# Unity-pelinkehitysprosessin tehostaminen

Teemu Sinkkonen

Opinnäytetyö
Marraskuu 2014
Tietojenkäsittelyn koulutusohjelma
Pelituotanto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Pelituotanto

SINKKONEN, TEEMU:
Unity-pelinkehitysprosessin tehostaminen

Opinnäytetyö 37 sivua
Marraskuu 2014

---

Työn tarkoituksena oli selvittää Unity-editorin laajennusmahdollisuuksia sekä muita pelinkehitysprosessin tehostamiskeinoja. Eri editorilaajennuksia ja työkaluja kokeiltiin käytännössä toteuttamalla toimintaroolipelin tekninen runko ja pieni testikenttä. Tavoitteena oli auttaa toimeksiantajan henkilöstöä hyödyntämään Unity-ohjelmistoa tarjoamalla raportissa käsitellyistä tekniikoista käytännön esimerkkejä.

Esimerkkiprojekti suunniteltiin niin, että se havainnollisti raportissa käsiteltyjä aiheita. Käytettävät editorilaajennukset ja muut työkalut valittiin kaupallisista Unity-peliprojekteista kertyneiden kokemusten pohjalta. Joitakin editorilaajennuksia toteutettiin myös itse. Raportissa selvitetään eri tekniikoiden käyttötapoja havainnollistavin kuvaesimerkein.

Toimeksiantajan palautteen perusteella projekti oli onnistunut ja sitä tullaan mahdollisesti käyttämään jatkossa uuden peliprojektin pohjana. Yhdessä kirjallisen raportin kanssa projekti tukee yrityksen peliohjelmoijien Unity-tietämystä ja mahdollistaa entistä tehokkaamman pelien tuotantoprosessin.

---

Asiasanat: Unity, ohjelmointi, pelinkehitys, editorilaajennukset.

**ABSTRACT**

Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Development

SINKKONEN, TEEMU:
Enhancing the Unity Game Development Process

Bachelor's thesis 37 pages
November 2014

---

The purpose of this thesis was to explore different avenues for enhancing productivity and quality of end product while developing games using the Unity game development software. A technical prototype for an action role-playing game was made using tried-and-true tools and techniques from previous commercial projects in order to demonstrate the Unity editor's capabilities to the client.

All techniques used in implementing the prototype were carefully considered and properly demonstrated. In addition to commercial Unity plugins and other software, some editor extensions were custom made to highlight the ease of customizing the Unity editor to suit a particular project's needs. This report discusses all the different techniques and their uses, along with example pictures.

Most of the techniques used were found to be useful and worth the time or money required to implement them. According to the client, the project has been useful in strengthening their programmers' knowledge of Unity and the prototype might be used as a basis for a future commercial project. It provides practical examples of how to use some very useful tools that greatly enhance the Unity workflow.

---

Key words: Unity, programming, editor extensions, game development.

**SISÄLLYS**

1 JOHDANTO ..... 6
2 UNITY-PELINKEHITYSOHJELMISTO ..... 7
2.1 Unity yleisesti ..... 7
2.2 Työnkulku Unity-editorissa ..... 8
2.3 Yleiset ongelmat ja niiden vaikutus projekteihin ..... 9
2.3.1 Tehottomuus ..... 10
2.3.2 Työkalujen puutteellisuus ..... 10
2.3.3 Konfliktit versionhallinnassa ..... 10
3 KEHITYSPROSESSIN TEHOSTAMISKEINOT ..... 12
3.1 Tehostamiskeinot esimerkkiprojektissa ..... 14
3.1.1 Ulkoiset työkalut ..... 14
3.1.2 Editorilaajennukset ..... 15
3.1.3 Valmiit koodipaketit ..... 16
4 UNITY -EDITORIN LAAJENTAMINEN ..... 17
4.1 Laajennusten asennus ..... 17
4.2 Omien laajennusten toteutus ..... 17
4.2.1 Assettien käsittelijät ..... 17
4.2.2 Datatyökalut ..... 18
4.2.3 Inspector-elementtien muokkaus ..... 20
4.2.4 Automaattitallennus ..... 21
4.2.5 Nollapisteeseen liikutus ..... 22
4.2.6 GameObjectien hierarkiamuutokset ..... 23
4.2.7 Custom context menu ..... 24
5 PELIRUNGON TOTEUTUS ..... 25
5.1 Hahmojärjestelmä ..... 25
5.2 Pelin kulku ..... 26
5.3 Pelikentän maasto ..... 27
5.4 Käyttöliittymä ..... 28
5.5 Tekstuurit ..... 29
5.6 Visuaaliset tehosteet ..... 32
5.6.1 Shaderit ..... 32
5.6.2 Partikkeliefektit ..... 34
6 POHDINTA ..... 35
LÄHTEET ..... 37

LYHENTEET JA TERMIT

| | |
|---|---|
| Asset | Mikä tahansa pelinkehityksessä käytetty resurssi kuten 3d-malli, tekstitiedosto tai musiikkitiedosto. |
| CSV | Comma-separated values, tiedostomuoto yksinkertaisten taulukoiden tallentamiseen. |
| GameObject | Unity-pelin rakennuspalikka, joka kuvastaa pelimoottoriin asetettua assettia. |
| Partikkeliefekti | Joukko pieniä kaksiulotteisia kuvia, joiden avulla simuloidaan tulta ja muita vaikeasti kuvattavia ilmiöitä. |
| Prefab | Ennalta tehty GameObjectin muotti, josta voidaan tarvittaessa luoda identtisiä kopioita joko editorissa tai ajonaikaisesti. |
| Scene | Unity-editorin käyttämä scene-tiedosto, joka sisältää yhden pelikohtauksen GameObjectit. |
| Shader | Peliobjektien piirtämistä ohjaava pieni tietokoneohjelma. |
| Skripti | Yleensä GameObjectiin liitettävä komponentti, joka sisältää ohjelmakoodia. |

# 1 JOHDANTO

Unity Technologiesin kehittämästä Unity-pelinkehitysohjelmistosta on viime vuosina tullut erittäin suosittu kehitysalusta etenkin mobiilisovelluksille. Moottorin vahvuuksista puhuttaessa nostetaan yleensä esiin kehitysprosessin visuaalisuus ja helppous, sovellusten vaivaton kääntäminen eri kohdealustoille sekä suuren käyttäjäkunnan tarjoama vertaistuki. Unityn perusversio on myös kehittäjälle täysin ilmainen, mikä tekee siitä helpon vaihtoehdon etenkin harrastajille sekä pk-yrityksille.

Työn toimeksiantaja Rimeforge Entertainment on hiljattain perustettu tamperelainen peli- ja ohjelmistotalo, joka käyttää peliprojekteissaan Unity-moottoria. Yrityksen ohjelmoijilla on vankka tausta ohjelmistotuotannosta, mutta Unity ja sen tarjoamat mahdollisuudet eivät vielä ole täysin tuttuja. Minulla on runsaasti kokemusta Unityn käytöstä erilaisissa projekteissa ja halusin tutkia tarkemmin ohjelmiston soveltuvuutta yrityskäyttöön, joten päätin tehdä aiheesta opinnäytetyön.

Tavoitteena työllä on tehostaa toimeksiantajan pelinkehitysprosessia ja parantaa tuotteiden laatua selvittämällä Unity-editorin tarjoamia mahdollisuuksia. Tämän raportin lisäksi työhön kuuluu Unity-projekti, jossa toteutettiin yksinkertainen pelirunko hyödyntäen ja havainnollistaen raportissa mainittuja menetelmiä. Projektia tai sen osia voidaan hyödyntää sellaisenaan uusien pelien pohjana, tai sitä voidaan käyttää ainoastaan esimerkkinä. Näin toimeksiantaja saa uuden tiedon lisäksi myös konkreettista hyötyä vähintään koodiesimerkkien muodossa.

Raportissa perehdytään Unity-editorin toimintoihin sekä toimintojen laajentamismahdollisuuksiin, minkä lisäksi käydään läpi joitain Unityn ulkopuolisia työkaluja. Aluksi luodaan yleinen katsaus Unityyn ja tarjolla oleviin kehitysprosessin tehostamiskeinoihin, minkä jälkeen esitellään tarkemmin esimerkkiprojektissa käytetyt työkalut ja laajennukset. Toteutusosissa havainnollistetaan kaikkia näitä tekniikoita selvittämällä niiden käyttöä esimerkkiprojektissa.

Aiheesta on tarjolla keskitetysti vain vähän tietoa, eikä siitä ole esimerkiksi kirjoitettu kirjoja, joten raportissa on pyritty myös kokoamaan tätä tietoa Unity-pelinkehityksestä kiinnostuneiden saataville. Kirjallisten lähteiden lisäksi, ja niiden tukena, on hyödynnetty paljon työelämästä ja peliprojekteista kertynyttä käytännön tietoa.

# 2 UNITY-PELINKEHITYSOHJELMISTO

## 2.1 Unity yleisesti

Unity mielletään yleisesti yksinomaan pelimoottoriksi, mutta tarkalleen ottaen kyseessä on ohjelmistopaketti, joka koostuu kolmesta eri komponentista: pelimoottorista, editorista sekä julkaisumoduuleista. Näistä kolmesta osasta moottori on keskeisessä asemassa, mutta kehittäjän näkökulmasta muut osat ovat vähintään yhtä tärkeitä. (Thorn 2013.)

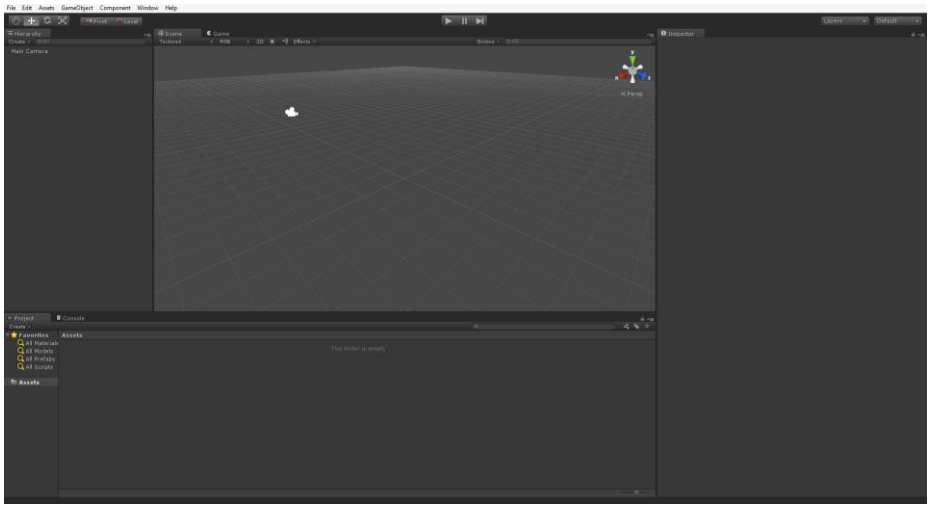
Pelimoottori itsessään koostuu erilaisista hallintakomponenteista (*managers*), jotka määrittävät pelimaailman toimintaa. Nämä komponentit muun muassa mallintavat painovoimaa ja muuta fysiikkaa, huolehtivat käyttäjän syötteiden tulkitsemisesta ja käyttöliittymän piirtämisestä sekä hallitsevat pelin äänimaailmaa. Hallintakomponenttien lisäksi moottorista löytyy GameObjecteja, jotka kuvaavat peliympäristöön asetettuja assetteja. Koko Unity-pelinkehitys pohjautuu nimenomaan näiden GameObjectien asetteluun ja manipulointiin. (De Byl 2012.)

Unityn editori on suunniteltu visuaaliseksi ja helppokäyttöiseksi niin, että käyttäjä voi periaatteessa tehdä mitä tahansa hiirellä raahaamalla. Näin pystytään muun muassa tuomaan projektiin uusia assetteja, yhdistämään pelimaailman objekteihin koodia ja jopa muokkaamaan koodin muuttujia. Käyttäjän on myös helppo muokata editorin ulkoasua ja ikkunoiden asettelua mieleisekseen. (Menard & Wagstaff 2014.) Editori on suunniteltu erittäin helposti laajennettavaksi, mihin myös tämä opinnäytetyö keskittyy.

Alustariippumattomuus on Unityn tärkeä myyntivaltti. Periaatteessa tämä tarkoittaa, että kehittäjä voi yhden Unity-projektin pohjalta julkaista pelinsä mille tahansa alustalle. Käytännössä asia ei ole aivan näin yksinkertainen, mutta Unity sisältää silti tarvittavat moduulit peliprojektin julkaisemiseksi kaikille tuetuille alustoille. Tuettuja alustoja ovat muun muassa Windows, Mac, Linux, web-selaimet, iOs, Android, Windows Phone 8, Xbox 360 ja PS3. Nämä moduulit yhdistävät Unity-pelimoottorin editorissa tuotettuun pelidataan niin, että tuloksena on valitulla alustalla itsenäisesti toimiva sovellus. (Thorn 2013.)

## 2.2 Työnkulku Unity-editorissa

Unityllä tehdyt pelit ja sovellukset koostuvat sceneistä, jotka sisältävät GameObjecteja. Näihin objekteihin puolestaan liitetään erilaisia komponentteja, joita on yhteensä 7 eri kategoriaa: objektin piirtämiseen käytetty Mesh, partikkeliefektit, fysiikkakomponentit, äänet, renderointikomponentit erikoistehosteiden piirtämiseen, skriptit eli pelikoodi ja sekalaiset komponentit, jotka eivät sovi mihinkään muuhun kategoriaan. (De Byl 2012.) Kuvassa 1 esitellään Unity-editorin perusnäkymä.

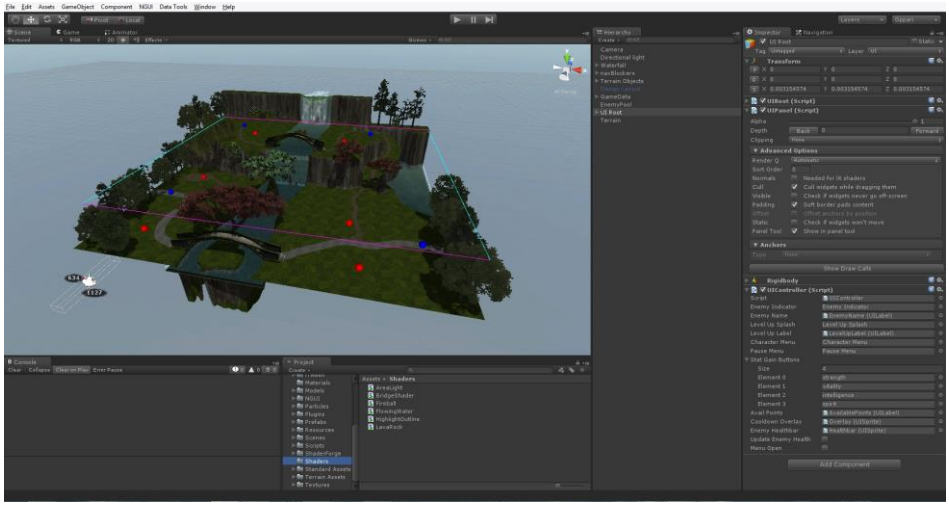


KUVA 1. Unity-editorin perusnäkymä (Unity 4.5.3f3 2014, kuvankaappaus)

Uuden projektin aloitettuaan on kehittäjällä edessään tyhjä scene, joka sisältää vain Main Camera -nimisen objektin. Tämä on kameraobjekti, jonka kautta pelejä oletusarvoisesti kuvataan. Ruudun ylälaidasta löytyvistä valikoista voidaan lisätä sceneen uusia GameObjecteja tai komponentteja jo olemassa oleviin objekteihin. Projektin asetuksista voidaan muokata esimerkiksi fysiikkamallinnuksen laskentanopeutta ja painovoiman voimakkuutta ja suuntaa.

Oletuksena näkyvissä on neljä ikkunaa. Scene-ikkunasta nähdään työstettävän scenen sisältö, Hierarchy-ikkunasta scenessä olevien GameObjectien keskinäinen hierarkia, Project-ikkunasta kaikki projektikansiosta löytyvät assetit ja Inspector-ikkunasta tarkemmat tiedot kulloinkin valittuna olevasta GameObjectista. Näiden taakse piilotettuina ovat Game-ikkuna, josta nähdään, miltä peli milloinkin näyttää pääkameran läpi katsottuna, sekä konsoli, johon tulostetaan erilaiset varoitukset, virheilmoitukset ja koodin gene-

roimat debug-viestit. Koko editorinäkymä on käyttäjän muokattavissa mieleisekseen, ainoastaan ruudun yläreunasta löytyviä valikoita ja työkalupalkkia ei voi siirtää. Kuvassa 2 nähdään esimerkkiprojektia varten muokattu editorinäkymä.



KUVA 2. Muokattu editorinäkymä (Unity 4.5.3f3 2014, kuvankaappaus)

Työnjako Unity-projekteissa järjestetään yleisesti niin, että jokaisella pelisisältöä tuottavalla työntekijällä on oma scenensä. Sceneissä tehdyt muutokset pyritään kokoamaan prefab-paketteihin, joista yleensä ohjelmoija kokoaa lopulliseen peliin päätyvän scenen. Ohjelmoijat voivat esimerkiksi luoda peliin täysin toimivan taistelusysteemin pelkästään laatikoista kootussa tilapäisnäkymässä, minkä jälkeen kaikki skriptit yhdistetään animaattorien valmistelemiin hahmo-objekteihin, jotka vielä istutetaan kenttäsuunnittelijan erikseen tekemään pelimaailmaan.

## 2.3 Yleiset ongelmat ja niiden vaikutus projekteihin

Unity-editorissa on joitakin puutteita ja epäloogisuuksia, joihin törmää niin opiskelijaprojekteissa kuin pelitalojen tuotannoissa. Osaan näistä ongelmista Unity Technologies pyrkii reagoimaan lisäämällä uusia toiminnallisuuksia. Osa taas on niin perustavanlaatuisia, että on keksittävä muita ratkaisuja.

### 2.3.1 Tehottomuus

Unityssä työskentelyyn liittyy välillä paljon turhaa mekaanista toistoa, joka hidastaa työskentelyä. Esimerkiksi kaikki assetit tuodaan aina peliprojektiin samoilla, tiedostotyypin mukaan vaihtelevilla asetuksilla. Jos kehittäjä haluaa muuttaa näitä asetuksia esimerkiksi niin, että 3d-mallien mukana ei tuoda animaatiodataa, pitää nämä muutokset muistaa tehdä manuaalisesti aina uusia 3d-malleja tuotaessa.

### 2.3.2 Työkalujen puutteellisuus

Erityisesti Unityn oma UI-järjestelmä on hyvin alkeellinen ja työläs käyttää. Käyttöliittymä pitää luoda käytännössä kokonaan koodirivi kerrallaan ja sen jokainen komponentti piirretään uudelleen joka ruudunpäivityksen yhteydessä (Bernardoff 2014). Useiden UI-elementtien uudelleen piirtäminen jopa 60 kertaa sekunnissa on hyvin raskasta ja kuormittaa näytönohjaimen lisäksi myös sovelluksen muistinhallintaa roskankeruun muodossa.

Myös käyttäjän syötteiden hallinta on Unityssä oletuksena melko vaatimattomalla tasolla. Projektin asetuksista voi vaihtaa joitakin valmiiksi määriteltyjä geneerisiä toimintapainikkeita, joita myös pelaaja pääsee itse vaihtamaan Unity-sovellusta käynnistäessään. Näitä ohjausasetuksia ei kuitenkaan ole mahdollista muuttaa ajonaikaisesti, mikä on etenkin moderneissa tietokonepeleissä itsestäänselvyys.

### 2.3.3 Konfliktit versionhallinnassa

Versionhallinta on yksi tärkeimmistä osa-alueista missä tahansa ohjelmistoprojektissa. Kaikki versionhallintaohjelmistot toimivat samalla perusperiaatteella niin, että projektitiedostoista on aina tallessa nykyisten tiedostojen lisäksi myös kaikki edelliset versiot. Näin voidaan aina palata edelliseen toimivaan versioon, mikäli kehityksessä tulee ylitsepääsemättömiä ongelmia. Mikäli useampi käyttäjä yrittää ladata versionhallintaan samaan tiedostoon yhtä aikaa tehtyjä muutoksia, yhdistää ohjelmisto joko tiedostot automaattisesti tai varoittaa käyttäjää konfliktista ja antaa valita halutut muutokset. (Loeliger & McCullough 2012.)

Unityn scene- ja prefab-tiedostojen yhdistäminen toisiinsa ei yleensä onnistu edes manuaalisesti, joten jos kaksi tiimin jäsentä on tehnyt muutoksia samaan tiedostoon, pitää toisen tekemät muutokset hylätä. Muutoin riskinä on pahimmillaan koko projektin rikkominen ja tunteja kestävä selvitystyö, kun yritetään palauttaa kunkin työntekijän työpisteet edellisiin versioihin ja korjata versionhallintaa niin, ettei ongelma pääse leviämään. Käytännössä ainoa tapa välttää näitä ongelmia Unity-projektissa on olla tekemättä muutoksia samoihin tiedostoihin.

# 3 KEHITYSPROSESSIN TEHOSTAMISKEINOT

## Valmiit koodipaketit

Varsinaisten editorilaajennusten ohella Unityyn on saatavilla erilaisia valmiiksi laadituista apufunktioista koostuvia paketteja, jotka antavat kehittäjän keskittyä enemmän korkean tason pelimekaniikan toteutukseen kuin esimerkiksi GameObjectin tehokkaaseen liikutusfunktioon. Paketit on yleensä rajattu aihealueen mukaan esimerkiksi kappaleen lentoradan simulointiin tai pelihahmojen reitinhakuun.

## Editorilaajennukset

Unity-editori on rakennettu niin, että käyttäjät voivat laajentaa sen toiminnallisuuksia lähes rajattomasti haluamillaan tavoilla (Jakson 2014). Editorilaajennukset ovat Unity-editoriin liitettäviä lisäosia, jotka tuovat käyttöliittymään uusia toimintoja. Näin käyttäjä voi muokata työkalunsa aina tekeillä olevaan projektiin sopiviksi. Laajennukset voivat olla miten kattavia tahansa, aina yhtä tehtävää nopeuttavista yksittäisistä koodiriveistä laajoihin, työnkulkua merkittävästi muokkaaviin työkalupaketteihin.

## Maksulliset laajennukset

Maksulliset laajennukset ovat yleensä Unity Asset Storen kautta ostettavia, kolmannen osapuolen tekemiä paketteja. Tyypillisesti tällaiset laajennukset ovat joko hyvin laajoja tai muuten haastavia toteuttaa. Hintataso vaihtelee muutamasta eurosta aina useisiin satoihin euroihin, mutta isompikin hinta on yleensä perusteltavissa säästetyillä työtunneilla.

## Ilmaiset laajennukset

Suuri osa ilmaisista laajennuksista on maksullisten laajennusten esittelyversioita, joista puuttuu suurin osa täyden version ominaisuuksista. Toisaalta saatavilla on myös paljon hyödyllisiä työkaluja ja varteenotettavia vaihtoehtoja maksullisille laajennuksille.

### Itse tehtävät laajennukset

Kuka tahansa voi laajentaa Unity-editoria, joten oman projektin tarpeisiin räätälöityjen ratkaisujen toteutus on täysin mahdollista. Editorin laajentamisesta on tarjolla huomattavasti vähemmän tietoa kuin pelinkehityksestä, mutta pienellä vaivannäöllä pääsee nopeasti alkuun. Isommankin laajennuksen tekeminen itse voi olla hyvä ratkaisu, mikäli kehittäjällä riittää mielenkiintoa ja projektin aikataulu sen sallii. Parhaassa tapauksessa omasta työkalusta hioutuu itsessään myytävä tuote.

### Ulkoiset työkalut

Varsinaisen Unity-editorin lisäksi onnistunut ja tehokas peliprojekti vaatii myös muita työkaluja. Tämän työn puitteissa ei ole tarkoitus perehtyä näiden työkalujen syvällisempään käyttöön, vaan lähinnä mainita muutamia hyväksi havaittuja esimerkkejä.

### Toimistosovellukset

Erityisesti taulukkolaskentaohjelmista voi olla suurta hyötyä lähes minkä tahansa pelin toteutuksessa erityisesti pelisuunnittelijan näkökulmasta. Sopivalla laskentataulukolla voidaan simuloida pelimekaniikkaa, optimoida monetisaatiota sekä pitää kirjaa peliin suunnitelluista ominaisuuksista ja niiden keskinäisistä suhteista. Laskentataulukoita voidaan myös tuoda Unityyn CSV-muodossa tai jopa lukea suoraan pilvestä.

### Assettien tuottaminen

Markkinoilla on saatavilla jonkin verran esimerkiksi tekstuurien tekemiseen keskittyviä ohjelmistoja, jotka on tarkoitettu nimenomaan käytettäväksi yhdessä Unityn kanssa. Näistä ohjelmista on yleensä tarjolla sekä itsenäinen, mahdollisesti rajoitettu versio sekä Unityyn integroitu täydellinen versio, joka nopeuttaa työskentelyä entisestään.

## 3.1 Tehostamiskeinot esimerkkiprojektissa

Esimerkkiprojektina päätettiin toteuttaa toimintaroolipelin runko, jossa käytetään kattavasti yllä kuvattuja työskentelyn tehostamiskeinoja. Sisällön toteuttaminen rungon päälle jäi tässä projektissa toissijaiseksi, pääpaino oli teknisessä toteutuksessa. Projektiin toteutettavat ominaisuudet pyrittiin valitsemaan niin, että ne havainnollistavat raportissa käsiteltyjä asioita mahdollisimman hyvin. Käytettäväksi valittiin aiempien kokemusten pohjalta hyväksi havaittuja työkaluja ja laajennuksia. Kunkin työkalun ja laajennuksen käyttöä käsitellään tarkemmin luvussa 5.

### 3.1.1 Ulkoiset työkalut

**Google Drive** tarjoaa ilmaisen taulukkolaskentaohjelman, jonka ehdottomana vahvuutena on työn tallentuminen automaattisesti pilveen sekä työkirjojen helppo jakaminen kaikille asianosaisille. Useampikin käyttäjä voi muokata samaa tiedostoa yhtä aikaa, mikä on tiimissä työskenneltäessä erittäin hyödyllistä. Esimerkkiprojektissa taulukkolaskentaa käytettiin määrittämään pelihahmojen ominaisuuksia niin, että CSV-muotoon tallennettua taulukkoa voitiin hyödyntää suoraan Unityn puolella. Näin esimerkiksi hahmojen tasapainotuksesta vastaava suunnittelija voi tehdä omaa työtään välittämättä pelin teknisestä toteutuksesta.

Allegorithmic-yrityksen kehittämä **Bitmap2Material** on työkalu, jolla mistä tahansa bittikarttakuvasta voidaan generoida nopeasti ja helposti korkealaatuinen materiaalitiedosto. Ohjelma soveltuu erityisen hyvin toistuvien tekstuureiden, kuten ruohon tai kiviseinän, tekemiseen. Itsenäisesti toimiva perusversio tuottaa annetun bittikartan ja asetusten perusteella valitut tekstuurit, kuten normaalikartan ja värit, erillisinä kuvatiedostoina, jotka pitää käsin yhdistää Unity-editorissa. Tämän lisäksi saatavilla on myös Unityyn integroitu kalliimpi versio, joka tuottaa suoraan reaaliaikaisesti editorin kautta muokattavia materiaalitiedostoja.

### 3.1.2 Editorilaajennukset

**NGUI** (Next-Gen UI kit) on suosituin ja monipuolisin ratkaisu Unity-pelin käyttöliittymän tekemiseen. NGUI:n kehittäjä palkattiin tekemään Unityn uutta virallista UI-systeemiä, joka on tällä hetkellä beta-vaiheessa.

**Shader Forge** on solmupohjainen (*node-based*) visuaalinen shader-editori. Shaderit ovat tietokoneohjelmia, jotka ohjaavat näytönohjainta piirtämään ruudulle halutut asiat. Unityssä näitä ohjelmia kirjoitetaan erillisellä ShaderLab-ohjelmointikielellä (Unity Technologies 2014a). Shadereilla voidaan normaalin piirtämisen lisäksi saada aikaan erilaisia erikoistehosteita jotka lisäävät pelien näyttävyyttä, mutta niiden ohjelmointi käsin on haastavaa ja vaatii syvällistä perehtymistä. Shader Forge auttaa tässä tarjoamalla visuaalisen käyttöliittymän, jossa kuka tahansa voi rakentaa monimutkaisia, kehittynyttä grafiikkateknologiaa hyödyntäviä shadereita.

Käyttäjän syötteiden hallintaan tarkoitettu **cInput** korvaa Unityn oman syötejärjestelmän ja tarjoaa monipuolisemmat mahdollisuudet esimerkiksi ajon aikana vaihtuvien näppäinoikoteiden toteutukseen. Lisäksi laajennukseen on toteutettu parempi tuki muun muassa rattiohjaimille. Kuvassa 3 on esimerkkejä cInputin käytöstä.

```
void Update () {
    if (cInput.GetKey("Attack"))
    {
        if (cInput.GetKey("Hold"))
        {
            Stop();
        }
```

```
foreach (string key in keybindStrings)
{
    if (!cInput.IsKeyDefined(key))
    {
        cInput.SetKey(key, defKeys[key]);
    }
}
```

```
public void ChangeKeybind()
{
    cInput.ChangeKey(transform.parent.name);
}
```

KUVA 3. Käyttäjän syötteiden kuuntelu, oletusnäppäinkomentojen tarkistus sekä näppäinkomennon muuttaminen cInputilla (Visual Studio Ultimate 2013, kuvankaappaus)

**Hayate** on partikkeliefektien ohjaamiseen tehty laajennus, joka toimii Unityn oman Shuriken-partikkelijärjestelmän ohessa, ja mahdollistaa todella monimutkaisetkin partikkeliefektit. Hayaten avulla yksittäiset partikkelit voidaan esimerkiksi saada helposti seuraamaan tiettyä pistettä, kimpoilemaan seinistä tai reagoimaan ääniaaltoihin.

Yllä mainittujen valmiiden laajennusten lisäksi suunniteltiin toteutettavaksi joukko omia laajennuksia, joista kerrotaan tarkemmin luvussa 4.

### 3.1.3 Valmiit koodipaketit

Bob Berkebilen kehittämä **iTween** on erittäin yksinkertainen ja tehokas animaatiojärjestelmä. Paketista löytyy apufunktiot lähes mihin tahansa GameObjectien liikutteluun liittyvään tilanteeseen, yksinkertaisesta pyörittämisestä monimutkaisiin kamera-ajoihin ja räjähdysvoiman mallinnuksiin.

**CSVReader** on nimensä mukaisesti CSV-tiedostojen lukemiseen tarkoitettu paketti, jota jaetaan avoimesti Unify Wikin kautta. Paketista löytyy funktiot sekä erillisten CSV-muotoisten rivien että kokonaisten CSV-taulukoiden erotteluun string-tyyppisiä muuttujia sisältäviksi taulukoiksi.

# 4 UNITY -EDITORIN LAAJENTAMINEN

## 4.1 Laajennusten asennus

Valmiit editorilaajennukset sekä ainakin suosituimmat koodipaketit löytää helpoimmin Unity Asset Storesta. Asset Storeen pääsee sekä selaimen kautta verkosta että suoraan Unity-editorin sisältä, jolloin haluamansa laajennukset ja muut assetit saa myös tuotua suoraan omaan peliprojektiin. Kuvassa 4 nähdään Import-näkymä, jonka kautta laajennukset tuodaan.


KUVA 4. Laajennuksen tuominen Unity-projektiin (Unity 4.5.3f3 2014, kuvankaappaus)

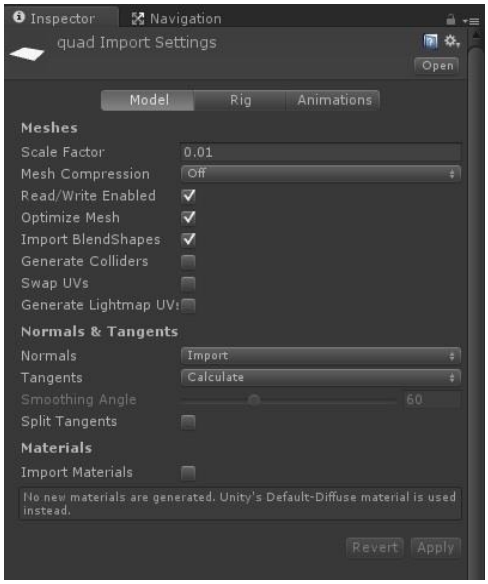
## 4.2 Omien laajennusten toteutus

Omat editorilaajennukset toteutetaan periaatteessa samalla tavalla kuin pelien ohjelmointi Unityllä. Ainoana erotuksena editorissa ajettavat kooditiedostot täytyy sijoittaa projektikansiossa omaan Editor-kansioonsa.

### 4.2.1 Assettien käsittelijät

Assettien käsittelijöitä (*asset postprocessors*) voidaan käyttää säästämään aikaa tuotaessa uusia assetteja projektiin. Erityyppisillä asseteilla on erilaisia ominaisuuksia, esimerkiksi 3d-mallin kohdalla voidaan päättää, tuodaanko mallin mukana myös sen materiaali ja mahdolliset animaatiot. Oletuksena Unity pyrkii tuomaan nämä kaikki ominaisuudet,

vaikka niitä ei alkuperäisellä mallilla edes olisi, jolloin projektikansioon syntyy paljon turhia materiaaleja ja muuta täytettä. Kuvassa 5 nähdään 3d-mallin tuomisen perusasetukset.




KUVA 5. 3d-mallin tuontiasetuksia (Unity 4.5.3f3 2014, kuvankaappaus)

Esimerkkiprojektiin toteutettiin käsittelijät 3d-mallien ja tekstuurien tuomiseen niin, että 3d-mallien materiaaleja ei oletusarvoisesti tuoda projektiin, ja oikein nimetyt tekstuuritiedostot tunnistetaan automaattisesti normaalikartoiksi. Kuvassa 6 esitellään 3d-mallien käsittelijän toteutus.

```
0 references
public class AssetProcessors : AssetPostprocessor
{
    0 references
    void OnPreProcessModel()
    {
        ModelImporter modelImporter = (ModelImporter)assetImporter;
        modelImporter.importMaterials = false;
    }

    0 references
    void OnPreProcessTexture()
    {
```

KUVA 6. Assettien käsittelijän toteutus (Visual Studio Ultimate 2013, kuvankaappaus)

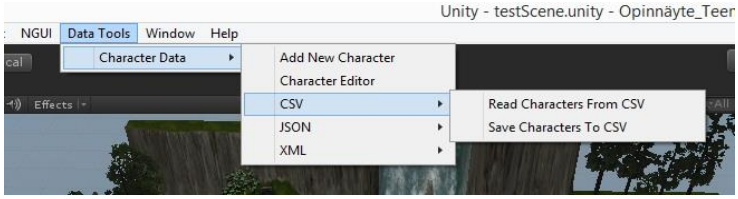
### 4.2.2 Datatyökalut

Yksi tärkeimpiä elementtejä missä tahansa roolipelissä ovat pelin hahmot ja näiden ominaisuudet. Ideaalitilanteessa pelitasapainon muuttelu ei vaadi muutoksia pelin tekniseen toteutukseen Unityssä. Tällä vältetään etenkin versionhallinnan konflikteja, ja toisaalta

suunnittelijan ei myöskään tarvitse välttämättä tuntea pelin koodia testatakseen omia muutoksiaan. Esimerkkiprojektiin toteutettiin järjestelmä, jolla Google Driven taulukkolaskennassa tuotettua ja CSV-muotoon tallennettua hahmodataa voi helposti tuoda Unity-projektiin. Kuvassa 7 nähdään Google Drivesta tuodun CSV-tiedoston sisältö.


KUVA 7. CSV-muotoista hahmodataa (Unity 4.5.3f3 2014, kuvankaappaus)

Työkalun toteutukseen kuuluvat erillisinä osina editoriin lisätty valikkorakenne ja CSV-tiedoston avaaminen sekä taustalla toimiva datankäsittelijäluokka, joka sisältää erilaisia funktioita tiedostosta luetun datan käsittelemiseen. Editoriin toteutettu valikkorakenne nähdään kuvassa 8.




KUVA 8. Editoriin lisätty valikko (Unity 4.5.3f3 2014, kuvankaappaus)

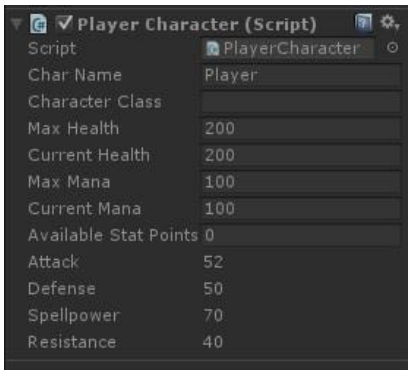
Kun käyttäjä valitsee valikosta haluavansa lukea hahmodataa CSV-tiedostosta, aukeaa valintaikkuna, josta valitaan haluttu tiedosto. Tämän jälkeen valittu tiedosto luetaan ja sen sisältö lähetetään CSV-lukufunktiolle. Funktio koostaa tiedoston sisällöstä listan hahmoista ja näiden ominaisuuksista ja palauttaa sen valikkoluokalle, jossa lista syötetään hahmodatan säilyttämiseen luodun CharacterData-luokan oliolle. Tämä olio puolestaan tallennetaan kiintolevylle Unityn omassa asset-tiedostomuodossa, jolloin se voidaan ladata käytettäväksi koska tahansa ajon aikana. Kuvassa 9 on esitetty osa funktiosta, joka muuttaa CSV-tiedoston sisällön käyttökelpoiseen muotoon.

```
[MenuItem("Data Tools/Character Data/CSV/Read Characters From CSV")]
0 references
public static void ReadCharactersFromCSV()
{
    CharacterData data = ScriptableObject.CreateInstance<CharacterData>();

    AssetDatabase.CreateAsset(data, "Assets/Resources/Data/CharacterData.asset");

    string path = UnityEditor.EditorUtility.OpenFilePanel("Select CSV file", Application.dataPath + "/Character Data/", "csv");
    path = path.Replace(Application.dataPath, "Assets");

    TextAsset asset = (TextAsset)UnityEditor.AssetDatabase.LoadAssetAtPath(path, typeof(TextAsset));

    data.characters = DataUtils.ReadCharactersFromCSV(asset);
```

KUVA 9. Hahmodatan lukeminen ja tallennus (Visual Studio Ultimate 2013, kuvankaappaus)

### 4.2.3 Inspector-elementtien muokkaus

Inspector-elementit näyttävät Unity-editorin samannimisessä ikkunassa valittuna olevaan GameObjectiin liitetyt komponentit ja niiden muuttujat. Oletuksena näkyvillä ovat kaikki public-näkyvyystason muuttujat eli muuttujat, jotka näkyvät luokan ulkopuolelle ja ovat kaikkien käytettävissä. Näin kehittäjä voi peliä testatessaan seurata ja muuttaa eri muuttujien arvoja, mistä on hyötyä etenkin ongelmatilanteissa ja pelitasapainon hienosäädössä. Public-muuttujien käyttö ei kuitenkaan kuulu hyvään olio-ohjelmointitapaan ja kehittäjä voi esimerkiksi haluta seurata monista eri muuttujista koostettuja arvoja. Kuvassa 10 on esimerkki laajennetusta Inspector-elementistä.




KUVA 10. Laajennettu Inspector-elementti näyttää hahmon perusominaisuuksista johdetut taisteluominaisuudet. (Unity 4.5.3f3 2014, kuvankaappaus)

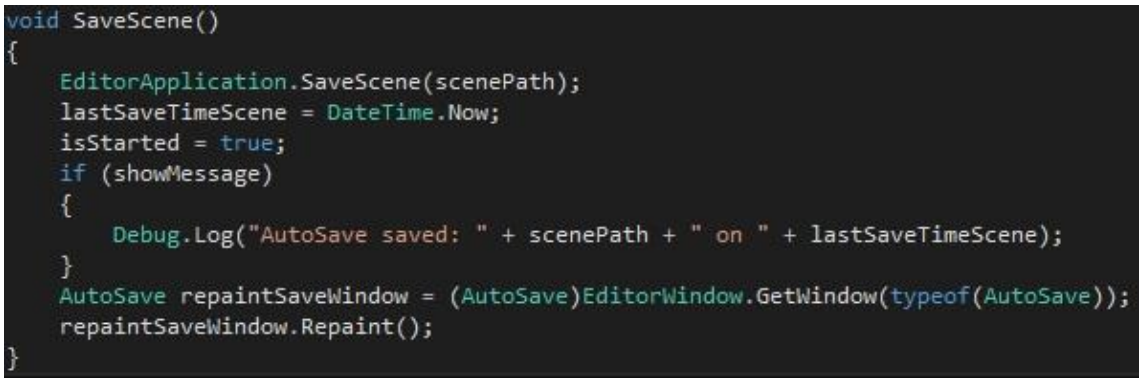
Esimerkkiprojektissa pelihahmoilla on perusominaisuuksina muun muassa fyysinen voima (*strength*), elinvoima (*vitality*), älykkyys (*intelligence*) ja sielu (*spirit*), joista johdetaan taisteluominaisuudet hyökkäysvoima (*attack*), puolustus (*defense*), taikavoima (*spellpower*) ja vastustuskyky (*resistance*). Näiden johdettujen ominaisuuksien helpon seuraamisen mahdollistaa mukautettu Inspector-ikkuna, joka näyttää PlayerCharacter-

luokan komponenteista oletuksena näkyvien public-muuttujien lisäksi myös muita haluttuja ominaisuuksia. Myös muokattavien kenttien tekeminen on mahdollista, jolloin esimerkiksi hyökkäysvoiman laskentakaavaa voisi muuttaa ajon aikana. Kuvassa 11 nähdään osa Inspector-laajennuksen toteutuksesta.


KUVA 11. Oman Inspector-elementin toteutusta (Visual Studio Ultimate 2013, kuvankaappaus)

### 4.2.4 Automaattitallennus

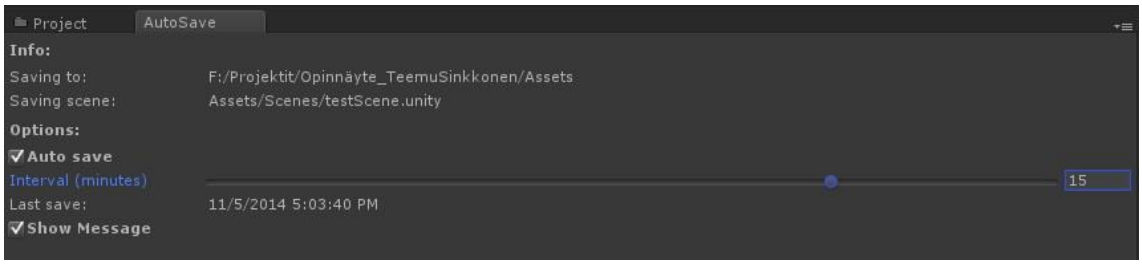
Unity-editoriin ei ole oletuksena rakennettu minkäänlaista automaattitallennusta, mikä saattaa pahimmillaan johtaa monen tunnin työn hukkaamiseen yhden koodivirheen vuoksi, kun editori kaatuu peliä ajaessa. Yleensä näin ei toki käy, mutta automaattitallennuksen toteuttaminen on silti hyvä varokeino. Kuvassa 12 on esitettynä funktio, joka huolehtii avattuna olevan scenen tallentamisesta.




KUVA 12. Automaattitallennuksen tallennusfunktio (Visual Studio Ultimate 2013, kuvankaappaus)

Automaattitallennusta varten tehdään oma editori-ikkuna, jonka kautta voidaan säätää tallennusväliä sekä sitä, onko automaattitallennus käytössä. Kun tämän ikkunan avaa ja liit-

tää omaan editorinäkymäänsä tallennetaan kulloinkin avoinna oleva scene-tiedosto automaattisesti annetun ajan välein. Kuvassa 13 nähdään valmis automaattitallennuksen editori-ikkuna.



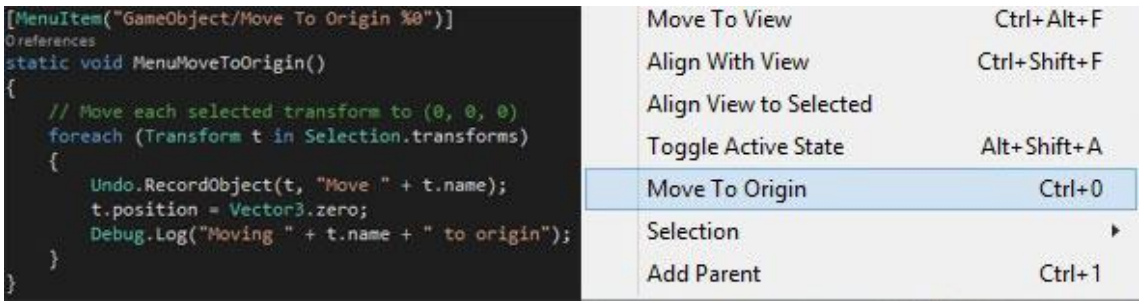

KUVA 13. Automaattitallennuksen editori-ikkuna (Unity 4.5.3f3 2014, kuvankaappaus)

### 4.2.5 Nollapisteeseen liikutus

Jokaisen pelimaailmassa olevan GameObjectin sijaintia ja asentoa säätelee Transform-komponentti (Unity Technologies 2014b). Transform-komponentin position-muuttuja määrittää objektin sijainnin riippuen siitä, missä objekti sijaitsee kulloinkin aktiivisena olevan scenen hierarkiassa. Mikäli objekti on hierarkiassa ylimmällä tasolla, eli sillä ei ole vanhempaa määritellään sen sijainti pelimaailman koordinaatistossa. Mikäli objektilla on vanhempi, määritellään sen sijainti vanhemman sijainnin mukaan.

Objektien hierarkiaa muuttaessa käy usein niin, että eri puolilla sceneä sijaitsevat objektit ovatkin yhtäkkiä liitettyinä toisiinsa ja esimerkiksi liikkuvat yhdessä. Näissä tapauksissa hierarkiassa alempi objekti halutaan yleensä siirtää joko vanhemman kanssa samaan pisteeseen tai ainakin lähelle sitä. Tämä vaatii normaalisti siirrettävän objektin valitsemisen sekä position-muuttujan x-, y- ja z-arvojen nollaamisen manuaalisesti. Unity-editori mahdollistaa tämänkaltaisen mekaanisen toiston vähentämisen toteuttamalla oma editoritoiminto, jolle voi määrittää myös näppäinyhdistelmän.

Kuvassa 14 näkyvä yksinkertainen funktio tekee editorista valmiina löytyvän GameObject-valikon alle uuden Move To Origin -toiminnon, jolle asetetaan näppäinyhdistelmä Ctrl+0. Tämä toiminto siirtää kaikki valittuna olevat GameObjectit hierarkiasta riippuen joko pelimaailman nollapisteeseen tai vanhemman keskipisteeseen. Siirron voi myös perua editorin Undo-toiminnolla.




KUVA 14. Objektien liikutusfunktion toteutus ja näppäinoikotie GameObject-valikossa (Visual Studio Ultimate 2013; Unity 4.5.3f3 2014, kuvankaappaus)

### 4.2.6 GameObjectien hierarkiamuutokset

GameObjectit on Unityn sceneissä järjestetty hierarkiaan, jossa toiset objektit voivat olla toisten alla vanhempi-lapsi-suhteessa. Lapsina olevien objektien sijainti pelimaailmassa määritetään vanhempien sijainnin perusteella, ja mikäli vanhempi asetetaan ei-aktiiviseksi, häviävät myös kaikki lapset. Vanhempina olevilla objekteilla ei usein ole mitään ylimääräisiä komponentteja, vaan niitä käytetään ainoastaan pitämään objektien hierarkia siistinä niputtamalla useita samankaltaisia objekteja yhden säiliöobjektin alle. Objektien hierarkiaa voi muutella helposti editorin hierarkianäkymässä vetämällä objekteja toistensa alle tai päälle, mutta etenkin käsiteltäessä kerralla suurta määrää objekteja tämä voi olla melko työlästä. Kuvassa 15 nähdään Add Parent -toiminnon näppäinoikotien toteutus sekä osa funktiosta, joka luo uuden objektin ja siirtää valitut objektit sen lapsiksi.

```
[MenuItem("GameObject/Add Parent %1")]
0 references
static void MenuInsertParent()
{
    Transform[] transforms = Selection.GetTransforms(SelectionMode.TopLevel |
        SelectionMode.OnlyUserModifiable);

    GameObject newParent = new GameObject("_Parent");
    Transform newParentTransform = newParent.transform;
```

KUVA 15. Osa AddParent-luokan toteutusta (Visual Studio Ultimate 2013, kuvankaappaus)

Esimerkkiprojektiin toteutettiin Ctrl+1-näppäinyhdistelmän takaa löytyvä editoritoiminto, joka tekee uuden tyhjän GameObjectin ja siirtää kaikki valittuna olevat GameObjectit tämän säiliöobjektin lapsiksi. Näin saadaan esimerkiksi kaikki pelikentältä löytyvät puut kerralla yhden säiliöobjektin alle. Vastaavasti voitaisiin helposti toteuttaa

toiminto, jolla luodaan valittujen GameObjectien alle lapsiobjektit, jotka voisivat tyhjien objektien sijaan olla esimerkiksi valoja.

### 4.2.7 Custom context menu

Näppäinoikoteiden lisäksi Unity-editorissa voidaan muokata myös hiiren oikean painikkeen toimintaa minkä tahansa komponentin kohdalla painettaessa. Näin voidaan nopeuttaa lähes mitä tahansa editorissa tehtävää toimintoa kuten GameObjectien lisäämistä sceneen. Esimerkkiprojektissa toteutettiin toiminto, joka mahdollistaa Transform-komponenttien ominaisuuksien kopioimisen ja liittämisen GameObjectien välillä niin, että sekä objektin sijainti, rotaatio että skaala voidaan siirtää vaivattomasti objektilta toiselle joko yhdessä tai erikseen. Kuvassa 16 nähdään funktio, joka kopioi valittuna olevan objektin Transform-komponentin rotaatio-muuttujan leikepöydälle.

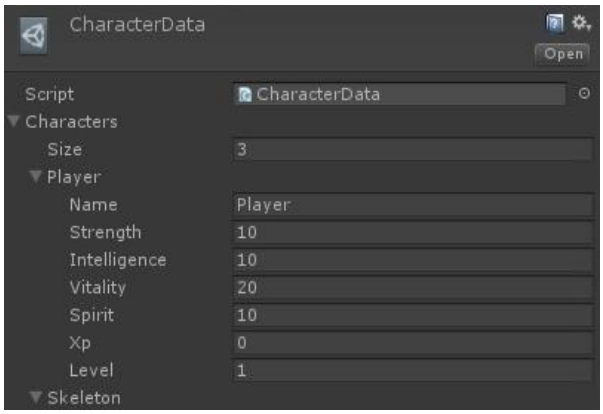
```
[MenuItem("CONTEXT/Transform/Copy Rotation", false, 152)]
static void CopyRotation()
{
    clipboard.rotation = Selection.activeTransform.localRotation;
    clipboard.isRotationSet = true;
}
```

KUVA 16. Transform-komponentin rotaation kopiointifunktio (Visual Studio Ultimate 2013, kuvankaappaus)

# 5 PELIRUNGON TOTEUTUS

## 5.1 Hahmojärjestelmä

Pelirungon hahmojärjestelmä toteutettiin niin, että sekä pelaajahahmo että tietokoneen ohjaamat viholliset periytyvät yhteisestä Character-luokasta. Tämä luokka sisältää kaikille hahmoille yhteiset ominaisuudet ja funktiot, kuten esimerkiksi vaurion ottamisen. Kun hahmot luodaan pelimaailmaan, haetaan niiden ominaisuudet tallennetusta hahmodatasta niiden nimen perusteella. Tätä varten hahmodatan sisältävältä CharacterData-luokalta löytyy GetCharacters()-funktio, joka palauttaa tallennetun hahmodatan helposti käytettävässä Dictionary-muodossa. Tästä hahmosanakirjasta löytyy hahmon nimen perusteella sen kaikki ominaisuudet. Kuvassa 17 nähdään luvussa 4.2.2 esitellyn datatyökalun tuottamaa, pelin käyttöön sopivaksi formatoitua hahmodataa.

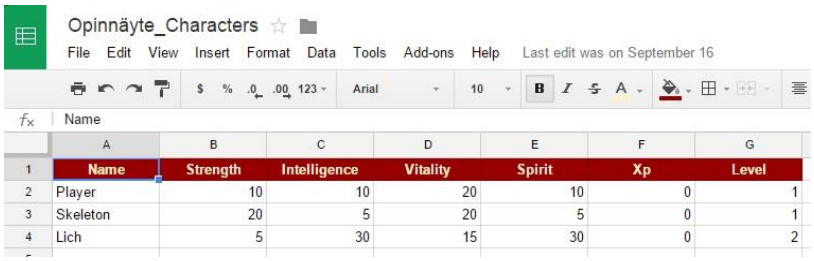



KUVA 17. Käyttövalmista hahmodataa (Unity 4.5.3f3 2014, kuvankaappaus)

Hahmodatatyökalujen ansiosta pelitasapainon muokkaaminen pelihahmojen ominaisuuksia muuttamalla ei vaadi muuta kuin uuden CSV-tiedoston tuomisen projektiin. Myös tulevaisuudessa lisättävät hahmot hakevat itselleen automaattisesti oikeat ominaisuudet, kunhan niiden nimi Unityssä on sama kuin Google Drivessä.

Google Driven taulukkolaskentaa varten tarvitaan ilmainen Google-tili. Tilin luomisen jälkeen toimistosovellukset ovat käytettävissä selaimen kautta. Itse taulukkolaskentaohjelma on käyttöliittymältään samankaltainen kuin Microsoft Excel tai OpenOffice Calc. Esimerkkiprojektissa ohjelmaa käytettiin vain tietojen helppoon taulukointiin ja tallenta-

miseen CSV-muodossa, mutta taulukkolaskennalla voidaan myös esimerkiksi testata taistelujärjestelmän kaavojen toimivuutta. Kuvassa 18 on esitetty Google Drivessä tehty taulukko pelihahmojen ominaisuuksista.




| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Name | Strength | Intelligence | Vitality | Spirit | Xp | Level |
| 2 | Player | 10 | 10 | 20 | 10 | 0 | 1 |
| 3 | Skeleton | 20 | 5 | 20 | 5 | 0 | 1 |
| 4 | Lich | 5 | 30 | 15 | 30 | 0 | 2 |

KUVA 18. Pelihahmojen ominaisuuksia Google Driven taulukkolaskennassa (Google Docs Spreadsheets 2014, kuvankaappaus)

## 5.2 Pelin kulku

Itse pelissä pelaajan ohjaama hahmo luodaan pelikentälle satunnaiseen paikkaan, minkä jälkeen peliä hallitseva skripti luo jatkuvasti lisää vihollisia satunnaisille paikoille. Pelaaja voi hiirtä käyttäen liikkua maailmassa ja taistella vihollisia vastaan joko tulipalloin tai lähietäisyydeltä miekalla. Vihollisten tappamisesta pelaaja saa kokemusta ja kokemuksen kasvaessa pelihahmon perusominaisuuksia voi parantaa. Pelilogiikka jätettiin melko yksinkertaiseksi, mutta sen toteutuksessa käytettiin joitakin hyödyllisiä toimintoja.

Pelaajan ja vihollisten ilmestymispaikat arvotaan satunnaisesti ennalta määrättyjen pisteiden joukosta. Näiden pisteiden ei ole syytä näkyä itse pelissä, vaan ne voivat olla tyhjiä GameObjecteja, joilla on vain pakollinen Transform-komponentti. Näkymättömien objektien tarkka sijoittaminen editorissa on kuitenkin työlästä, joten hahmojen luomisesta huolehtivaan skriptiin toteutettiin OnDrawGizmos()-funktio. Gizmot ovat vain editorinäkymässä piirrettäviä apugrafiikoita, joita voidaan käyttää hahmottamaan selkeästi erilaisia pisteitä ja etäisyyksiä pelimaailmassa. Kuvassa 19 näkyy, kuinka skripti piirtää editorinäkymään siniset pallot pelaajan mahdollisille ilmestymispaikoille ja punaiset pallot vihollisten ilmestymispaikoille. Lisäksi näkyvissä ovat violetit ja syaanit kameran rajat, joiden sisälle pelikameran liike on rajoitettu.


KUVA 19. Gizmot kuvaavat pelaajan ja vihollisten ilmestymispaikkoja sekä kameran rajoja. (Unity 4.5.3f3 2014, kuvankaappaus)

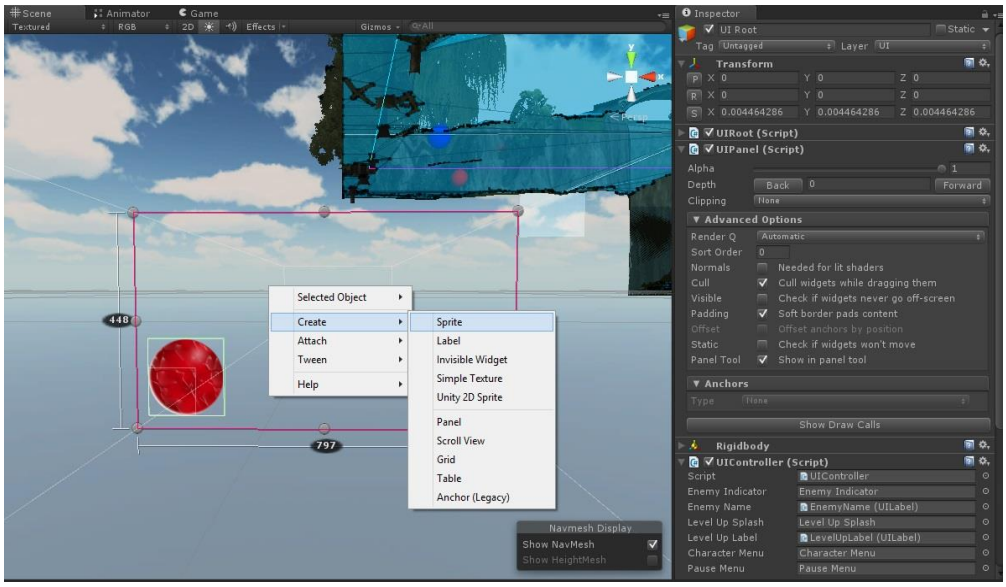
## 5.3 Pelikentän maasto

Esimerkkiprojektin pelimaaston tekemisessä ei käytetty mitään erityisiä työkaluja, mutta eräältä kokeneelta kenttäsuunnittelijalta opittu tekniikka nopeutti työtä huomattavasti. Maasto tehtiin Unityn omalla Terrain-työkalulla, jolla alussa tasaiselle tasolle voidaan erilaisilla siveltimillä maalata korkeuseroja ja tekstuureita. Ennen työn aloittamista Unityssä piirrettiin pelikentästä Microsoft Paintilla yksinkertainen suunnitelma kentältä löytyvistä teistä, vuorista ja muista elementeistä. Kuten kuvassa 20 nähdään, tämä suunnitelma voitiin Unityssä heijastaa osittain läpinäkyvänä Terrain-objektin kokoiselle tasolle, minkä jälkeen korkeuserojen ja tekstuurien suuret linjat voitiin muokata nopeasti tämän muotin mukaan.

KUVA 20. Microsoft Paint -ohjelmalla piirretty design-ohjekuva uuden Terrain-objektin päällä sekä oikealla valmis pelimaasto (Unity 4.5.3f3 2014, kuvankaappaus)

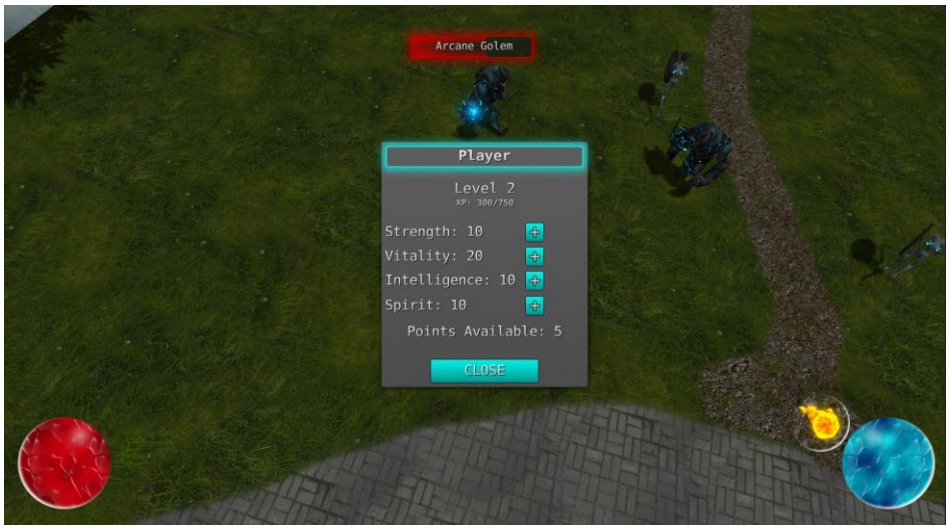
## 5.4 Käyttöliittymä

Pelin käyttöliittymä toteutettiin kokonaan NGUI:lla. Aluksi sceneen lisätään UI Root -objekti, joka pitää sisällään kaikki käyttöliittymän perusasetukset, kuten halutaanko tehdä pikselintarkkaa vai näytön tarkkuuden mukaan skaalautuvaa käyttöliittymää. Tämän pääobjektin alle lisätään kaikki muut halutut UI-elementit. Elementtien lisääminen onnistuu helpoiten käyttämällä hiiren oikeasta painikkeesta avautuvaa valikkoa, joka nähdään kuvassa 21.




KUVA 21. Uuden sprite-kuvan lisääminen NGUI:lla (Unity 4.5.3f3 2014, kuvankaappaus)

Pelin sisäinen käyttöliittymä koostuu kuudesta elementistä: pelaajan terveys- ja taikavoimaindikaattoreista, tulipallon latausaikaindikaattorista, pelaajan hahmoruudusta, pelin keskeytysvalikosta, hiirellä osoitetun vihollisen terveysindikaattorista sekä pelaajan tasonnousun myötä näytettävästä koristeesta. Kaikki nämä on tehty yhdistelemällä UISprite- sekä UILabel-komponentteja ja animoimalla niitä NGUI:n omilla UITweener-komponenteilla. UI-elementtejä voidaan myös helposti manipuloida koodista käsin, esimerkiksi pelaajan resurssi-indikaattorit päivittyvät reaaliajassa pelitilanteen mukaan. Kuvassa 22 esitetään pelitilanne, jossa pelaaja on juuri saanut uuden tason ja avannut hahmoruudun.




KUVA 22. Pelin käyttöliittymä; hahmoruutu näyttää pelaajan ominaisuudet (Rime Quest 0.0.1 2014, kuvankaappaus)

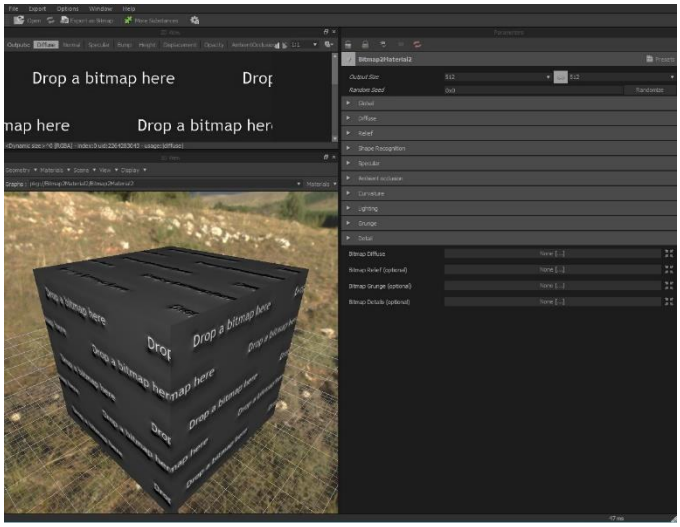
Päävalikkoon toteutettiin cInputin avulla ajonaikainen pelinäppäinten muokkaus, jonka avulla pelaaja voi vaihtaa hyökkäyksen, tulipallon ampumisen, hahmoruudun avaamisen sekä pelin keskeytyksen tapahtumaan haluamistaan näppäimistä. Aina peliä ladattaessa skripti tarkistaa, onko cInputissa määritettynä kaikki tarvittavat näppäimet, mikäli näin ei ole, määritetään pelille kehittäjän asettamat oletusnäppäimet. Tämän jälkeen pelaaja voi muuttaa näppäimet haluamikseen, ja cInput tallentaa nämä määritykset kovalevylle tulevia pelikertoja varten.

## 5.5 Tekstuurit

Vihollisten tekstuurit sekä kaikki maastotekstuurit työstettiin itse otetuista valokuvista Bitmap2Materialin avulla. Kuvista tehtiin toistuvat perustekstuurit sekä normaalikartat

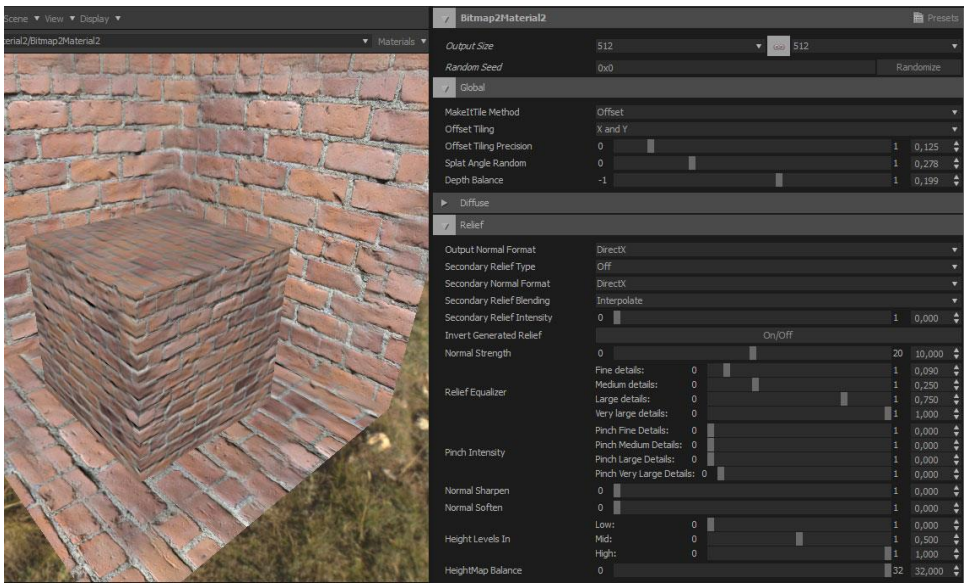
tuomaan lisää yksityiskohtia. UI-tekstuureina käytettiin pääasiassa NGUI:n mukana tulleita esimerkkitekstuureita, joiden värejä muokattiin editorin sisällä.

Esimerkkiprojektia varten käytettiin Bitmap2Materialin halvempaa standard-versiota, joka on saatavilla Steam-jakelupalvelun kautta. Ostamisen jälkeen ohjelma asennetaan ja käynnistetään Steamin käyttöliittymän kautta. Steam myös huolehtii automaattisesti ohjelman päivittämisestä. Kun ohjelma käynnistetään, aukeaa kuvassa 23 näkyvä käyttöliittymä.




KUVA 23. Bitmap2Material-ohjelman käyttöliittymä: tekstuuritiedoston esikatselu vasemmalla ylhäällä, materiaalin esikatselu vasemmalla alhaalla ja materiaalin asetukset oikealla (Bitmap2Material 2.2 2014, kuvankaappaus)

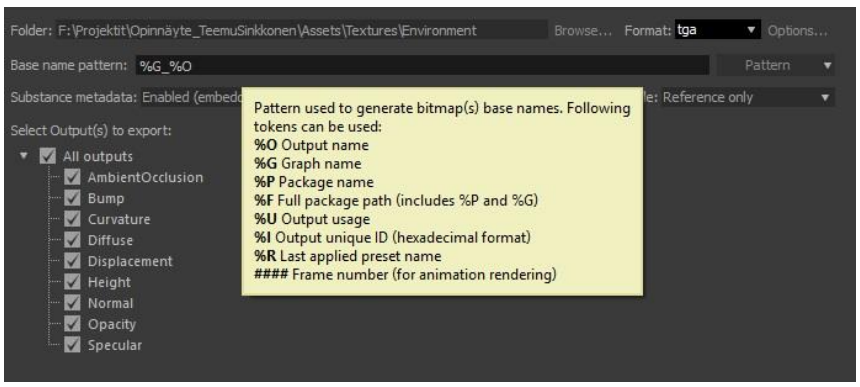
Ohjelman käyttöliittymä on varsin yksinkertainen. Haluttu bittikarttakuva voidaan joko avata ohjelman kautta tai vetää suoraan kuvassa 5 näkyvään tekstuurin esikatseluikkunaan. Materiaalin esikatselu päivittyy reaaliaikaisesti käyttäjän tehdessä muutoksia ja esikatseluun voi käyttää erilaisia primitiivimuotoja ja valaistusasetuksia materiaalin käyttötarkoituksesta riippuen. Kuvassa 24 nähdään materiaalin toistumisen sekä muodontunnistuksen asetuksia sekä tiiliseinätekstuurin esikatselu.




KUVA 24. Materiaalin asetuksia sekä kuutio kuution sisällä -esikatselu (Bitmap2Material 2.2 2014, kuvankaappaus)

Materiaalin asetuksista säädetään muun muassa valmiiden tekstuuritiedostojen koko sekä se, halutaanko tekstuureista tehdä toistuvia (*tiling*) vai ei. Lisäksi voidaan säätää kuvankäsittelyohjelman tapaan kuvan väritasapainoa, terävyyttä ja värikylläisyyttä. Kenties tärkeimpiä asetuksia ovat muodontunnistukseen liittyvät asetukset, joiden perusteella luodaan materiaalin normaalikartta ja muut erikoistekstuurit. Näillä asetuksilla kaksiulotteiseen bittikarttaan saadaan luotua kolmiulotteisuuden vaikutelmaa, mikä tekee tekstuureista näyttävämpiä pelin sisällä.

Kun materiaali on valmis, viedään se ulos ohjelmasta käyttäen Export as Bitmap -toimintoa. Tuloksena syntyy joukko tekstuuritiedostoja, jotka kootaan Unity-editorissa materiaaliksi. Kuvassa 25 nähdään materiaalin viennin asetuksia sekä selite kuvatiedostojen nimien formaatista.

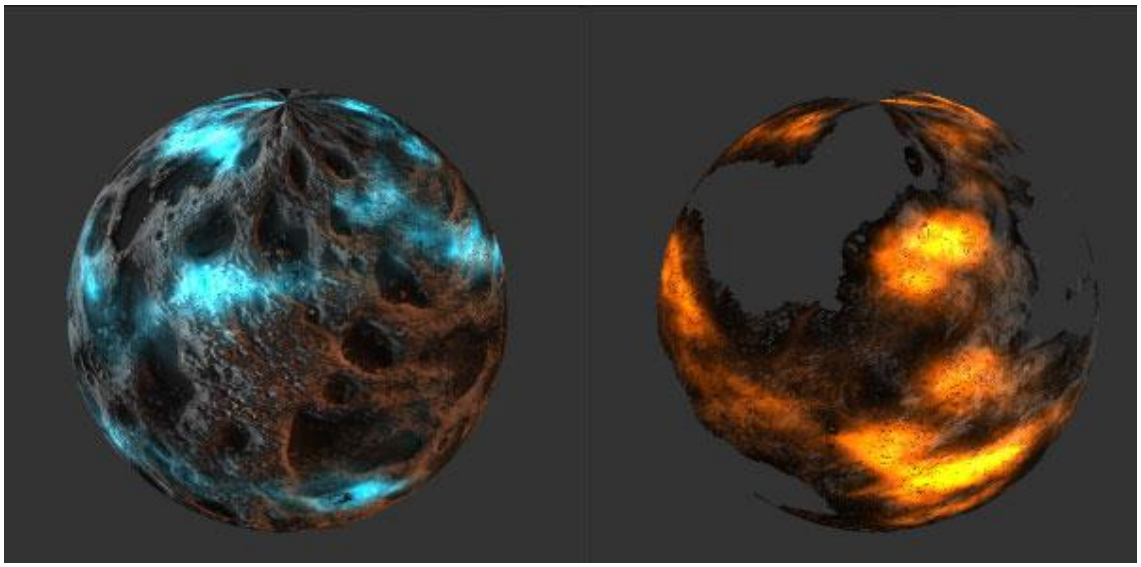



KUVA 25. Materiaalin viennin asetuksia (Bitmap2Material 2.2 2014, kuvankaappaus)

## 5.6 Visuaaliset tehosteet

### 5.6.1 Shaderit

Shaderit ovat objektien materiaaleihin liitettäviä skriptejä, jotka määräävät, millaisia ominaisuuksia materiaalilla on. Yksinkertainen shader ainoastaan piirtää objektin pinnalle halutun tekstuurin, kun taas monimutkaisemmilla shadereilla voidaan kuvata vaikkapa räjähdyksen käyttäytymistä realistisesti. Projektia varten toteutettiin joitakin erikoisshadereita tuomaan näyttävyyttä tiettyihin pelielementteihin ja kokeilemaan Shader Forgen tuomia mahdollisuuksia. Kuvassa 26 nähdään esimerkit kahdesta eri shaderista.

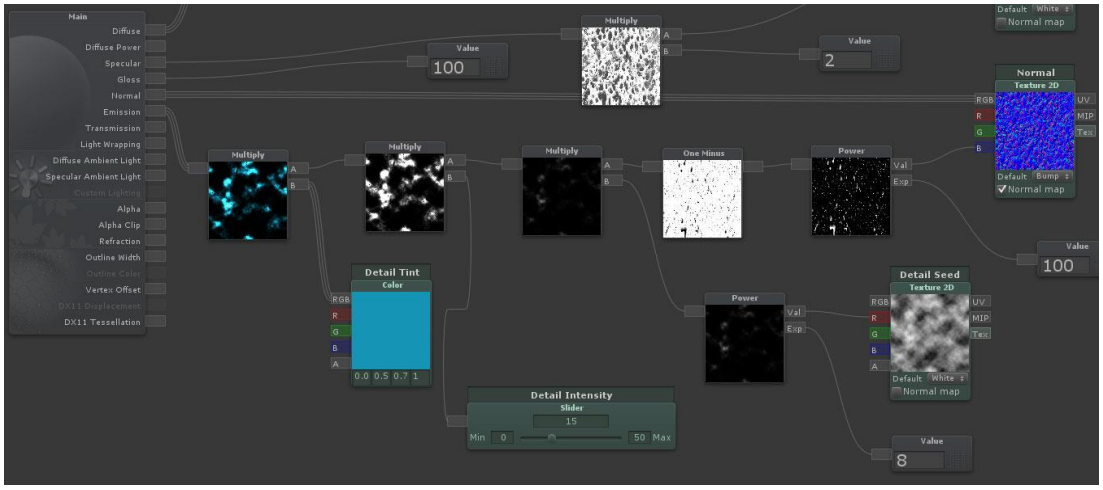


KUVA 26. Shader Forgella tehtyjä shader-efektejä (Unity 4.5.3f3 2014, kuvankaappaus)

Pelaajan tulipallohyökkäykseen toteutettiin shader, joka pyrkii jäljittelemään osittain puhkipalanutta, liikkuvaa ja kuumuutta hehkuvaa massaa. Shader heijastaa normaalikartalla ehostetun pohjatekstuurin päälle kolmannesta detail-tekstuurista johdettuja hohtavia yksityiskohtia. Lisäksi kokonaisuus on animoitu UV-animaatiolla ja tehty osittain näkymättömäksi normaalikartasta johdetulla alpha clipping -maskilla.

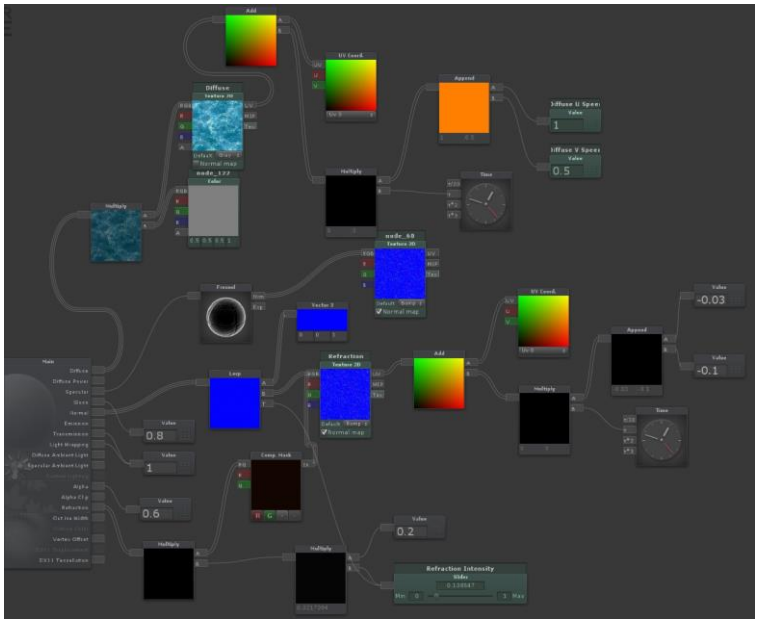
Arcane Golem -vihollisia varten toteutettiin shader, joka lisää pohjatekstuuriin normaalikartan ja hohtavien yksityiskohtien lisäksi heikosti kiiltävän pinnan. Tämä shader mahdollistaa myös hohtavien yksityiskohtien värin ja valovoiman säätelyn jopa ajonaikaisesti. Kuten kuvasta 22 voidaan havaita, on yksityiskohdilla kirkkaimmissa kohdissa niin voimakkaat kirkkausarvot, että pelin sisällä pelikameraan yhdistetty Bloom-komponentti

saa ne hohtamaan näyttävästi. Kuvassa 27 nähdään osa shaderin toteutusta Shader Forgella.




KUVA 27. Arcane Golem -shaderin toteutusta Shader Forgella (Unity 4.5.3f3 2014, kuvankaappaus)

Pelikentällä virtaavat joet toteutettiin myös heijastamalla itse tehdyllä shaderilla varustettu vesitekstuuri kahdesta polygonista koostuvalle tasolle. Vesishaderissa sekä pohjatekstuuri että normaalikartta on animoitu UV-animaatiolla, mistä syntyy juoksevan veden illuusio. Lisäksi materiaalin heijastuspisteiden väriä säädellään fresnel-efektillä, mikä tekee veden pinnasta dynaamisemman. Shaderiin on myös toteutettu läpinäkyvyys sekä editorissa säädettävä valon taittoarvo, joka vaikuttaa veden läpi nähtyjen objektien piirtämiseen. Kuvassa 28 on esitetty vesishaderin toteutus kokonaisuudessaan.



KUVA 28. Vesishaderin toteutus Shader Forgella (Unity 4.5.3f3 2014, kuvankaappaus)

### 5.6.2 Partikkeliefektit

Pelihahmon tulipallohyökkäykseen toteutettiin partikkeliefektit hyödyntäen Hayaten ominaisuuksia. Sekä suuremmat liekkipartikkelit että pienet kipinäpartikkelit laitettiin seuraamaan tulipallo-objektia, mutta suuremmat liekit pyrkivät pysymään pallon muotoisen objektin sisällä, kun taas kipinät sinkoilevat sen ulkopuolelle. Seurattava piste määritellään yksinkertaisesti lisäämällä halutun partikkeliefektin sisältävälle GameObjectille Hayate-komponentti, josta kytketään päälle Follow point -asetus. Tämän jälkeen seurattava objekti vedetään Follow Transform -kenttään tai valitaan listasta. Kuvassa 29 nähdään osa Hayaten tarjoamista asetuksista sekä valmis tulipallo-objekti partikkeleineen.


KUVA 29. Hayaten asetuksia sekä valmis tulipallo (Unity 4.5.3f3 2014, kuvankaappaus)

# 6 POHDINTA

Esimerkkiprojektin toteutus onnistui hyvin ja tilaajan palautteen perusteella myös asetettuihin tavoitteisiin päästiin. Projekti havainnollistaa ohjelmointia ennestään tunteville käyttäjille Unity-editorin tarjoamia mahdollisuuksia ja esittelee yrityksen käyttöön soveltuvia editorilaajennuksia sekä työkaluja. Projektia saatetaan myös käyttää tulevan kaupallisen roolipeliprojektin pohjana. Mikäli projektia jatketaan, ensimmäisenä toteutetaan nykyistä kattavampi datatyökalu, joka koostaa uusista pelihahmoista valmiit prefab-paketit. Lisäksi visuaalinen työkalu vihollisten tekoälyrutiinien luomiseen ja muokkaamiseen helpottaisi työskentelyä tulevaisuudessa.

Työn aihealue oli melko laaja ja projektin toteuttamiseenkin meni hieman suunniteltua enemmän aikaa. Tämä tosin johtui suurilta osin työn mielenkiintoisuudesta, erityisesti Shader Forge osoittautui niin monipuoliseksi ja hyväksi työkaluksi, että uusien shadereiden tekemiseen käytti aikaa mielellään. Suurimpana haasteena työssä olikin aihealueen rajaaminen realistiseen mittakaavaan. Tekninen toteutus sinällään ei tuottanut suuria ongelmia. Hieman työn toteutusosaa sekä raportin kirjoittamista vaikeutti myös aihetta käsittelevien kattavien lähteiden puute. Suuri osa käytetystä tiedosta oli työpaikalla sekä oman tekemisen kautta opittua hiljaista tietoa.

Kaikki projektissa käytetyt Unity Asset Storesta ladatut editorilaajennukset ovat maksullisia ja niiden hinta vaihtelee 14,25 eurosta 90,25 euroon. Kaikki laajennukset ovat myös hintansa arvoisia verrattuna niiden avulla säästettyihin työtunteihin ja erityisesti Shader Forge mahdollistaa Unityn mittapuulla lähes uskomattomat visuaaliset efektit. NGUI on laajennuksista ainoa, jota ei voi suositella täysin varauksetta, sillä Unityn versiossa 4.6 julkaistava uusi UI-systeemi on ilmainen ja beta-version kokeilun perusteella vähintään yhtä hyvä, ellei jopa parempi. NGUI on myös laajennuksista kallein; yhden käyttäjän lisenssi maksaa 90,25 ja koko studion kattava lisenssi 2 000 euroa.

Itse toteutettuihin editorilaajennuksiin löytyi runsaasti malleja Unity Technologiesin ylläpitämästä Unify Wikistä, jonne Unity-kehittäjät saavat ladata omia skriptejään vapaaseen levitykseen. Omien laajennusten paras puoli on luonnollisesti se, että kehittäjä tuntee niiden toiminnan täysin, ja ne voidaan alusta asti räätälöidä nimenomaan omaa peliprojektia täydellisesti tukeviksi. Suuria kokonaisuuksia, kuten uutta UI-systeemiä, on turha

lähteä rakentamaan ainakaan pienellä projektiryhmällä, mutta jokaisen Unity-kehittäjän olisi hyvä tuntea ainakin assettien käsittelijöiden tasoiset perusasiat.

Ulkoisista työkaluista Bitmap2Material on oiva tapa tuottaa etenkin näyttävän näköisiä maastotekstuureita suhteellisen vaivattomasti, kunhan saatavilla on riittävän korkealaatuisia valokuvia tai muita bittikarttoja. Vaikka peliin ei tavoiteltaisikaan fotorealistista grafiikkaa, ohjelmaa voi silti hyödyntää esimerkiksi normaalikarttojen generoimiseen tai saumattomasti toistuvien tekstuurien tekemiseen. Google Driven hyödyntäminen projektissa esitellyllä tavalla vähentää muiden kuin ohjelmoijien tarvetta puuttua pelin tekniseen toteutukseen, mikä on aina positiivinen asia vähintään versionhallinnan kannalta.

Opinnäytetyön tekeminen osoitti, että Unityn editorin laajentamiseen voisi käyttää vähintäänkin yhtä paljon aikaa kuin varsinaiseen pelien tekemiseen. Uutta oppia kertyi myös runsaasti, erityisesti shaderien osalta. Esimerkkiprojektin suunnittelu haastoi ajattelemaan paitsi ohjelmoijan, myös pelisuunnittelijan ja graafikon näkökulmasta, mikä avarsi näkökulmaani koko pelinkehitysprosessiin yleisesti.

**LÄHTEET**

Bernardoff, C. 2014. NGUI for Unity. Packt Publishing.

Bitmap2Material 2.2. 2014. Allegorithmic SAS.

De Byl, P. 2012. Holistic Game Development with Unity. Focal Press.

Jackson, S. 2014. Mastering Unity 2D Game Development. Packt Publishing.

Loeliger, J & McCullough, M. 2012. Version Control with Git, 2nd Edition. O'Reilly Media, Inc.

Menard, M. & Wagstaff, B. 2014. Game Development with Unity®, Second Edition. Course Technology PTR.

Rime Quest 0.0.1. 2014. Rimeforge Entertainment.

Thorn, A. 2013. Unity 4 Fundamentals: Get Started Making Games with Unity. Focal Press.

Unity 4.5.3f3. 2014. Unity Technologies.

Unity Technologies. 2014a. Shaders: ShaderLab & Fixed Function shaders. Luettu 15.10.2014. http://docs.unity3d.com/Manual/ShaderTut1.html

Unity Technologies. 2014b. Unity Scripting API: Transform. Luettu 15.10.2014. http://docs.unity3d.com/ScriptReference/Transform.html

Visual Studio Ultimate. 2013. Microsoft Corporation.