todella suurikokoista karttatiedostoa. Mittasuhteet suunnitelmissa tulivat ongelmaksi kolmessa kentän prototyypissä, joissa tila loppui kesken.

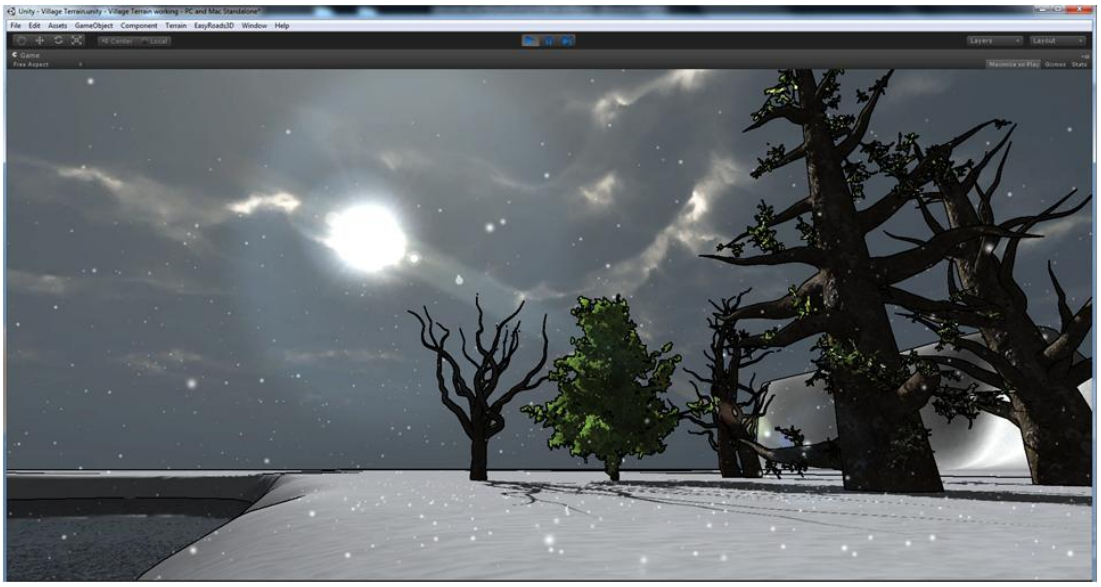


Kuva26. Kenttäsuunnittelu TileStudiolla.

Suppean pelisuunnitelman kirjoittaminen kesti työajanseurantani mukaan 11 tuntia. Suunnittelu kulki myös projektin mukana, vaikka siirryin käyttämään nopeaa prototypointia. Suunnittelu muuttui enemmän graafiseksi, kuin dokumentoinniksi. Tällöin suunnittelun vaatima ajankäyttö väheni ja enemmän aikaa jäi prototypoinnille. Marraskuun ensimmäinen päivä 2010 järjestettiin Opinnäytetöiden edistymiskatselmointi, silloin prototyyppi näytti tältä:


Kuva27. Edistymiskatselmointi marraskuussa (kuva 1)

Edistymiskatselmointi-vaiheessa olin tutkinut pelin ulkonäköä. Totesin että sarjakuvatyyli sopisi hyvin, koska kyseessä on seikkailupeli eikä tätä elementtiä ole käytetty kauhupeleissä aikaisemmin, ainakaan tietääkseni. Edellä olevassa kuvassa on kokeiltu järven pinnan renderöinti-efektiä ja syvyyden omaavia pilviä. Pilvet jäivät mielestäni melko huono laatuisiksi ja päätin tutkia taivaan renderöinti -efektejä myöhemmin. Taivaassa on käytetty Unityn mukana tulevia SkyBoxeja, jotka liitetään kameraan ja tämän jälkeen ne renderöityvät automaattisesti. Auringossa on käytetty valonlähdettä, jossa on linssin heijastus -efekti.

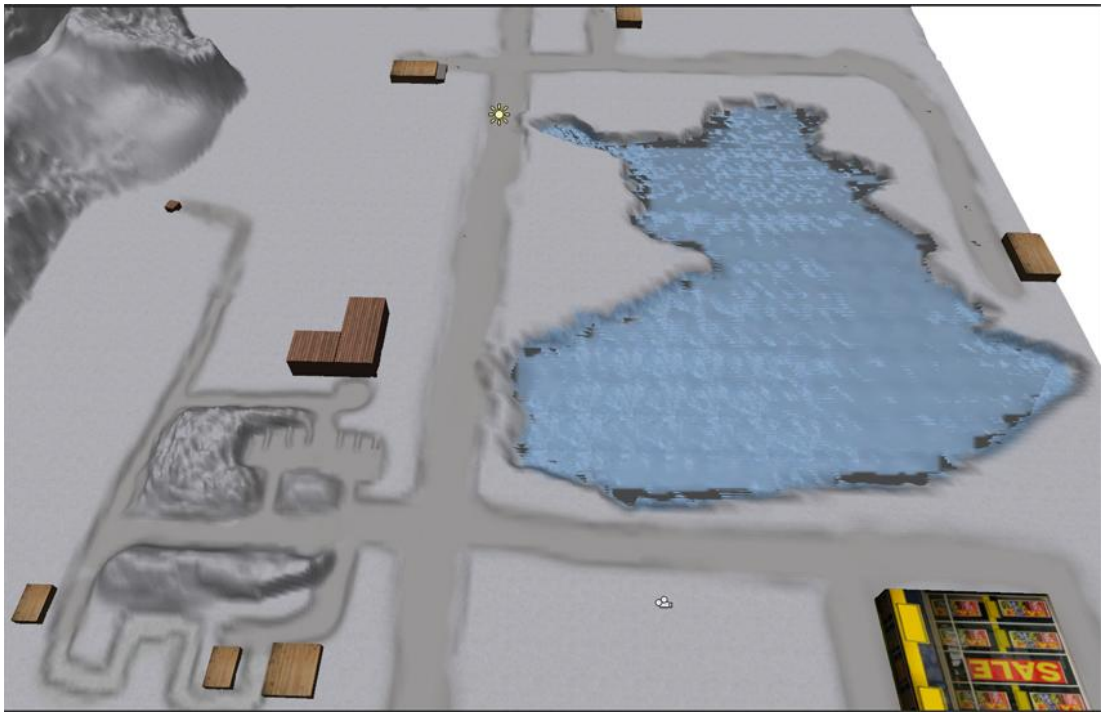


Kuva28. Edistymistkatselmointi marraskuussa (kuva 2)

Toista kuvaa varten vaihdettiin SkyBox ja linssin heijastus -efekti. Efektien lisäksi tein puita käyttäen UnityTerrainEditoria. Tein puut muutamaa päivää ennen esitystä ja pelkästään niiden tekemiseen kului kymmenen tuntia. Puiden teossa en pyrkinyt realismiin vaan kokeilin eri asetuksia ja tutustuin työkaluun. Lopuksi kokeilin vielä miltä pimeys näyttää pelissä ja lisäsin valo-objektin pelaajaan ja siihen sopivan taskulamppu-efektin. Lopputulos näytti mielestäni hyvältä, pimeys tulisi sopimaan pelin ulkonäköön hyvin. Edistymiskatselmoinnissa sain positiivista palautetta luokkatovereiltani ja itseluottamukseni projektiin kasvoi.

Kuva29. Edistymiskatselmointi marraskuu (kuva 3)

Tekniikoiden testauksen jälkeen aloin tekemään prototyyppiä kylästä ja huomasin, että aikaisemmin suunnittelemani kenttä ei toiminut hyvin. Niinpä kokeilin ideoita ja tein muutaman prototyypin. Edistymiskatselmointi vaiheessa mietin järvelle muotoa, mutta mieleeni ei tullut mitään erikoista. Kokeilin liittää joen järveen ja tehdä alueelle puroja, siltikään en ollut tyytyväinen. Myöhemmin marraskuussa mieleeni välähti, tehdä järvestä suomen kartan muotoinen. Parin iteraation verran tein järveä UnitynTerrainEnginellä, mutta se oli melko kömpelö tapa tehdä mutkikasta järveä. Ajattelin että järvi olisi helpompi tehdä käyttäen 3D-mallinnustyökalua, joten sijoitin järven tekemisen seuraavaan vaiheeseen.




Kuva30. Kenttäsuunnittelu marraskuun lopussa.

Projektin alkamisesta joulukuulle asti, pidin karkeaa työajanseurantaa calc-laskentataulukossa. Opinnäytetyön raporttia aloin työstämään 14. lokakuuta ja jatkoin raportin kirjoittamista koko projektin ajan. Joulukuun lopulla päätin kirjoittaa ajantasalla olevan version suppeasta pelisuunnitelmasta englanniksi. Dokumentin kirjoittamiseen kului 18 työtuntia, kolmena peräkkäisenä päivänä. Dokumentointi on mielestäni hyvä asia tässä projektissa, koska silloin tulee pohdittua miten asiat toteutetaan, ennen kuin niitä aletaan toteuttaa. Dokumentoinnin haittapuolena on, että prosessi käyttää aikaa, jota voitaisiin käyttää muuhun. Käänsin dokumentit englanniksi, koska projektin jatkokehityksessä tiimi tulee olemaan kansainvälinen. Dokumentoinnin jälkeen jatkoin kenttäsuunnittelua.

Kuva31. Kenttäsuunnittelu joulukuun lopulla

Kokeilin lisätä prototyyppiin pieniä yksityiskohtia, kuten kuvan alaosassa olevan sinisen alueen. Yritin sijoittaa pientä solaa, jossa kulkisi jäätynyt puro. Suunnittelin myös hautausmaan sijoittamista, kuvan oikeaan laitaan. Kartan yläreunalle tulisi jyrkänne ja vuoristonäkymä. Alueen läpi kulkeva tie tulee pujottelemaan alas vuoristolta. Kartan oikeaan osaan sijoittamani seinämä, osottautui muuten huonoksi ideaksi, vaikka se auttoi suunnistamisessa.
Dokumentin kirjoittamisen jälkeen kesti vajaan viikon verran, että suppea

pelisuunnitelma jäi paljon jälkeen prototypoinnin aiheuttamasta kehityksestä. Tällöin päätin sijoittaa aikani enemmän prototypointiin. Joulun aikana etsin jouluun liittyviä tarinoita, joita voisin esittää pelissä, sekä kirjoitin alustavan liiketoimintasuunnitelman pelille.

### 2.3.4 Tekovaihe

Pelin konkreettinen työstäminen alkoi Unity projektin valmistelulla lokakuun puolessa välissä. Järjestin kesällä keräämäni assetit kansioihin tyypeittäin ja aloitin vaiheen kertaamalla Unity terraineditorin toimintoja. Sen käyttäminen palautui mieleen nopeasti, muutamalla lyhyellä esimerkkivideolla, jotka katsoin YouTube:sta. Kentän tekemisessä tuli tärkeäksi valita sopiva tientekemistyökalu-liitännäinen Unityyn. Vertailin maksullista ja ilmaista vaihtoehtoa ja päädyin maksulliseen EasyRoads 3D:hen. Teiden lisäksi sillä pystyi tekemään puroja ja siltoja, mutta sen hienoin ominaisuus oli saada tie mukautumaan Unityn maastoon ja asettamaan objekteja tien reunaan halutulla välillä. Tienreunaobjektit voivat olla kaiteita tai katuvaloja.

Kevään aikana olin hankkinut 2D-tekstuureja varten tehokkaan liitännäisen Above and Beyond Softwarelta nimeltänsä Sprite Manager, sekä pelin tallennusta varten EZ Game Saver -liitännäisen. Sprite Manager on 2D-tekstuurin animointi liitännäinen, jolla pystytään yhdistelemään tekstuureja ja vähentämään Draw Callseja. Ez Game Saver on pelin tallennus liitännäinen, jolla voidaan tallentaa pelin objektien tilannetiedot. Sprite Manager-liitännäinen oli hyvä tekstuureja varten ja sillä pystyi tekemään myös valikoita ja muita GUI-elementtejä.

Sprite Manager tuli hyödyllisemmäksi tässä projektissa, kun hankin Above and Beyond Softwarelta EZ GUI -liitännäisen, joka on yhteensopiva Sprite Managerin kanssa. EZ GUI on työkalu, jolla voidaan tehdä GUI-elementtejä, kuten painikkeita, palkkeja ja valikoita. Se ei ole valmis käytettävä GUI-paketti, vaan se on työkalu, jolla voidaan tehdä halutun kaltaiset GUI-elementit. Sprite Managerin avulla saadaan animoituja valikoita, jotka ovat kevyitä. Pelien tekemisessä elementtien kevyys on erittäin tärkeää suorituskyvyn kannalta. Ulkoisten liitännäisten avulla pystyin nopeuttamaan kentän prototypointia, käyttämällä valmiita kehyksiä, joilla pystyn suoraan

tekemään sisältöä.

Myöhemmin ostin myös Advanced Ragdoll -liitännäisen, joka helpottaa räsynukke-simulointia ja sen yhdistämistä animointiin. Ajattelin käyttää liitännäistä kaatumisen simuloinnissa. Liitännäinen ei ole ehkä olennainen eikä hyödyllinen projektille, mutta räsynukke-simulointi on ollut mielestäni yksi hienoimmista ja hauskimmista ominaisuuksista peleissä ja haluan sisällyttää sen myös omaan peliini. Muita liitännäisiä joihin tutustuin, olivat Angry Antin polunhaku-liitännäinen ja Behavior-käyttäytymisliitännäinen, jotka ovat tekoäly-liitännäisiä. Harmikseni kumpikaan ei toiminut uuden Unity 3-version kanssa joulukuussa, joten jätin tekoälyn kehityksen pelin jatkokehitykseen. Nyt keväällä molemmat toimivat Unity 3-version kanssa, mutta päätin uudelleen siirtää tekoälyä jatkokehitykseen. Syy tähän on Unity Technologiesin ilmoitus yhteistyösopimuksesta saksalaisen Artificial Technologyn kanssa. UT aikoo integroida EKI One A.I -tekoäly-väliohjelmiston Unityyn. EKI One A.I tulee olemaan kattava tekoäly-väliohjelma, jonka uskon olevan parempi kuin Angry Antin liitännäiset.

Unityyn on tarjolla paljon ilmaisia liitännäisiä, mutta yhteisön tekemät maksullisetkin liitännäiset tuntuvat lähes ilmaisilta. Seuraava toteutus ongelma oli selvittää, miten saataisiin realistisempi taivas kuin SkyBox:eilla tehdyt staattiset taivaat. Löysin Unityn keskustelupalstoilta Skydome-menetelmää kehittäneen Pixel Studiosin. Skydome-menetelmällä renderöidään taivas halkaistun pallon sisäosaan ja tällä menetelmällä pystytään muokkaamaan taivaan renderöintiä reaaliajassa. Tällä tavalla voidaan myös simuloida ajan kulkua Auringon laskulla ja nousulla. Pixel Studiosin lisäksi pilvien simulointia kehitti keskustelupalstojen jäsen 3dDude. Jätin taivaan simuloinnin tauolle ja odotin vuoden vaihteen yli. Tammikuussa Three Times Nothing julkaisi UniSky-liitännäisen, joka sisälsi myös sään simuloinnin (Six Times Nothing 2011).

UniSky oli ainut maksullinen näistä liitännäisistä, mutta se oli mielestäni laadultaan paras, joten hankin sen lisenssin. Ennen UniSkyn hankintaa kentässäni oli lumisade-efekti, jonka tein käyttämällä Unityn Particle-komponenttia. Halusin peliin myös Decal-järjestelmän, jolla olisi helppo

asettaa tekstuureja esimerkiksi maastoon. Tätä voitaisiin käyttää jalanjälkien jättämisessä tai kun lumipallo osuu seinään niin siitä jää jälki, joka erottuu seinästä 3D:mäisenä bumpmapien avulla, eikä pelkkänä litteänä tekstuurina. Löysin Unityn Asset Storesta Frameshiftin kehittämän Decal-liitännäisen, jolla pystyttäisin tekemään myös syviä jalanjälkiä, joiden toteutus vaatisi muuten paljon aikaa. Liitännäinen oli kuitenkin maksullinen ja päätin jättää sen hankinnan myöhemmälle. Unityssä mielestäni sen ominaisuuksiakin parempaa on sen yhteisö, jossa tehdään työtä yhteisten päämärien eteen, kuten laadukkaiden työkalujen kehittämiseen. Usein maksullisten liitännäisten lisenssit eivät ole edes projekti sidonnaisia, joten niitä voidaan käyttää kaikissa projekteissa.

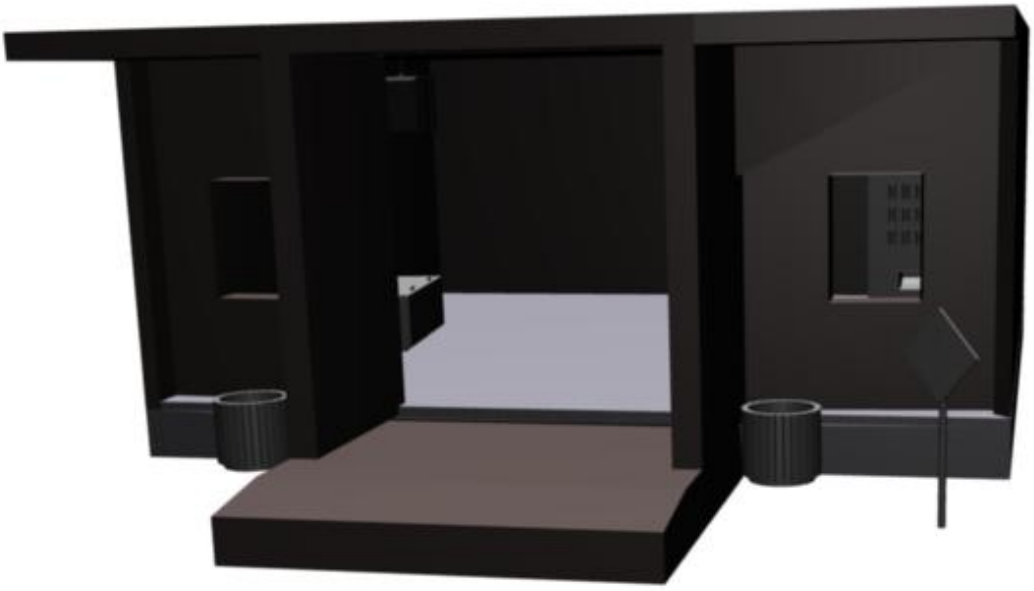
Tekovaiheessa tuli tutkittua paljon menetelmiä ja liitännäisiä, joilla voitaisiin toteuttaa ideoita. Tutkimisen lisäksi peliä toteutettiin ja pelkästään kentän prototyyppejä syntyi kahdeksan kappaletta. Kentän prototypoinnissa aloitettiin usein tyhjältä pöydältä ja kokeiltiin useita eri ideoita. Rakennusten sijoittamista eri paikkoihin kokeiltiin ja erityisesti välimatkoja kasvatettiin, jokaisessa versiossa. Lopulta pelin alkaminen siirtyi kokonaan muualle, kuin mitä oli ajateltu ja tehty tähän mennessä. Peli tulee alkamaan pienessä kaupungissa, jossa pelaajan hahmo käy koulussa. Koulussa pelaajalle opetetaan pelin perus-mekaniikat, sekä kerätään tietoa pelaajasta, jolla pelaajan hahmoa voidaan muokata. Joulukuun lopussa tein pienen ilotulitus-minipelin, jossa pelaaja voi kävellä ja asettaa ilotulitteita pulloihin ja sytyttää ne painamalla tiettyä näppäintä.

Kuva32. Ilotulite-demo

Asettaessa raketteja ohjelma arpoo räjähdysefektin, joka voi olla neljää eri kokoa ja väriä. Sytytyksen jälkeen ohjelma arpoo sytytyslangan palamisen keston pieneltä välitä, jonka jälkeen käynnistää tuli-hiukkasefektin ja soittaa ääniefektin. Langan palamiskeston jälkeen raketti lentää ylöspäin toistaen erin ääniefektin ja räjähtää arvotulla lentoajalla. Sain Unity-yhteisöltä posiitivista palautetta ilotulite-demosta ja lopulta julkaisin sen yhteisön käytettäväksi.

Alkuvuodesta sain valmiiksi seitsemännen kenttä prototyypin, jonka jälkeen aloin tekemään 3D-malleja. Aloitin tekemällä 3D-mallin huoltoasemasta, jota varten mallinsin aseman ja polttoainesäiliö perävaunun.

Kuva33. Polttoainesäiliö



Kuva34. Huoltoasema

Tämän jälkeen aloin mallintamaan koulua ja sen objekteja, kuten pöytiä ja tuoleja.

Kuva35. Koulun objektit

3D-mallinnuksen alkuvaiheessa minulle tuli ongelmana 3D-mallien siirtäminen Unityyn. Usein kävi niin että jostain suunnasta seinää näki kokonaan läpi ja tämä oli ehkäpä suurin este mitä projektissa tuli vastaan. Ongelma kuitenkin korjautui, kun aloitin rakennuksen puhtaalta pöydältä ja varmistin, että kaikki seinien lakipisteet olivat kiinnitettyinä toisiinsa. Sama ongelma toistui kun tein 2D-tasoja lippua varten, lippu oli näkymätön toiselta puolelta. Korjasin ongelman tekemällä UV-unwrapin, jossa asetin UV-koordinaatit oikein ja liitin tekstuurin siihen. Koulun seinissä auttoi myös pintojennormaalien uudelleen laskeminen. Koulun 3D-mallinnuksen jälkeen tein 3D-mallin Suomen muotoisesta järvestä, joka tulee kylän keskelle.

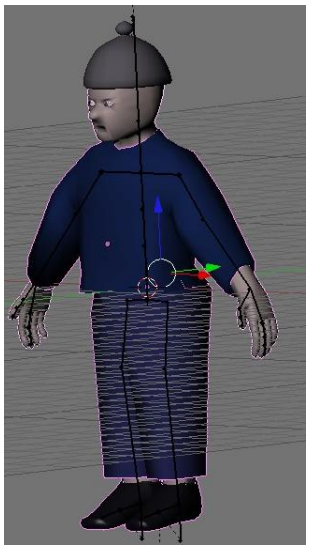
Kuva36. Suomen muotoisen järven 3D-kehys

Tein myös jäänpintaa 3D-malliin, mutta yksityiskohtien toteuttaminen alkoi viedä paljon aikaa, joten siirsin niiden toteuttamisen jatkokehitykseen. Tämän jälkeen siirryin tekemään lippua, johon lisäsin Unityn vaatefysiikkakomponentit. Lopputulos oli kaunis, mutta huomasin että vaatteiden reaaliaikainen mallinnus vie paljon tehoja, joita voitaisiin käyttää muualla.

Tämän jälkeen aloin 3D-mallintamaan pelaajan hahmoa. Tämä osuus oli projektissa totettamisen kannalta haastavinta. Toteutin kaksi eri esimerkkiä, joista ensimmäisen oli toteuttanut Ryan, Blender Summer of Documentationia varten (Ryan 2006). Ensimmäinen esimerkki ei tehnyt hahmosta juurikaan ihmismäistä. Hahmon kasvot muistuttivat enemmänkin sammakkoa kuin ihmistä, mutta se opetti hahmon mallinnuksen perusteet kohtalaisen hyvin. Tämän jälkeen yritin käyttää oppimiani mallinnustekniikoita ja tehdä hahmon alusta loppuun asti. Sen jälkeen yrin vielä kaksi kertaa, kunnes luovutin ja otin seuraavan esimerkin. Seuraavan esimerkin oli kirjoittanut Jonahthan Williamson, sen nimi oli The Basics of Character Modeling (Williamson 2010). Löysin tämän esimerkin Blenderin tutorials-sivulta ja sen lopputulos näytti käytännöllisemmältä kuin ensimmäisen esimerkin lopputulokset. Esimerkkiä seuraten viidennellä hahmon mallinnus yrityksellä sain mallinnettua humanoidin, johon olin tyytyväinen.

Hahmon mallinnuksessa käytin mirror-modifieriä, jonka avulla minun tarvitsi mallintaa vain puolet hahmosta. Esimerkin valmistuttua mallinsin hahmolle vaatteet ja silmät. Mallinnettuani hahmon asetin sille luurangon ja maalasin

Weight Paint-tilassa alueet, joihin jokainen luu vaikuttaa.



Kuva37. Pelaajan Hahmo Blenderissä

Nyt hahmo oli valmis animoitavaksi, mutta koska olin animoinut objekteja aikaisemmin ja ajattelin että olisi hyvä harjoitella ulkoisten animointien tuomista 3D-malliin Unityssä. Animointien asettaminen malleihin on hyödyllistä, koska voidaan käyttää samaa animointia useissa malleissa. Normaalisti animointi tulee 3D-mallin mukana tuotaessa Unityyn, mikäli sillä sellainen on, mutta tällä kertaa jätin animoinnin liittämisen 3D-malliin manuaaliseksi.

Kuva38. Pelaajan Hahmo Unityssä

Unityssä pelaajan hahmosta saatiin nopeasti tehtyä sarjakuvamaisempi käyttämällä ToonShaderiä, mutta samalla hahmosta tuli entistä pelottavamman näköinen. Maaliskuussa siirryin käyttämään uutta tietokonetta, jonka suorituskyky nopeutti työkalujen käyttämistä.

Viimeisintä kenttä protyyppiä varten päätin kokeilla korkeuskäyrien tuomista Google Earthsitä Unityyn. Korkeuskäyrien asettaminen Unityn terrainiin onnistui ja lopputulos oli hieno. Tämän jälkeen tein Unityssä puita käyttäen Unity tree creator-työkalua. Puiden tekoa varten olin kerännyt kuvia talven aikana, joita voisin käyttää tekstuureina, sekä mallina. Maaston nopeaa teksturointia varten latasin Unityn kotisivuilta Unity terraintoolkitin, jolla pystyin asettamaan tekstuureja korkeusarvoittain. Tämän jälkeen käytin Unityn terrain editorin Mass place trees-painiketta, joka sijoittaa puita sattuman varaisesti kenttään. Tämän jälkeen siistin puut vuorilta ja asetin sinne paremmin sopivia puulajeja. Kenttä alkoi hiljalleen näyttää elävämmältä.

Seuraavaksi otin käyttöön Unity Terrain Toolsin Easy Roads 3D-liitännäisen ja asetin teitä kenttään. Kenttä tarvitsi lisää rakennuksia, mutta jätän kaupungin mallinnuksen jatkokehitykselle. Tämän jälkeen mallinsin muutamia mökkejä, jotka asettelin vuoristoon. Pyrin pitämään 3D-mallit yksinkertaisina, mutta oikeassa mittasuhteessa. Asettelulla pyrin saamaan helposti jatkokehitettävän prototyypin, johon tarvitsisi vain liittää paremmat 3D-mallit. 3D-mallinnus vei paljon aikaa, mutta se oli erittäin palkitsevaa. Seuraavaksi asettelin pelaajan kenttään ja liitin häneen Third Person Controller-komentosarjan, joka liikuttaisi hahmoa ja pitäisi kameran sopivalla etäisyydellä. Hahmon liikkumista varten tehtäisiin omia liitännäisiä myöhemmin. Liitännäinen voisivat olla esimerkiksi komentosarja, joka päivittäisi hahmon kestävyyspalkkia tämän juostessa tai levätessä ja estäisi hahmon juoksemisen mikäli palkki olisi tyhjä. Prototyypin virtuaalimaailma alkaa näyttää hyvältä ja kun Unity Technologies integroi tekoäly-väliohjelmiston, tulee pelin sisällön kehittäminen olemaan helppoa.

# 3 TESTAUS

## 3.1 Käytettävyystestaus

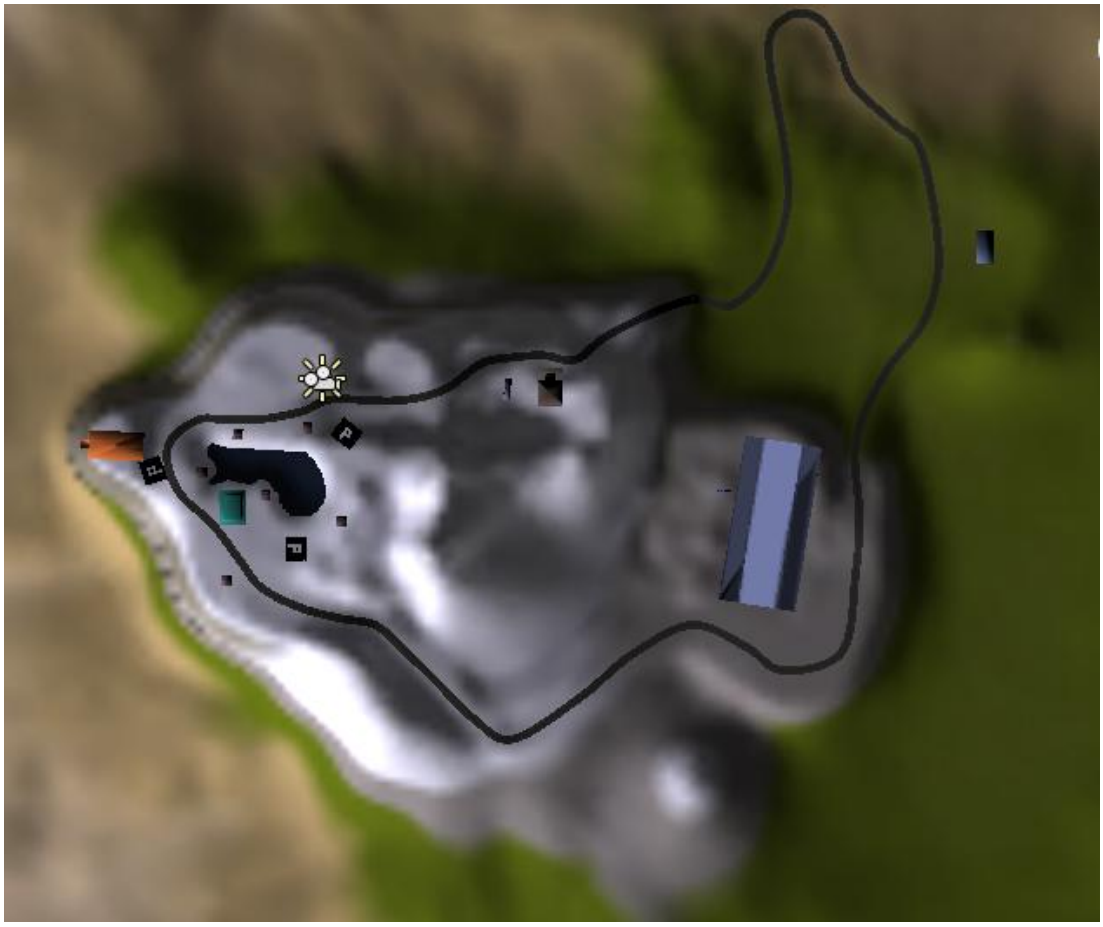
Yksi suurimmista tietokonejärjestelmien ongelmista, jonka kehittäjät kohtaavat, on varmistaa onko tuote mitä loppukäyttäjä haluaa ja tarvitsee (Usability Engineering 2000, 1). Käytettävyystestauksessa testataan: pelin valikoita, muuta GUI:ia, näppäimiä, gameplaytä ja kenttäsuunnittelua. Testauksen tavoitteena on parantaa opittavuutta ja lisätä valikoiden tehokkuutta, sekä vähentää käyttäjän turhautumista ja tahattomien virheiden tekemistä pelissä. Tämän testauksen jälkeen järjestetään erikseen pelattavuus-testaus, koska pelattavuus on niin tärkeä elementti peleissä. Käytettävyyden testaajia pyrin saamaan viidestä kymmeneen henkilöä. Testaajille annettiin lyhyt johdanto käytettävyyteen, jotta he pystyivät arvioimaan sitä paremmin. Käytettävyys testaus siirrettiin jatkokehitykselle, koska GUI-elementien toteutukselle ei jäänyt tarpeeksi aikaa. Käytettävyys testaus korvattiin protyypin rasitustestauksella, joka esitetään tulokset osiossa.

## 3.2 Pelattavuustestaus

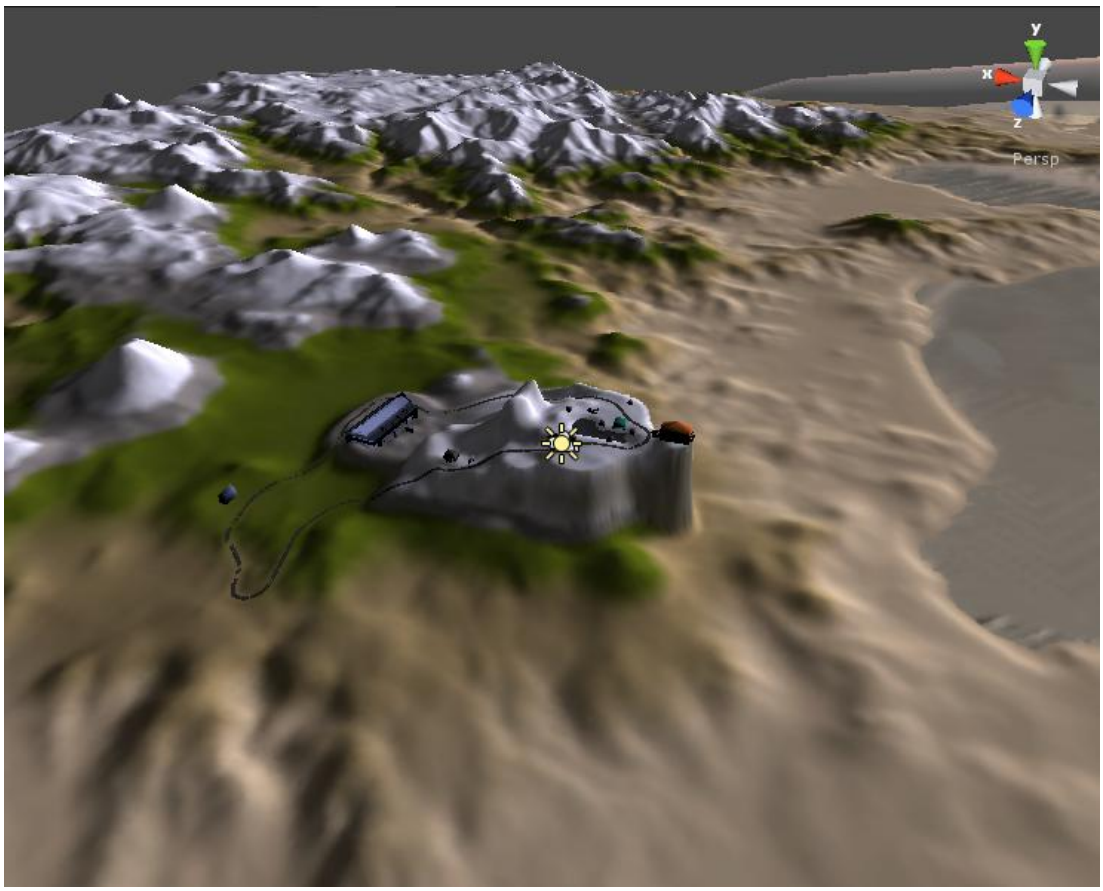
Pelin pelattavuustestauksesta tehtiin julkinen palaute-lomake, jonka tavoitteena oli saada yleinen mielipide pelistä ja määritellä kohdeyleisö, johon tutustua. Lomakkeen tuloksista toivottavasti erottuvat ikäryhmät, jotka ovat kiinnostuneet pelistä. Tämä on kuin asiakastutkimus, jolla saan esiin mielipiteitä mahdollisilta tulevilta asiakkailta. Tietojen avulla pystyn ohjaamaan pelin kehitystä asiakaslähtöisemmin ja asettamaan hinnat, siten että pelaajat ovat tyytyväisiä (Liite1.) Pelin gameplay saatiin hyvin suunniteltua, mutta suurinosa totetuksesta jouduttiin jättämään jatkokehitykselle. Testaus aiotaan silti järjestää RAMK:ssa. Pelattavuustestaus korvattiin keräämällä mielipiteitä pelin ulkoasusta. Palautteessa tuli esille hyviä jatkokehitys ideoita ja samalla saatiin rakentavaa kritiikkiä. Esimerkiksi rannan sijoittamista lumisen vuoren läheisyyteen ei pidetty järkevänä.

# 4 TULOKSET

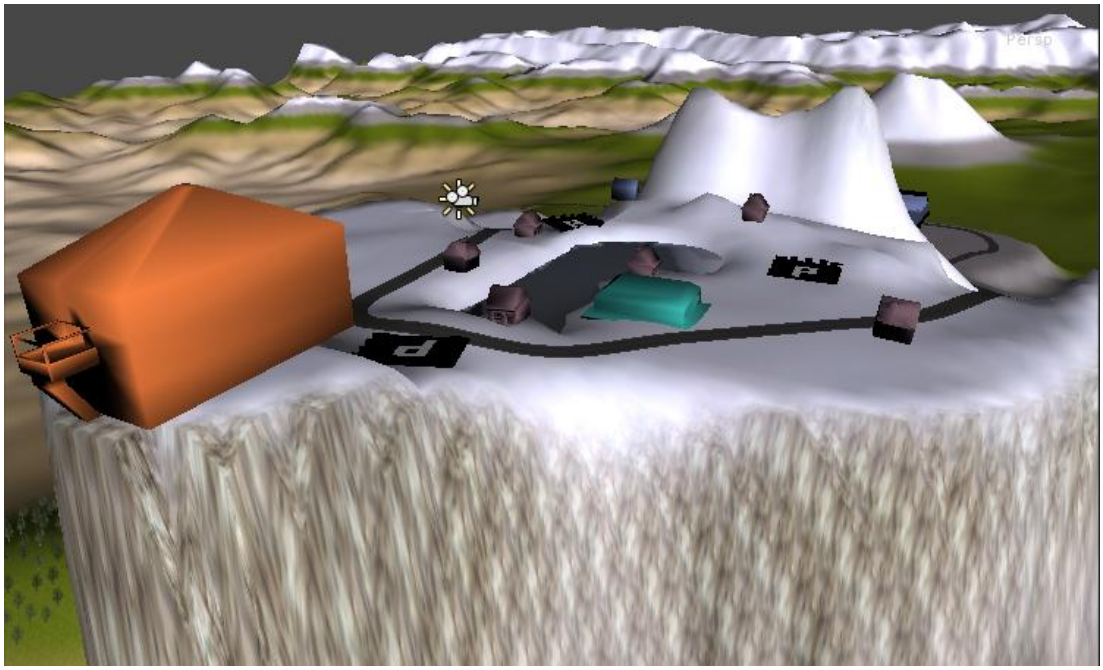
Pelistä valmistui toimiva prototyppi, joka näyttää mielestäni hyvältä. Pelin grafiikasta saatiin selville miltä se voisi näyttää ja tuntua. Pelin kenttä päätyi edellisen version kokoiseksi, joka oli pinta-alaltaan 20 neliökilometria ja 800 metriä korkeimmillaan. Viimeinen versio toteutettiin alusta loppuun asti 8ssa tunnissa käyttäen kerättyjä kokonaisuuksia. Vastaan tuli muutamia ongelmia, kun liitettiin uusia liitännäisiä joita ei ollut testattu aikaisemmin. Tästä johtuen kuvissa näkyy graafisia häiriöitä. Häiriöt on listattu jatkokehityssuunnitelmaan.



Kuva39. Kuva ylhäältäpäin.



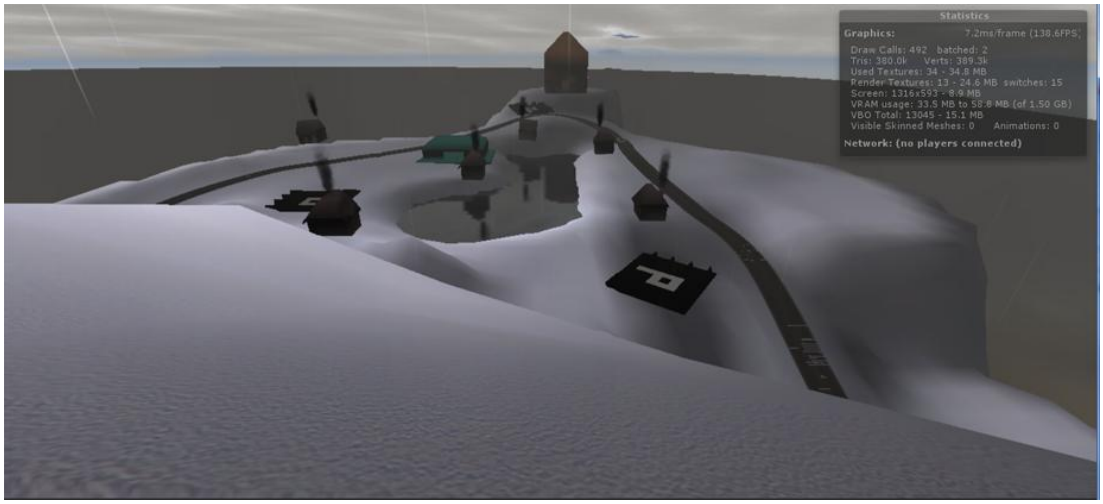

Kuva40. Yleiskuva 1




Kuva41. Yleiskuva 2

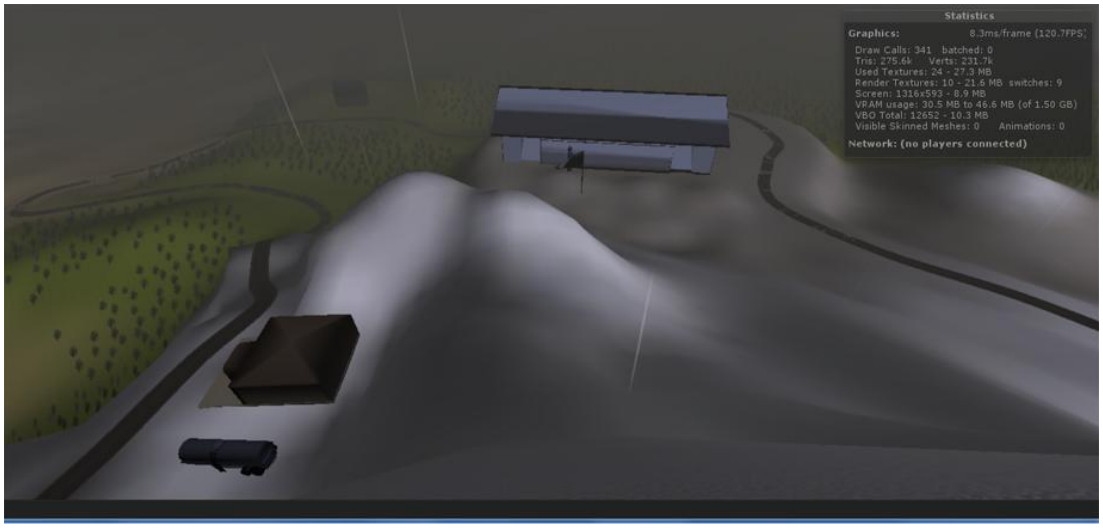
Edellä olevista kuvista näkyy rakennusten sijoittuminen pelissä. Kentän oikeanpuoleisin rakennus on pelaajan koti, jossa peli alkaa. Toinen sininen rakennus oikealla on koulu, jossa pelaajaa opastetaan peliin. Pieni ruskea

rakennus keskellä, on huoltoasema, jonka vieressä on polttoainesäiliö. P-merkkiset laatat ovat pysäköintialueita. Vihreä rakennus on kauppa, josta pelaajan tulee ostaa mökkiinsä ruokaa. Pienimmät neliöt ovat mökkejä, joissa pelaaja voi asua. Vasemmanpuoleisin rakennus on hotelli, jossa pelaaja voi myös asua. Hotellista on näkymä merelle.




Kuva42. Laskettelumäen loiva puoli

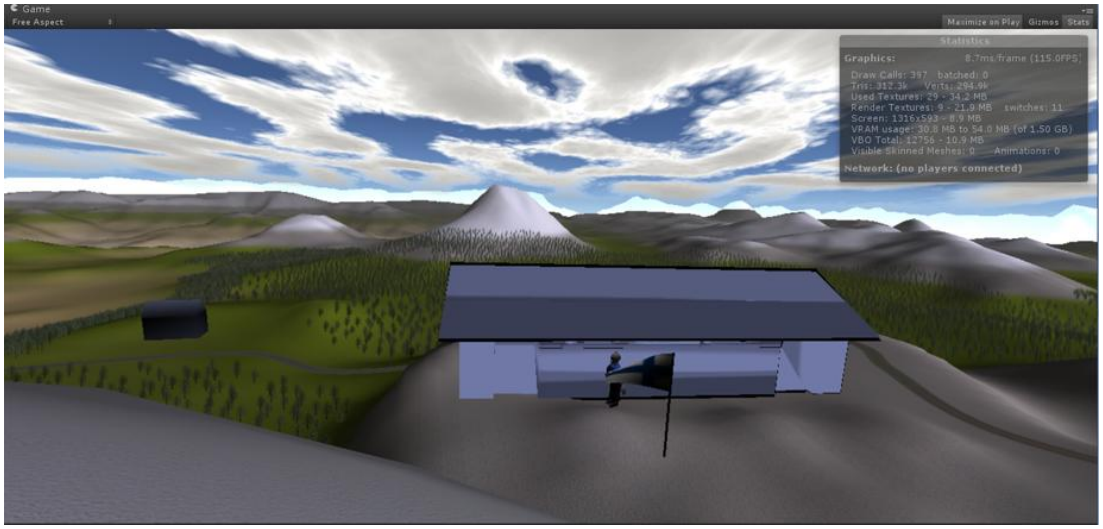
Edellä olevassa kuvassa näkyy kylä laskettelumäen päältä. Meri ei näy kuvassa, koska vesisateen aikana sumun määrä kasvaa horisontissa. Kuvan Draw Callsien määrä on lähes 500. Mökkien savupiipuista nousee savua, josta tuli mielestäni liian tummaa. Kylässä käytetyt 3D-objektit toteutettiin viimeisen iteraation aikana, jonka takia ne jäivät kaikkein yksinkertaisimmiksi.




Kuva43. Laskettelumäen jyrkkä puoli

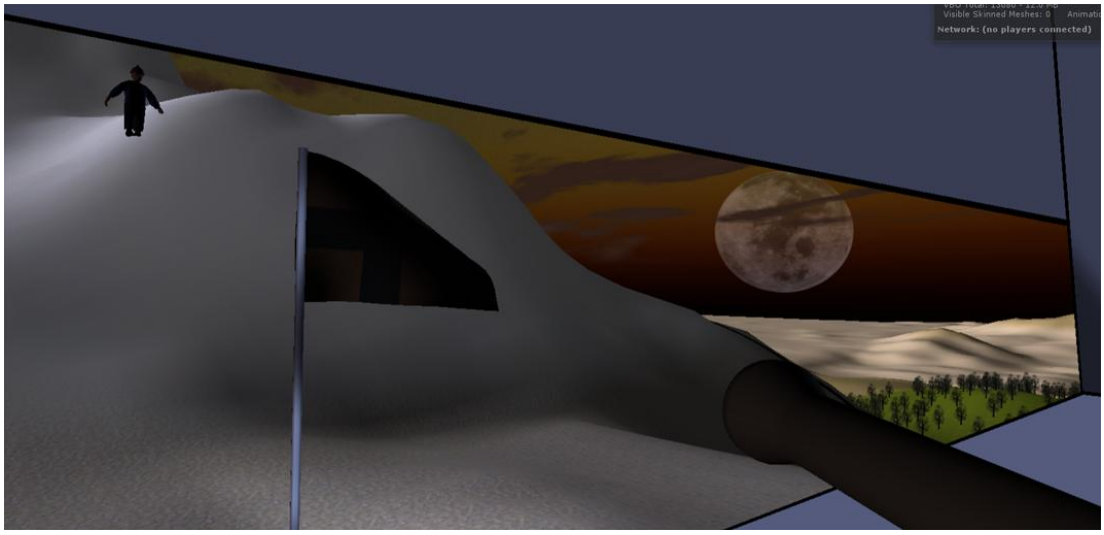
Laskettelumäen ideana, oli tarjota pelaajalle nopea tapa liikkua kylältä

koululle, käyttämällä suksia tai pulkkaa. Tuolla pelin hetkellä lähetetään 341 Draw callsia sekunnissa. Tavoitteenani oli pysyä alle 500n Draw Callsin, jotta pelattavuus pysyy sujuvana.




Kuva44. UniSky käytössä.

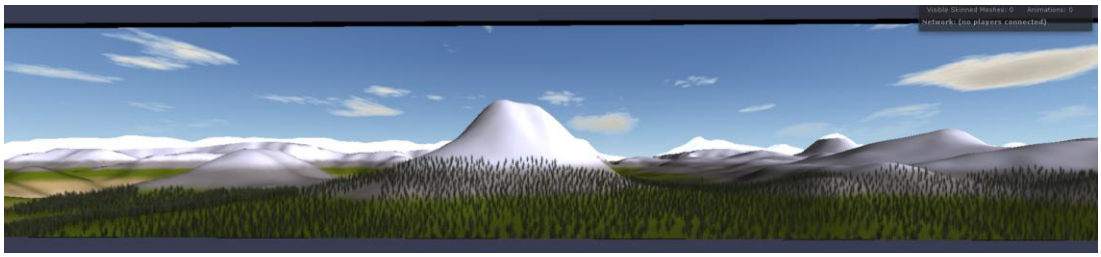
UniSky tarjoaa vaihtelevan sään ja valoisuuden. UniSkytä testatessani yllätyin kuinka hyvältä pilvet näyttävät. Yksi graafinenhäiriö syntyi kun liitin UniSkyn projektiin. Häiriö näkyy edellisessä kuvassa. Kuvan kaukaisimmat vuoret törmäävät taivaankannen kanssa ja tämä saa vuoret näyttämään oudon kirkkailta. Uskoisin että tämä virhe, on helppo korjata muuttamalla UniSkyn asetuksia.




Kuva45. Koulun ikkunanäkymä kuun noustessa

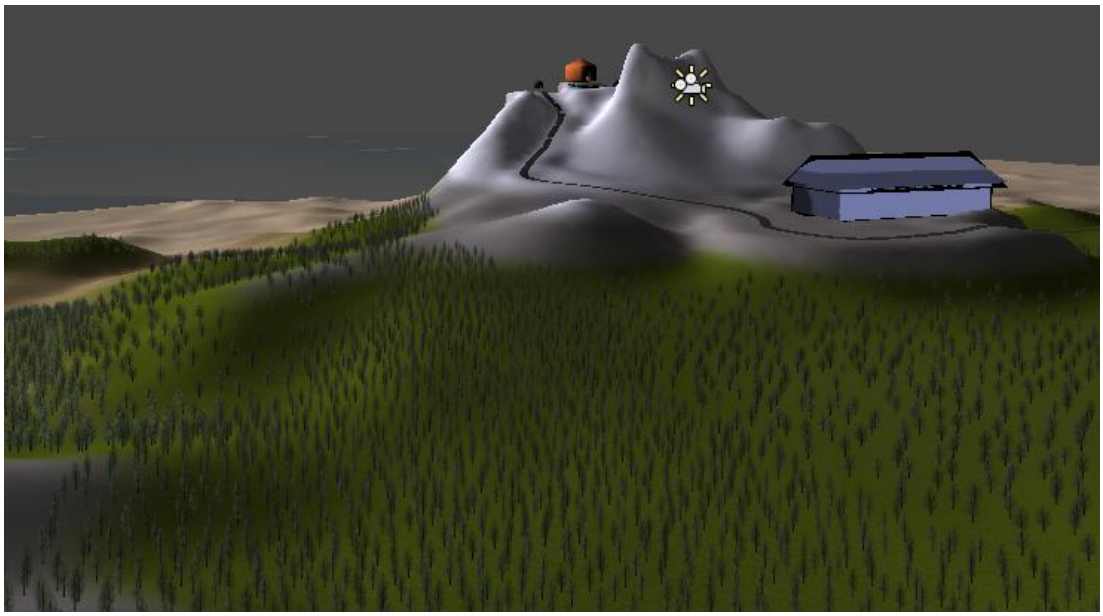
Pelaajan hahmoa en saanut suoraan liikkumaan Unityssä. Huomasin, että tätä varten minun olisi pitänyt liittää hahmon luut Skinned Mesh renderiksi,

joka on animoitujen hahmojen renderöijä. Illan koittaessa Aurinko laskee ja taivas muuttuu punertavaksi, tämän jälkeen kuu nousee ja taivas muuttuu hetkeksi pimeäksi. Kuu valaisee kenttää hieman ja taivaalla alkaa näkymään tähtiä.



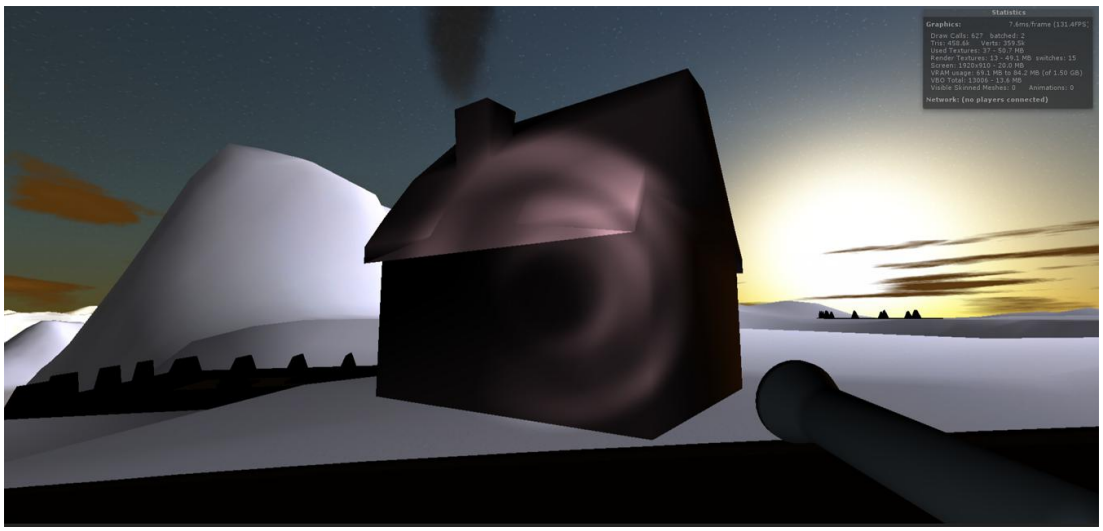
Kuva46. Koulun ikkunan näkymä koulun taakse

Koulun takana on paljon metsää, jonne pelaajan ei haluta eksyvän. Metsästä on pyritty tekemään luotansa pois työntävä. Alueet, joihin pelaajan halutaan menevän, asetetaan harvempaa metsää tai aukeaa.



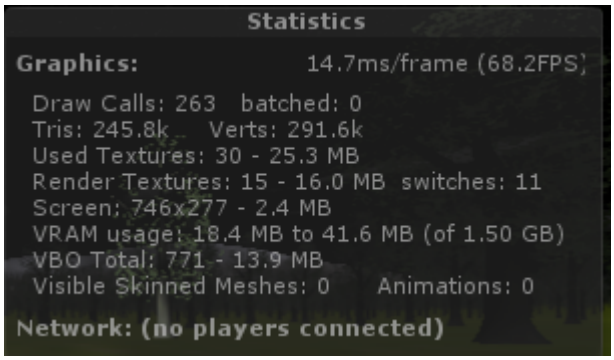
Kuva47. Näkymä merelle

Edellä olevan kuvan yläosassa näkyy meri. Merellä on tuulesta riippuen erikokoisia aaltoja. Taivas ei näy, koska kuva on otettu scene-ikkunasta. UniSky aiheutti myös ongelmia meren renderöimisessä. Uskon, että meri alkaa toimia, kun taivaan aluetta kasvatetaan meren yli. Metsää on sijoitettu pelissä ainoastaan vihreille alueille. Myöhemmin vuoristoon tehdään talviseen maisemaan sopivia puita.



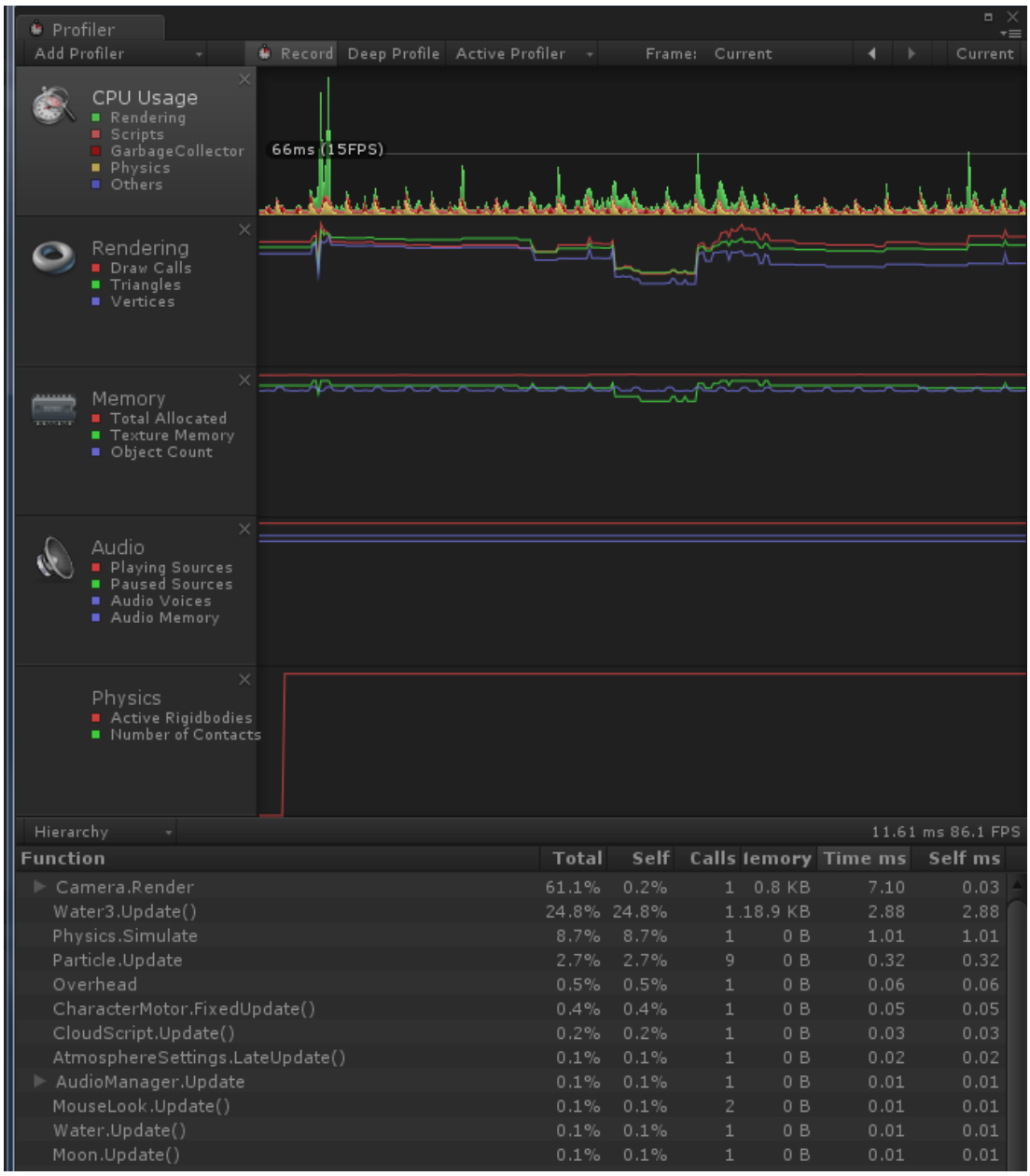
Kuva48. Auringonlasku

Pelaajalla on edellisessä kuvassa kädessään taskulamppu. Kuvan vasemmalla näkyy parkkialue, joka jakaantuu osiin korokkeilla. Kuvassa Draw Callsien määrä on 427. Projektin loppuvaiheessa yritin rasittaa peliä ja kerätä tilastoja. Pelin rasitustestauksen tuloksena huomasin, että peli toimii hyvin ja sen suorituskykyä voitaisiin entisestään optimoida.



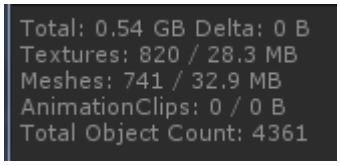

Kuva49. Rasitustesti 1

Juoksin puiden ympäröimällä alueella ja Draw Callsien määrä oli vähemmän kuin kylässä seisoessa. Videomuistia käytettiin alle 50 megabittiä ja Draw Callsien määrä pysyi alle 300n. Kuvataajuus pysyi myös kohtalaisena ja peli toimi hyvin. Unity pystyy optimoimaan metsän puut erittäin hyvin ja sitä voitaisiin käyttää vaikkapa metsien simulointiin.




Kuva50. Rasitustesti 2

Jatkoin metsässä juoksemista ja katsoin mitä profiler näyttää. Profilerin mukaan suurin osa tehoista menee meren päivittämiseen, vaikka olen kaukana sisämaassa. Tämä voitaisiin optimoida käyttämällä Umbran Occlusion Cullingia.




Kuva51. Muistinkäyttö

Rasitustestin aikana en onnistunut käyttämään paljoa yli 500 megabittiä

Random Access Muistia. Seuratessani tilastoja, huomasin että FPS pysyy välillä 60—180, kun meri ei ole näkyvissä. Kun katsotaan merelle, niin Frame Rate putoaa jyrkästi, noin 40 tienoille. Meren suorituskyvyn raskaus johtuu suurilta osin vedenpinnan heijastumisesta, jolloin renderöinnin määrä lähes kaksinkertaistuu. Peli on silti pelattava, mutta meri näyttää olevan tärkein optimoinnin kohde.

# 5 JOHTOPÄÄTÖKSET

Prototyyppi toimi yllättävän hyvin. Viimeisessä versiossa en käyttänyt Umbraa enkä Beästiä ja silti FPS ja Draw Callsien määrä pysyi kohtalaisena. Aluksi halusin välttää valoitus ongelmia UniSkyn kanssa, joten otin valokartoituksen pois käytöstä. Kokeilin kuitenkin lopulta Beästiä ja se toimi hyvin UniSkyn kanssa. Umbraa en käyttänyt, koska sen asettaminen kenttään kestää useita tunteja. Kentästä riippuen laskelmointi voi kestää jopa yli kymmenen tuntia. Onneksi Umbran kanssa voi työskennellä samanaikaisesti, mutta mielestäni Umbra tarvitsee itsessään optimointia ja kunnon käyttöohjeet.

Mielestäni ToonShaderin käyttäminen ei ollut välttämätöntä, peli näyttää valoisana aikana tarpeeksi mukavalta. ToonShaderiä käyttämällä voidaan piirtää ääriviivoja reunan tunnistuksen avulla 3D-malleihin, kuten maastoon tai värin perusteella myös pilviin.

Projekti oli kokonaisuudessaan onnistunut, opin projektin aikana niin paljon, että sitä on vaikea sanoin kuvailla. 3D-mallintaminen oli minulle kokonaan uutta ja opin mielestäni perusteet hyvin. Unityn käyttämisessä opin optimoimaan paremmin, sillä yhdessäkään aikaisemmassani projektissa kuvataajuus ja Draw Callsien määrä ei ole pysynyt samantasoisena projektin laajuuteen nähden. Tutustuin myös Unityn moninpeliominaisuuksiin, joita aion hyödyntää pelin jatkokehityksessä. Pelin tekemisen laajuus selvisi minulle vasta projektin puolessa välissä. Sen takia olen varannut runsaasti aikaa pelin jatkokehitykselle, jotta pelistä tulee laadukas. Projektin alkuvaiheessa suoritustavoitteet olivat erittäin korkealla, mutta melko nopeasti jouduin karsimaan asioita jatkokehitykselle. Onneksi olin tietoinen, aikataulusuunnittelun epäluotettavuudesta ja aloitin valmistelun puolivuotta etukäteen. Aikataulusuunnittelu on epäluotettavaa, koska työtehtävien määrien ja kestojen arviointi on lähes mahdotonta. Jopa viimeisenä iltana, jona valmistelin tätä projektia, tuli vastaani yllättäviä vastoinkäymisiä. Mikäli olisin pysähtynyt korjaamaan, jokaista virhettä, ei peli olisi valmistunut ajoissa. Kuitenkin kohtuullinen määrä optimointia suoritettiin ja loput kirjattiin ylös jatkokehitystä varten.

Projektin asia sisältö oli järisyttävän laaja, näin jälkeenpäin pohdittuna. Pelkästään kenttäsuunnitteluun olisi voinut käyttää koko projektin ajan tai tärkeimpiin työkaluihin, kuten Unity3D:hen ja blenderiin, joihin myös olisin voinut tutustua koko vuoden ajan. Mutta mielestäni opin hyvin, jokaisesta aihealueesta ja nyt tiedän mihin haluan tutustua tarkemmin. Hansoft ja Blender herättivät minussa suurta kiinnostusta tämän työn aikana. Avoimen lähdekoodin sovelluksiin tutustuminen oli erittäin viisasta. Ominaisuudet ovat lähestulkoon samat, kuin maksullisissa sovelluksissa ja sovellukset ovat ilmaisia, sekä niitä tukevat vahvat yhteisöt. Viimeisinä sanoinani haluan kertoa, että haluan ehdottomasti viedä tämän projektin loppuun asti.

# LÄHTEET

Ante J. 2007. Performance Optimization. Osoitteessa http://unity3d.com/support/resources/files/Unite07_Optimization.pdf Syyskuu 2007

Essenthel 2011. Features. Osoitteessa http://www.esenthel.com/?id=features 19.4.2011

Faulkner X. 2000. An Introduction to usability engineering. Iso Britannia: Antony Rowe ltd.

Gamasutra 2010. Unity Technologies and Allegorithmic Announce Technical and Strategic Partnership. Osoitteessa http://www.gamasutra.com/view/pressreleases/63199/Unity_Technologies_and_Allegorithmic_Announce_Technical_andStrategic_Partnership.php 21.9.2010

Gamasutra 2011. Latest EKI One A.I. Middleware Integrates With Unity3D. Osoitteessa http://www.gamasutra.com/view/news/33116/Latest_EKI_One_AI_Middleware_Integrates_With_Unity3D.php 18.2.2011

Hammel, M. 1999. The Artists' Guide to the Gimp The GNU Image Manipulation Program. Yhdysvallat: Specialized Consultants, Inc.

Haukilehto, A. 2003. Visual C# .Net. Helsinki: Edita Prima Oy.

Lammi O. - Malmirae P. 2003. OpenOffice. Jyväskylä:

Myers, E. - Badgett, T. - Thomas, T. - Sandler, C. 2004. The Art of Software Testing Second Edition. Yhdysvallat: John Wiley & Sons, Inc.

OSI 2001. Mission. Osoitteessa http://www.opensource.org/ 8.4.2011

OSI 2001 The Open Source Definition Introduction. Osoitteessa http://www.opensource.org/docs/osd 18.4.2011

Oracle 2011 Oracle hands OpenOffice to open-source community, gives up commercial sales. Osoitteessa http://www.betanews.com/article/Oracle-hands-OpenOffice-to-opensource-community-gives-up-commercial-sales/1303142878 18.4.2011

Peltomäki, J. 2001. JavaScript. Suomi: Tummavuoren kirjapaino.

Profiler 2011. Osoitteessa http://unity3d.com/support/documentation/Manual/Profiler.html 19.1.2011

Schilpp, J. 2001. Lock Mechanism in game design. Osoitteessa http://www.gamedev.net/page/resources/_/reference/103/general-game-design/game-mechanics/lock-mechanisms-in-game-design-r1386 16.6.2001

SIO2-engine 2011. Features. Osoitteessa http://sio2interactive.com/features.php 19.4.2011

Steamreview 2005. Steam's Finances. Osoitteessa http://steamreview.org/posts/finances/ 26.10.2005

Torque3D 2011. Overview. Osoitteessa http://www.garagegames.com/products/torque-3d 19.4.2011

UDK 2011. Features. Osoitteessa http://www.udk.com/features 19.4.2011

Renderöintitilastoikkuna 2010. Rendering Statistics window. Osoitteessa http://unity3d.com/support/documentation/Manual/RenderingStatistics.html 7.9.2010

Ryan D. 2006. Character Animation. Osoitteessa http://wiki.blender.org/index.php/Doc:Tutorials/Animation/BSoD/Character_Animation 26.7.2006

Six Times Nothing 2011. UniSky. Osoitteessa http://www.sixtimesnothing.com/unisky/ 19.4.2011

Unity 2010a. High level networking concepts. Osoitteessa http://unity3d.com/support/documentation/Components/net-HighLevelOverview.html 19.07.2010

Unity 2010b. Modeling optimized Characters. Osoitteessa http://unity3d.com/support/documentation/Manual/Modeling%20Optimized%20Characters.html 24.9.2010

Unity 2011. Overview: Coroutines & Yield. Osoitteessa http://unity3d.com/support/documentation/ScriptReference/index.Coroutines_26_Yield.html 22.4.2011

Unity 2011a. Beast Lightmapping. Osoitteessa
http://unity3d.com/unity/whats-new/unity-3.html 19.4.2011

Unity 2011b. Batching. Osoitteessa http://unity3d.com/unity/engine/rendering
19.4.2011

Unity 2011c. Licenses. Osoitteessa http://unity3d.com/unity/licenses
19.4.2011

Unity 2011d. Unity Assets Server. Osoitteessa
http://unity3d.com/unity/features/unity-asset-server 19.4.2011

Unity 2011e. Union. Osoitteessa http://unity3d.com/union/ 19.4.2011

Unity 2011f. Asset Store. Osoitteessa http://unity3d.com/unity/editor/asset-
store.html 19.4.2011

Wikipedia 2010a**.** Iterointi. Osoitteessa http://fi.wikipedia.org/wiki/Iterointi
21.11.2010

Williamson J. 2010. Character Modeling in Blender. Osoitteessa
http://cg.tutsplus.com/tutorials/blender/
character-modeling-in-blender-basix/ 10.3.2010

# LIITE

## Liite 1. Testauksen kyselylomake

1. Arvioi itsesi pelaajana:
    - □ Noviisi
    - □ Casual
    - □ Pro

2. Sukupuoli
    - □ Mies
    - □ Nainen

3. Ikä
    - □ <18
    - □ 18–25
    - □ 26–35
    - □ 36–45
    - □ 46+

4. Kuinka useasti pelaat pelejä?
    - □ Pelasin kerran jotain.
    - □ Harvoin
    - □ Joskus
    - □ Useasti
    - □ Aina kun mahdollista

5. Valitse pelialustasi.
    - □ PC
    - □ MAC
    - □ Linux
    - □ Web
    - □ Xbox 360
    - □ PS3
    - □ Wii

□ iPhone/iPad/iPod Touch

□ Android

# Liite 1. Testauksen kyselylomake

□ Muu:_____

6. Millä alustalla testasit peliä?
   □ Windows
   □ Mac
   □ Muu:_____
7. Arvioi kiinnostuksesi tähän peliin, ennen kuin testasit peliä
   □ Ei yhtään
   □ Vähän
   □ Jonkin verran
   □ Paljon
   □ Todella paljon
8. Tuntuiko sinusta että peli eteni:
   □ Liian hitaasti
   □ Hitaasti
   □ OK
   □ Nopeasti
   □ Liian nopeasti
9. Mistä pidit pelissä:_____
   _____
10. Mistä et pitänyt pelissä:_____
    _____
11. Kyllästyitkö missään vaiheessa peliä, jos kyllästyit niin missä vaiheessa: _____
12. Aseta ehdotuksia tähän kohtaan:_____
    _____

# Liite 1. Testauksen kyselylomake

13. Arvioi kiinnostuksesi tähän peliin, pelin testaamisen jälkeen
    - □ Ei yhtään
    - □ Vähän
    - □ Jonkin verran
    - □ Paljon
    - □ Todella paljon

14. Pelaisitko tätä peliä uudelleen?
    - □ Kyllä
    - □ En
    - □ Vain jos se parantuisi paljon

15. Paljonko maksaisit tästä pelistä
    - □ 0€ Mielestäni sen tulisi olla ilmainen
    - □ 1–5€
    - □ 5–10€
    - □ 10–15€
    - □ 15–20€
    - □ 20–25€
    - □ Mitä tahansa

16. Paljonko maksaisit yksinpelistä, josta pidäit tai joka vaikuttaa mielenkiintoiselta:_____
17. Paljonko maksaisit moninpelistä, josta pidät tai joka vaikuttaa mielenkiintoiselta:_____
18. Paljonko maksaisit ladattavasta lisäsisällöstä ilmaiseen peliin: _____
19. Paljonko maksaisit ladattavasta lisäsisällöstä ostamaasi peliin: _____
20. Mistä kuulit tästä pelistä:_____