Opinnäytetyö (AMK)

Tietotekniikka

NTIETS12P

2018

Martti Gullström

ROCKODILE GAMES - YRITYKSEN PELIMOOTTORIN SISÄLLÖNTUOTTOTYÖKALU

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka

2018 | 28

Martti Gullström

ROCKODILE GAMES -YRITYKSEN PELIMOOTTORIN SISÄLLÖNTUOTTOTYÖKALU

Opinnäytetyön tavoitteena oli tehdä Rockodile Games -yrityksen peli- ja pelimoottoriprojektiin Godsbane omat sisällöntuottotyökalut. Samalla yritettiin minimoida työmäärää ja maksimoida työstä saatava hyöty.

Työkalut yritettiin tehdä ottamalla mallia jo olemassa olevien pelien ja pelimoottorien työkaluista. Projektin edetessä, tuli peliin lisää ominaisuuksia, jotka tarvitsivat omia muokkaustyökaluja. Ominaisuuksien ja työkalujen määrän kasvaessa, myös ylläpidettävän koodin määrä kasvoi. Loppujen lopuksi ylläpidettävän koodin määrä kasvoi liian suureksi.

Projektin loputtua työkalut olivat siinä pisteessä, että niillä sai tehtyä nopeasti sisältöä pelimoottoriin. Godsbane pelinä jäi tekniikkademon tasolle ja projekti hylättiin. Koska työkalut oli tehty käyttäen Microsoft Windows Presentation Foundation -käyttöliittymätyökaluja ja logiikka on tehty omalle pelimoottorille, ei tehtyä ohjelmakoodia voitu käyttää suoraan hyödyksi muissa projekteissa.

Rockodile Games ei lähitulevaisuudessa tule tekemään omaa pelimoottoria, vaan käyttää kaupallisia tuotteita. Jos kaupallisiin pelimoottoreihin ei ole saatavilla tarvittavia työkaluja, eikä liitännäisiä, yritetään työkalut tehdä pelimoottorin sisälle, eikä erillisenä ohjelmana, tai pelimekaniikan, joka työkalua tarvitsee, tarvetta arvioidaan uudelleen.

ASIASANAT:

pelinkehitys, pelimoottori, kehitystyökalu

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Game Technology

2018 | Total number of pages

Martti Gullström

CONTENT CREATION TOOLS FOR ROCKODILE GAMES' GAME ENGINE

The aim of this thesis was to make content creation tools for Rockodile Games' game and game engine project Godsbane. At the same time, effort was made to minimize the work required and maximize the amount of benefit gained from the work.

The content creation tools were made by using already existing tools in games and game engines as examples. As the project progressed, new features were added to the game that needed new tools. As the number of features and tools kept growing, the amount of code needing to be maintained also grew. In the end the amount of code needing to be maintained became too large.

At the end of the project, the tools were usable for fast content production for Rockodile's custom engine. Godsbane as a game was left in a state of technical demo and the project was abandoned. Since the tools were made using Microsoft Windows Presentation Foundation user interface tools and the logic was made for their own game engine, no code would be directly reusable in future projects.

Rockodile Games won't create another game engine in the near future, but will switch to using commercial engines. If selected commercial engine would lack the tools needed for content types and there is no-addon available, the custom tool will be built into the engine instead of as a external program, or the need for the content type will be re-evaluated.

KEYWORDS:

game development, game engine, development tool.

SISÄLTÖ

KÄYTETTY SANASTO **6**

1 JOHDANTO **7**

2 MITÄ SISÄLTÖÄ PELEISSÄ ON **8**
2.1 Grafiikka- ja äänisisältö 8
2.2 Kentät ja käyttöliittymät 9
2.3 Tietokoneen hallitsemien hahmojen käyttäytymismallit 9
2.4 Tarinat ja teksti 10
2.5 Komentosarjaohjelmointi 10

3 SISÄLLÖNLUONTITYÖKALUT JULKAISTUISSA PELEISSÄ **12**
3.1 Modifioinnin eri tasot 13
3.2 Jälkikäteen lisätyn sisällön hyöty 13

4 PELIMOOTTORIEN ULKOPUOLISET TYÖKALUT **14**
4.1 Itsenäisten työkalujen puute 14
4.2 Pelimoottorien sisäisten työkalujen käyttö ulkopuolisessa ohjelmassa 14
4.3 Työkalujen käyttö niille ei-tarkoitetun sisällön tuottamiseen 15

5 PROSEDURAALINEN JA IHMISEN LUOMA SISÄLTÖ **16**
5.1 Proseduraalisesti tuotettu maailma 16
5.2 Ihmisen luoma kenttä 17
5.3 Modulaarisuus ihmisen luomassa sisällössä 17

6 TYÖKALUOHJELMA OMAAN PELIMOOTTORIIN JA GODSBANE PELIIN **19**
6.1 Kenttä- ja moduulieditori 19
6.2 Kampanjaeditori 21
6.3 Toiminto- ja taitoeditori 21
6.4 Toimijaeditori 21
6.5 Blender-mallinnusohjelman Python-liitännäinen 22
6.6 Käyttäytymiseditori 23
6.7 Vaikutuseditori 23
6.8 Tekstieditori 23
6.9 Sisällön tallentaminen 24

6.10 Satunnaisesti valitun sisällön lisääminen jälkikäteen 26

7 YHTEENVETO **27**

LÄHTEET **28**

LIITTEET

Liite 1. Liitteen otsikko.
Liite 2. Kaavat, kuvat, kuviot ja taulukot

KAAVAT

Kaava 1. Esimerkki kaavasta (Lähdeviite). 17

KUVAT

Kuva 1. Liitteen ylätunnisteen muuttaminen. 12
Kuva 2. Osanvaihtokoodi. 13
Kuva 3. Esimerkki kuvaotsikon lisäämisestä. 14
Kuva 4. Uuden otsikkolajin lisääminen. 15

KUVIOT

Kuvio 1. Esimerkki kuviosta (Lähdeviite). 16

TAULUKOT

Taulukko 1. Esimerkki taulukosta. 16

KÄYTETTY SANASTO

Editori	Sisällön tuottamiseen ja muokkaamiseen tarkoitettu työkalu
Modifiointi	Pelin sisältötiedoston muokkaaminen kuluttajan toimesta
Ohjelmakirjasto	Koodipaketti, jonka toimintoja voidaan käyttää vähentämään projektin ohjelmointityön määrää.
Pelimoottori	Pelien tekemiseen tarkoitettu ohjelmointikirjasto, joka hoitaa useita eri tehtäviä, kuten näppäinpainalluksien lukemisen, näytölle piirtämisen ja äänen toiston.
Komentosarja	Usein lyhyt, tulkattu ohjelmakoodi, joka ajettaessa tietyssä ympäristössä tekee tietyn toiminnon.
Toimija	Yleiskäsite kaikille pelikentässä sijaitseville esineille, asioille ja hahmoille.
UI	User interface. Käyttöliittymä, jota käyttäen käyttäjä on vuorovaikutuksessa pelin kanssa
Liitännäinen	Olemassa olevan ohjelman ominaisuuksia lisäävä tai muokkaava ohjelma.

1 JOHDANTO

Sisällöntuottotyökalut ovat ryhmä ohjelmatyökaluja, joiden avulla peleihin tehdään sisältöä. Ilman oikeita työkaluja on sisällön tuottaminen joko äärettömän raskasta tai käytännössä mahdotonta. Pelimoottoreissa on useita eri työkaluja sisäänrakennettuna. Jos tarvittavaa työkalua ei löydy moottorin sisältä eikä markkinoilla ole tarpeeseen tarkoitettua työkalua, on todennäköistä, että työkalu pitää tehdä itse.

Opinnäytetyön tavoitteena oli tuottaa sisällönluontityökalut Godsbane-nimiseen peli- ja pelimoottoriprojektiin, jotta kaikille halutuille mekaniikoille on olemassa sisältöä. Tämän lisäksi työssä oli tarkoitus lisätä tuotetun sisällön määrää samalla vähentäen tarvittavan työn määrää. Projekti kesti noin vuoden ja päättyi sen peruuttamiseen.

Työssä kerrotaan ensimmäisissä luvuissa yleisesti sisällöstä sekä sisällöntuotosta. Tämän jälkeen työssä esitetään, miten sisällöntuottoon tarkoitettuja työkaluja voidaan käyttää hyödyksi pelien julkaisun jälkeen. Neljännessä luvussa käsitellään menetelmiä, kun työkalua tiettyyn tehtävään ei näytä olevan olemassa. Työn viides luku kertoo miten vähäisestä työmäärästä saa normaalia enemmän irti. Kuudes luku keskittyy kertomaan työkalujen tekemisestä sekä tuloksista. Viimeinen luku kertoo lopputuloksista ja siitä mitä opittiin.

2 MITÄ SISÄLTÖÄ PELEISSÄ ON

Pelit koostuvat sitä ohjaavasta koodista sekä pelin sisällöstä. Sisältöä on kärjistettynä kaikki mitä pelaaja voi nähdä tai kuulla tai johon pelaaja voi vaikuttaa (Shaker ym. 2016, 1). Sisältö voidaan upottaa koodiin kunkin pelin lähdekoodiin, jolloin sitä kutsutaan kovakoodatuksi sisällöksi. Vaihtoehtoisesti sisältö voidaan ladata dynaamisesti. Esimerkiksi pelin käynnistyessä pelimoottori voi lukea päävalikon taustakuvan kiintolevyltä ja tallentaa sen muistiin.

Ennen kuin pelimoottori voi lukea sisältöä, pitää se olla tietyssä muodossa ja tietyssä paikassa. Tämän muodon ja paikan määrittävät erilaiset työkalut. Esimerkiksi monet grafiikkatyökalut voivat tallentaa päävalikon taustakuvan bittikarttatiedostomuodossa, jonka lukeminen ja käsittely on yleisesti tunnettu toimenpide (Wikipedia 2003). Tämänlainen työkalu on siis pelimoottorista riippumaton. Vastaavasti Unreal Engine -moottorin tekoälyä mallintava käyttäytymispuu-työkalu tai Unity-moottorin animaatiotyökalu ovat pelimoottorikohtaisia, eikä niiden käyttö moottorien ulkopuolella ole työmäärältään järkevää tai edes mahdollista.

Eri pelimoottorit säilövät sisältötiedostot eri tavoin, mutta kun sisältötiedosto ladataan kiintolevyltä muistiin, annetaan sille oma sisältötunnus. Tunnus voi olla esimerkiksi sisältötiedoston nimi, tai satunnaisesti luotu numero. Tätä tunnusta käyttäen eri järjestelmät voivat viitata sisältötiedostoon.

2.1 Grafiikka- ja äänisisältö

3d- ja 2d-grafiikka ja ääni- sekä animaatiosisältö tuotetaan usein niille suunnatuissa ohjelmissa. Nämä ohjelmat pystyvät viemään sisällön standardoituihin tiedostotyyppeihin, joiden lukeminen pelimoottorissa on tuettu. Näille sisältötyypeille harvoin tehdään pelimoottoriin muokkaustyökaluja, sillä audiovisuaalista sisältöä tekevät artistit käyttävät valmiiksi oppimiaan työkaluja, joita on työstetty pitkään.

2.2 Kentät ja käyttöliittymät

Pelin eri kentät tehdään yleensä pelimoottorin sisään rakennetuilla työkaluilla usein siksi, koska pelimoottorin tietorakenteet sekä ominaisuudet ovat erilaiset, eikä tämäntyyppiselle sisällölle ole standardia. On kuitenkin olemassa työkaluja, joilla voi tehdä kenttiä myös pelimoottoreista riippumatta. Esimerkiksi erilaisten 3d-mallinnusohjelmien avulla voidaan kenttä tuoda pelimoottoriin tavallisena 3d-mallina.

Pelien graafiset käyttöliittymät tehdään usein pelimoottorien omilla työkaluilla. Omaa pelimoottoria ohjelmoitaessa on syytä huomioida monialustallisuus, jos käyttää ohjelmointikirjastoja, sillä monikaan UI kirjasto ei tue kaikkia alustoja, kuten pelikonsoleita.

2.3 Tietokoneen hallitsemien hahmojen käyttäytymismallit

Peleissä on usein tietokoneen hallitsemia hahmoja, jotka tarvitsevat erilaisia käyttäytymismalleja tilanteesta riippuen. Esimerkiksi räiskintäpelissä vihollinen osaa yleensä liikkua kentässä väistellen esteitä ja ampua pelaajaa kohti. Esimerkin vihollinen tarvitsee polunetsintää, eli tavan etsiä kuljettava reitti kahden pisteen välillä, sekä logiikkaa päättääkseen mitä toimintoa vihollinen lähtee seuraavaksi suorittamaan. Toimintojen valintaan keskittyviä menetelmiä on monia, mutta graafisilla työkaluilla muokattavista yleisimmät ovat päättelylista ja käyttäytymispuu.

Kuva 1. Dragon Age: Origins -pelissä pelaaja voi tehdä joukkueensa hahmoille päättelylistan jota tietokone käyttää, kun pelaaja ei itse ohjaa hahmoa

Unreal Engine -pelimoottorissa on sisäänrakennettu graafinen solmupohjainen käyttäytymispuueditori jolla voi luoda monimutkaisia tietokoneen ohjaamia hahmoja.

Koska eri pelien sisältämät säännöt vaihtelevat paljon, on yleispätevää tekoälytyökalua lähes mahdotonta tehdä. Esimerkiksi ylhäältä päin kuvatun vuoropohjaisen strategiapelin hahmon siirtäminen ei vastaa reaaliaikaisen tasohyppelypelin sääntöjä ollenkaan. Tästä syystä, vaikka eri toimintojen valinta voidaan yleistää, ei toimintoja itseään voida, sillä toiminnot ovat peli- ja pelimoottoririippuvaisia.

2.4 Tarinat ja teksti

Peliprojekteissa projektin alkuvaiheessa usein tapahtuu muutoksia, varsinkin jos käytössä on iteroiva työskentelymenetelmä (Luton 2009).

Tästä syystä peleissä käytetyt tekstipätkät ovat usein kovakoodattu projektin alussa. Tekstiä varten omaa editoria tarvitaan vasta, kun vaihtuvan tekstin määrä ylittää kipukynnyksen, tai peli halutaan kääntää eri kielille. Tekstieditori voi toimia myös tarinan ja vuoropuheiden muokkaustyökaluna. Tekstisisällön yksinkertaisuuden takia voi pelkkä laskentataulukko riittää eri tekstitietueiden säilömiseen ja laskentataulukko-ohjelma niiden muokkaamiseen.

2.5 Komentosarjaohjelmointi

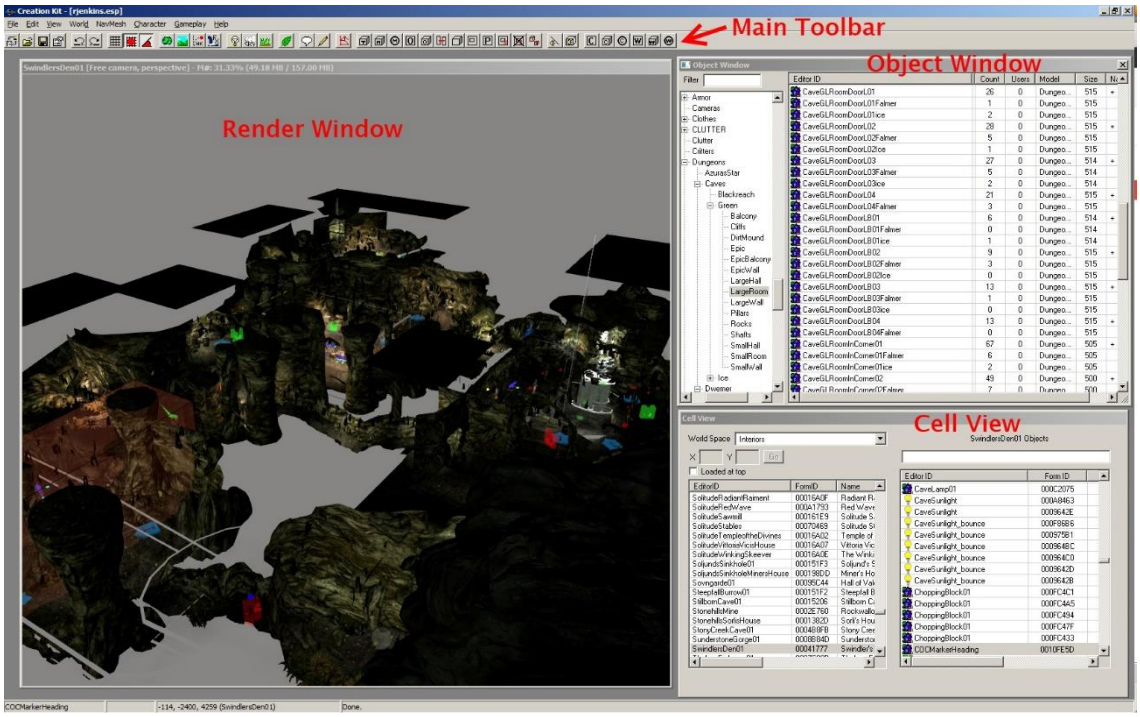
Peliä ohjelmoitaessa yritetään hahmoista, esineistä, asioista ja tapahtumista löytää yhteneväisyyksiä, paloitella ne omiin osiin ja saada niistä mahdollisimman uudelleenkäytettäviä. Tekemällä eri ominaisuuksista uudelleenkäytettäviä säästyy ohjelmoijilta työaikaa suunnittelu, toteutus ja korjaus vaiheissa. Liiallinen yhteneväisyyksien käyttö tekee pelin ominaisuuksista kuitenkin suppeita. Tätä koitetaan parantaa tekemällä yllättäviä käänteitä jo olemassa oleviin käytäntöihin. Esimerkiksi oven avaamisen tapahtumasarja voi pelissä olla seuraavanlainen: Pelaaja kulkee lähelle ovea ja painaa ovenavausnappia, minkä jälkeen ovi aukeaa. Jos jokainen ovi toimisi täysin samalla tavalla, tulisi oven avaamisesta tylsää rutiinia. Käyttämällä komentosarjaohjelmointia apuna, voi oven avaamisesta tehdä mielekästä laittamalla oveen esimerkiksi lukon, joka aukeaa vain, jos pelaajalla on avain mukanaan. Komentosarjat ovat siis apuvälineitä, joilla peruslogiikan

voi muuttaa. Komentosarjakieliä on useita, joista suurin osa kirjoitettavia, mutta suosituimmissa pelimoottoreissa on usein joko sisäänrakennettu tai mahdollisuus ladata liitännäisenä visuaalinen komentosarjaohjelmointityökalu. Visuaalinen komentosarjaohjelmointi mahdollistaa artistien ja suunnittelijoiden tekevän muutoksia pelin logiikkaan vain tietyille asioille, jolloin mahdollisten virheiden vaikutukset ovat vain paikallisia.

3 SISÄLLÖNLUONTITYÖKALUT JULKAISTUISSA PELEISSÄ

Monessa pelissä voi pelaaja luoda omia kenttiä tai kampanjoita, tai muuta sisältöä ja mahdollisesti myös jakaa sitä muille. Pelien sisällön muokkaaminen eli modifiointi on ollut olemassa käsitteenä jo pitkään, sillä vuonna 1981 julkaistu Munchkin-peli sisälsi yksinkertaisen kenttäeditorin (Wikipedia 2006).

Sisällön muokkaus siirtyi pelkistä kentistä myös muun tyyppisiin sisältöihin, kun kovakoodatun sisällön sijasta siirryttiin käyttämään dynaamista sisältöä. Nykyään kehittäjät voivat julkaista eri laajuisia työkaluja aina yksinkertaisesta kenttäeditorista muokkausohjelmistoon, jolla koko peli on tehty. Esimerkiksi The Elder Scrolls V Skyrim -pelin mukana tulevalla Creation Kit -työkalulla voi peliin lisätä mm. omia loitsuja, tasoja, vihollisia sekä tarinoita.



Kuva 2. The Elder Scrolls V Skyrim -pelin modifiointityökalu (Creation Kit 2011).

3.1 Modifioinnin eri tasot

Liitännäissisältö lisätään osaksi pelin jo olemassa olevaa sisältöä. Tämä voi tarkoittaa esimerkiksi uutta kenttää, tai esinettä johon pelaaja voi törmätä pelatessaan peliä. Usein pelinkehittäjien omat maksulliset ladattavat sisältömodifikaatiot ovat liitännäissisältöä, koska ne eivät muokkaa jo olemassa olevan sisällön käyttökokemusta suoraan.

Uudistusmodifikaatio vaihtaa pelin mekaniikkoja tai grafiikkaa huomattavasti. Uudistusmodifikaatio pyrkii parantamaan tai uudelleenluomaan tietyn osa-alueen pelistä. Esimerkki tällaisesta modifikaatiosta on pelin kaikkien aseiden vahinkoarvon lisääminen, jolloin väistelyn ja piilossa pysymisen merkitys lisääntyy.

Konversiomodifikaatio vaihtaa alkuperäisen pelin mekaniikat tai grafiikat kokonaan. Suosittuja konversiomodifikaatiota ovat mm. Defence of The Ancients ja Counter Strike, joista molemmista on tehty jatko-osia.

3.2 Jälkikäteen lisätyn sisällön hyöty

Jos peli on ohjelmoitu niin että modifiointi on mahdollista, voi kehittäjä tehdä helpommin jälkikäteen lisättävää ladattavaa sisältöä ja myydä sitä. Tällainen ylimääräinen ladattava sisältö on riskittömämpää kuin uuden pelin tekeminen, mutta riippuen pelaajakunnan koosta, voi se silti tuottaa huomattavia summia.

Jos kehittäjä julkaisee työkalunsa julkiseen käyttöön, voivat pelaajat tehdä omia modifikaatiota, jotka lisäävät pelin pitkäikäisyyttä. Käyttäjät voivat myös korjata ongelmia pelissä kehittäjien puolesta.

4 PELIMOOTTORIEN ULKOPUOLISET TYÖKALUT

Blender-3d-mallinnustyökalu ja Adobe Photoshop -kuvanpiirtotyökalu eivät kuulu mihinkään pelimoottoriin, mutta niistä ulos vietyä sisältöä voi käyttää suoraan esimerkiksi Unity-pelimoottorissa vetämällä tiedostot moottorin sisältö kansioon. Tällaisia pelimoottorista riippumattomia työkaluja on monia, mutta riippuen moottorista, ei jokaista sisältötyyppiä vastaa yhtään ulkopuolista työkalua.

Jos sisältötyyppiä ei suoraan tueta pelimoottorissa, on suosituimpiin moottoreihin saatavilla usein liitännäinen joka lataa sisällön käytettävään muotoon. Omaa pelimoottoria tehtäessä on sisällön tuominen ohjelmoitava itse, mikä voi sisältötyypistä riippuen viedä pitkäänkin. Esimerkiksi jos pelimoottorin äänisysteemi vaatii sisällön pakkaamattomassa muodossa, mutta pelitiedostojen kokonaiskoon pienentämiseksi on äänitiedostot pakattu, pitää tiedostot purkaa ennen niiden käyttöä.

4.1 Itsenäisten työkalujen puute

Kaupallisten pelimoottorien sisällön tuottaminen on usein suunniteltu niin, että niille työkaluille jotka pelimoottorista puuttuvat on vastike markkinoilla. Omaa pelimoottoria rakentaessa pitääkin ottaa huomioon, mitä kaikkia työkaluja voi tai kannattaa jättää tekemättä. Kaikkeen sisältötyyppiin ei kuitenkaan ole olemassa jo valmista työkalua, varsinkin jos sisältötyyppi on pelinkehittäjien itsensä tekemä. Tilannetta pahentaa se, että markkinoilla oleviin kaupallisiin pelimoottoreihin työkalujen tuottaminen on todennäköisesti kannattavampaa kuin yleisen rajapinnan kehittäminen (Conditt 2014).

4.2 Pelimoottorien sisäisten työkalujen käyttö ulkopuolisessa ohjelmassa

Kaupallisissa pelimoottoreissa on usein paljon sisäänrakennettuja työkaluja. Nämä työkalut ovat tehty yleensä käyttäen WYSIWYG-periaatetta, eli työkalussa on jonkinlainen esikatseluruutu, jossa lopputulos näkyy lähes samanlaisena kuin se olisi itse pelissä. Jos sisällön saa vietyä moottorista ulos niin että sitä voi käyttää myös ulkopuolisessa ohjelmassa, voi hyvin hiotuista pelimoottoreiden työkaluista olla hyötyä myös omassa moottorissa. Esimerkiksi Unreal Engine -pelimoottorissa voi tehdä animaation ja tallentaa sen normaaliksi videotiedostoksi, jonka voi esittää oman pelimoottorin sisällä. Unity-

moottoriin voi luoda omia editoreja melko vaivattomasti ja osan sisällöstä voi viedä ulos moottorista YAML-tekstitiedostomuodossa.

4.3 Työkalujen käyttö niille ei-tarkoitetun sisällön tuottamiseen

Vaikka Blender ja 3ds Max ovat 3d-mallinnusohjelmia voi niiden avulla tehdä 3d-kenttiä. Nämä kentät eivät todennäköisesti näytä samalta kuin mitä pelin sisällä, koska ohjelmien ja pelimoottorin väliset varjostimet ja piirtotekniikat ovat erilaiset. Myös käyttäjäkokemus voi olla huonompi ohjelmissa, jotka eivät ole tarkoitettuja tietyn sisällön tuottamiseen.

Jos pelimoottorissa on tarkoitus olla luolastoroolipeleistä tunnettuja huoneita ja käytäviä, voi niitä mallintaa esimerkiksi Tiled tai Gleed-laattaeditoreilla.

Vaikeinta työkalujen väärän sisällön tuottamiseen on sisällön tuominen pelimoottoriin, sillä sen sijasta, että sisältö luettaisiin niin kuin se normaalisti luetaan, käytetään dataa tarkoituksellisesti väärin. Tämä kasvattaa mahdollisten virheiden määrää, varsinkin huonosti dokumentoiduissa tiedostoformaateissa, sillä tiedostoissa on todennäköisesti ylimääräistä dataa mikä pitää hylätä sitä lukiessa.

5 PROSEDURAALINEN JA IHMISEN LUOMA SISÄLTÖ

Sisällön tuottaminen pienellä henkilöstömäärällä vie paljon aikaa. 3d-mallin tekemiseen menee peligenrestä, teemasta ja mallin tarkkuudesta riippuen noin päivästä useampaan viikkoon. Kenttien luominen tuhansista palasista vie myös paljon aikaa.

Proseduraalisesti tuotettu sisältö on tietokoneen tekemää algoritmin mukaan tehtyä sisältöä. Jos algoritmin alkuarvoiksi asetetaan satunnaisesti valitut arvot, kutsutaan sisältöä satunnaisgeneroiduksi sisällöksi. Satunnaisgeneroitua sisältöä voidaan tuottaa nopeasti ilman ihmisen valvontaa, jolloin projektin työmäärää voidaan laskea.

5.1 Proseduraalisesti tuotettu maailma

Koska maailmojen luonti on hidasta käsin, on se käytännöllinen paikka käyttää satunnaisgenerointia. Hyvin tehty proseduraalinen algoritmi voi luoda itseään lähes koskaan toistamattomia äärettömän laajoja maailmoita, kuten esimerkiksi Minecraft-pelin maailma on noin Neptunuksen pinnan kokoinen jos yksi Minecraftin yksikkö on yksi metri.

Kuva 3. Kuvakaappaus Minecraft-pelin satunnaisgeneroidusta maailmasta (alarion99 2012).

Ohjelmoitu algoritmi pitää valita oikein kyseiseen tehtävään, jotta pelimaailmasta tulee immersiivinen, eikä maailma näytä selvästi tietokoneen luomalta.

Satunnaisgeneroidussa maailmassa jokainen pelikerta on erilainen. On kuitenkin vaikeaa ja työlästä tehdä algoritmia joka tekisi maailmasta reilun, tarpeeksi haasteellisen ja samaan aikaan mielenkiintoisen. Ohjelmoitaessa tulee olla myös tarkka, ettei peliä rikkovia tilanteita pääse tapahtumaan, kuten pelin maali on seinän sisällä tai pelaaja pääsee ulos pelialueelta (Green 2016, 16). Koska satunnaisgeneraattorin luoma kenttä on joka kerta erilainen, on ongelmatapauksia vaikea löytää ennen pelin testauttamista suurella yleisöllä. Vain muutaman testin jälkeen ei ongelmatapausta välttämättä syntynyt, jos sen ilmenemismahdollisuus on vain joka sadannes kenttä.

Satunnaisgeneroitu sisältö ei sovellu jokaiseen peliin. Esimerkiksi mitä kilpailuhenkisempi peli, sen vähemmän satunnaisgenerointi yleisesti ottaen siihen sopii, sillä satunnaisuus luo epäreiluutta alkuasetelmissa. Satunnaisgeneroidussa sisällössä ei pelin kehittäjällä ole enää absoluuttista hallintaa vaan pelaajan pelikokemus on algoritmin varassa.

5.2 Ihmisen luoma kenttä

Jos kentän sisäistä jaksotusta halutaan hallita täysin, on kentän oltava ihmisen luoma. Jaksotus on tärkeää pelin hauskuuden kannalta, eikä kaikkea sisältöä kannata näyttää ensimmäisessä kentässä (Ryan 1999).

Uudelleenpeluuarvo on kuitenkin käsin tehdyssä kentässä pienempi kuin satunnaisgeneroidussa, sillä kun pelaaja on kerran mennyt kentän läpi ja nähnyt kaiken kentän sisällön, ei saman kentän uudelleen pelaaminen ole yhtä mielenkiintoista. Käsin tehdyissä kentissä voidaan kuitenkin ottaa huomioon edelliset sekä tulevat kentät ja tätä voidaan käyttää jaksottamisen apuna, kuten esimerkiksi intensiivisen räiskintäkentän jälkeinen hidas tarinavetoinen kenttä.

5.3 Modulaarisuus ihmisen luomassa sisällössä

Täysin ihmisen ja täysin proseduraalisen sisällön välimaastossa on moduuleista proseduraalisesti rakennettu sisältö. Moduulit ovat käsin tehtyä sisältöä jotka sopivat toisiinsa kiinni tietyn säännönmukaisuuksin. Esimerkiksi Diablo 2 -pelissä hahmojen ylle puettavat vaatteet voitiin vaihtaa toisiin ilman että hahmon 3d-mallia tarvitsi vaihtaa ja pelin kentät ovat myös satunnaisgeneroitu moduuleista.

Moduuleissa on mahdollista tehdä mielenkiintoisia ratkaisuja moduulin sisällä, ilman että se rikkoo ympäröiviä moduuleita. Kenttämoduulissa voisi olla esimerkiksi normaalia vaikeammin kukistettavissa oleva vihollinen joka vartioi arkkua. Tämä vihollinen ei kävele moduulin ulkopuolelle vaan pelaajan on tapeltava tietyllä alueella, jolloin pelaaja ei vahingossa tee muista moduuleista vaikeampia kuljettamalla vaikean vihollisen toisen moduulin vihollisten lähelle.

Moduuleita satunnaisesti yhdistelemällä saadaan osa proseduraalisen sisällön uudelleenpeluuarvosta, mutta kuitenkin pitäen osa sisällön hallinnasta kehittäjien käsissä. Moduuleiden yhdistäminen vaatii hyvin suunnitellut käsin tehdyt visuaaliset sisällöt, sekä hyvin ohjelmoidut säännöt, ettei moduulien rajat ole liian helposti huomattavissa. Jos moduulit näyttävät selvästi eroavan toisistaan, laskee pelaajan immersio välittömästi (Schaefer 2000).

6 TYÖKALUOHJELMA OMAAN PELIMOOTTORIIN JA GODSBANE PELIIN

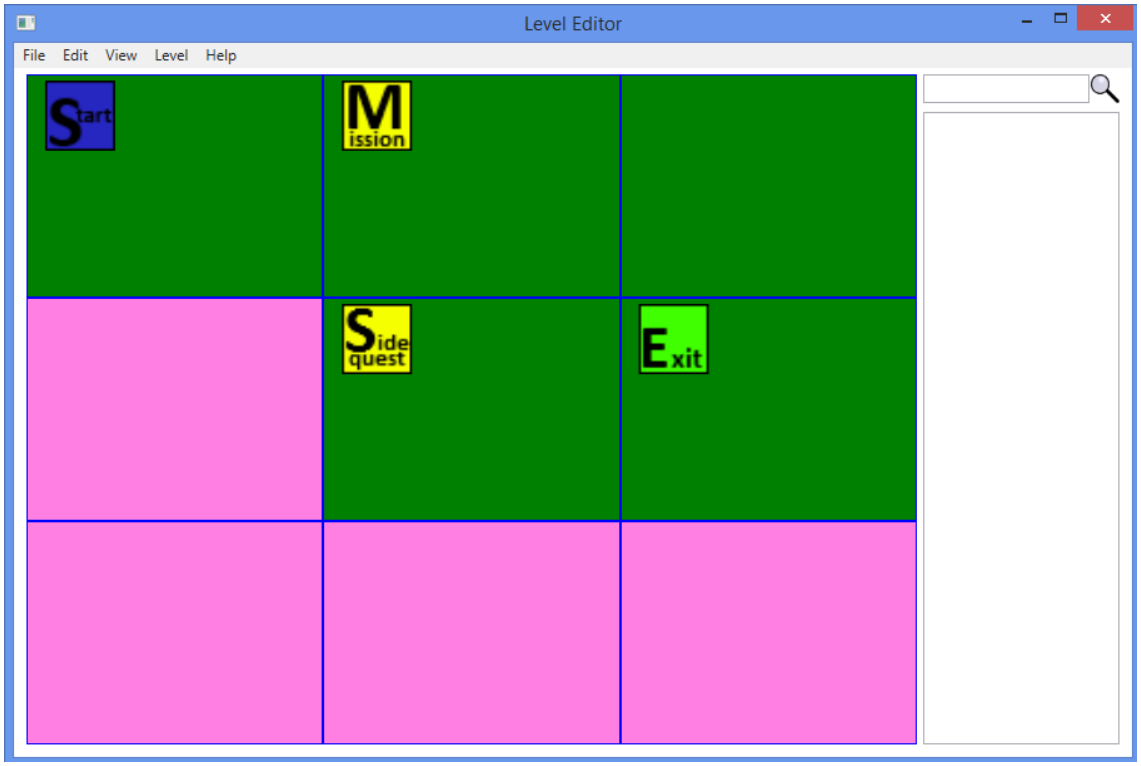
Godsbane on Rockodile Games -yrityksen keskeytetty peliprojekti, jonka tekeminen aloitettiin loppuvuodesta 2015. Pelissä pelaaja räiskii ylhäältäpäin kuvatussa fantasiamaailmassa käyttäen taikavoimia hyödykseen. Pelissä on mahdollista pelata 3 muun pelaajan kanssa internetin välityksellä tai samalta koneelta. Peli jäädytettiin lähes vuoden työpanoksen jälkeen kustannusten jatkaessa kasvuaan ja yleisön kiinnostuksen pelille ollessa olematon.

Godsbane alkoi huomattavasti eri projektina, kuin mihin se päätyi loppujen lopuksi. Peli suunniteltiin alun perin Unity-moottorilla tehtäväksi, mutta nopean stressitestin jälkeen ei moottorin suorituskyky riittänyt haluttujen graafisten efektien piirtämiseksi. Yritimme käyttää Unreal Engine -pelimoottoria, mutta sen sisältämän editorin laitevaatimukset olivat liian suuret. Päätimme loppujen lopuksi tehdä projektiin oman pelimoottorin. Koska oma pelimoottori tarvitsi omia tietorakenteita, päätimme myös tehdä moottorille oman editorin Microsoft WPF -työkaluilla. Suurin virhearvio editoreita tehtäessä oli oletus siitä, että kun peli on ylhäältä päin kuvattu, ei siinä tarvitse muokata peliobjekteja kuin toisessa ulottuvuudessa, jolloin WPF-työkalut riittäisivät.

6.1 Kenttä- ja moduulieditori

Moduulieditorilla voi tehdä kenttämoduuleita kentän satunnaisgenerointia varten. Yksi moduuli koostuu 42 kertaa 42 ruudukosta, jossa jokaisessa ruudussa voi olla määriteltynä lattiagrafiikka, tieto siitä onko ruutu osa seinägrafiikkaa vai ei, sekä fyysinen informaatio ruudulle, eli voiko ruudun yli kävellä ja jos voi, millä nopeudella. Ruutujen lisäksi moduulissa voi olla missä tahansa pisteessä, kuten ruutujen välissä, peliobjekteja, esimerkiksi pieniä kiviä, vihollisia, ansoja, kirstuja jne. Moduuliin merkitään, kuinka vaikea se on läpäistä, sekä missä suunnassa moduulissa on seinät, jotta satunnaisgeneraattori ei laita vahingossa moduulia joka ei sovi tiettyyn kohtaan.

Kenttäeditorissa merkitään kentän muoto moduuleina, sekä kiinnostavat paikat kentässä, jotka määrittävät mitkä moduulit mihinkin kohtaan kenttää voidaan muodostaa.



Kuva 4 Kenttäeditori, jossa yleisimmät kiinnostavat paikat merkittynä

Kenttään lisätään hiiren vasemmalla näpäytyksellä moduulille paikka ja hiiren oikealla näppäimellä avataan valikko, josta voidaan valita jokin kiinnostava paikka, kuten pelaajan aloitusmoduuli. Kenttään voidaan määritellä arvio siitä, kuinka pitkä se on. Tätä tietoa käytetään laskemaan kampanjan pituutta. Kenttään merkitään lopuksi, onko kyseessä tavallinen kenttä, joka sisällytetään normaaliin satunnaisgenerointiin mukaan, vai onko kyseessä erikoiskenttä joka pitää käsin laittaa kampanjaan.

Pelikentän käynnistyessä ja moduulien paikkojen vaihtuessa huomattiin, että moduuliin valmiiksi sijoitetut seinät eivät ole hyvän näköisiä koska moduulien rajat tulevat liian selvästi näkyviin. Tätä ongelmaa varten kehitimme algoritmin, joka leikkaa kaikki moduulin ruudut, joissa on seinää, mahdollisimman isoihin osiin. Leikkauksen jälkeen voidaan luoda jokaista osaa kohden oikean kokoinen seinäpala. Koska seinäpalat luodaan moduulien paikoilleenlaiton jälkeen, pystytään moduulien reunat peittämään paloilla jotka jatkuvat moduulista toiseen, esimerkiksi 4 x 4 kokoinen seinä palanen voidaan laittaa moduulin reunaan, josta se jatkuu seuraavaan moduuliin ilman että niiden leikkauskohdassa on selkeä aukko.

6.2 Kampanjaeditori

Kampanjat ovat kenttäkokoelmia jotka määrittävät missä järjestyksessä kentät pelataan. Kampanjat ovat eroteltu viiteen eri vaikeustasoon, mutta vaikeustaso ei ole absoluuttinen, vaan kehittäjän tulisi miettiä kaikkien kenttien keskimääräinen vaikeus. Pelissä kampanjalle annetaan keskimääräinen läpäisyaika, joka voidaan näyttää pelaajalle ennen kampanjan aloittamista, jolloin pelaaja osaa valmistautua joko pitkään koetukseen tai lyhyeen otteluun.

Kampanjan kenttälistaan voidaan laittaa jokin tietty tai satunnaisesti valittu taso. Näin vaikka pelaaja pelaisi saman kampanjan, on siinä uusia tasoja, joissa on uusia moduuleita edelliseen pelikertaan verrattuna.

6.3 Toiminto- ja taitoeditori

Toimintoeditorissa lisätään peliin eri kovakoodattujen toimintojen säädettävät arvot. Esimerkiksi TuotaVahinkoa toiminto vähentää toiminnon kohteelta osumapisteitä niin paljon kuin toimintoon on merkitty.

Koko toimintoeditori olisi voinut jättää pois ja kaiken tiedon olisi voinut lisätä suoraan toimintoja vaativiin paikkoihin, kuten taitoeditoriin. Järjestelmästä tulee huomattavasti hankalampikäyttöisempi mitä enemmän eri tiedostoihin joudutaan viittaamaan. Toimintoja harvoin käytetään useassa paikassa samoilla arvoilla, joten lähes jokaista tarvittavaa toimintoa varten joudutaan työkalussa tekemään uusi toiminto, joka tarkoittaa uuden tiedoston luomista.

Taitoeditorissa yksi tai useampi toiminto yhdistetään yhdeksi taidoksi, jolla on minimiodotusaika käyttöjen välillä sekä toimintojen aktivointiin vaikuttavia ehtoja. Esimerkiksi taitoeditorilla voi tehdä taidon, joka puolet aktivointikerroista tekee vahinkoa käyttäjään itseensä, tai puolet ajasta vahinkoa lähimpään viholliseen.

6.4 Toimijaeditori

Toimijaeditorissa lisätään toimijoille yleiset tiedot, kuten nimi, toimijan tyyppi, osumapisteiden määrä, käytettävä päättelylista, yms. Editorilla tehdään kaikki kivistä vihollisiin ja

pelaajiin. Toimijalle määritetään fysiikkatörmäyttimen muoto ja koko sekä muut törmäyttimet, joihin toimija voi törmätä pelissä. 3d-mallit lisätään listaan, josta editori lukee listan ensimmäisen nimen ja yrittää löytää sillä nimellä esikatselukuvan editorin kansioista. Kuvan löydyttyä sitä käytetään kaikkialla missä toimijoihin viitataan, kuten moduuleissa ja toiminnoissa. Toimijoille voidaan lisätä partikkelisysteemeitä, joita tehdään PopcornFX -partikkelieditorilla.

Kuva 5 Godsbane kuva kentästä jossa kolme pelaajaa yrittää väistää vihollisten ammuksia. Ammukset ovat toimijoita, joissa on partikkelisysteemi kiinni.

6.5 Blender-mallinnusohjelman Python-liitännäinen

Lyhytnäköisyyden vuoksi oli moduulieditori suunniteltu näyttämään vain 2d-grafiikkaa, vaikka peli sisältääkin 3d-grafiikkaa. Editoria käyttäessä huomattiin pian, ettei 2d-esitysmuoto riittänyt kaikkiin tarvittaviin toimintoihin. Ongelman korjaamiseksi kehitimme Blender-3d-mallinnusohjelmaan liitännäisen, joka osaa lukea moduulitiedoston, rakentaa seinät, maan, sekä toimijat moduulin pohjalta, ja joka pystyy kirjoittamaan muutokset takaisin moduulitiedostoon. Liitännäinen jäi vajavaiseksi, eikä esimerkiksi hyödyllinen controlz näppäinyhdistelmä toiminut oikein, vaan se vuotaa muistia ladattujen grafiikkojen verran, aiheuttaen usein ohjelman kaatumisen.

6.6 Käyttäytymiseditori

Käyttäytymiseditorissa määritellään, onko toimijalla vain reaktiivinen tekoäly vai käyttääkö se sen lisäksi päättelylistaa. Käyttäytymiseditorissa määritellään myös, onko toimija vihollisen vai pelaajan puolella, vai ei kenenkään puolella.

Käyttäytymiseditorissa voi myös määrittää pyrkiikö tekoäly pitämään välimatkaa kohteestaan vai ei. Editorissa määritellään myös, miten tekoäly yrittää liikkua haluamaansa paikkaan. Liikkumistapavaihtoehdot ovat suoraan kohdetta päin liikkuminen, sekä reitinetsintä A* algoritmilla.

Vihollisten tärkein osa-alue päättelylista sisältää kohteen etsinnän, yhden tai useamman ehtolauseen, liikkumistapahtuman jos ehto toteutuu, sekä taidon, jota vihollinen käyttää, jos se on mahdollista. Työkalulla voidaan esimerkiksi laittaa vihollinen liikkumaan pelaajaa kohti, ampumaan pelaajaa, käyttämään osumapisteitä parantavaa taitoa, jos osumapisteet laskevat alle 50 %, parantamaan muita vihollisia yms.

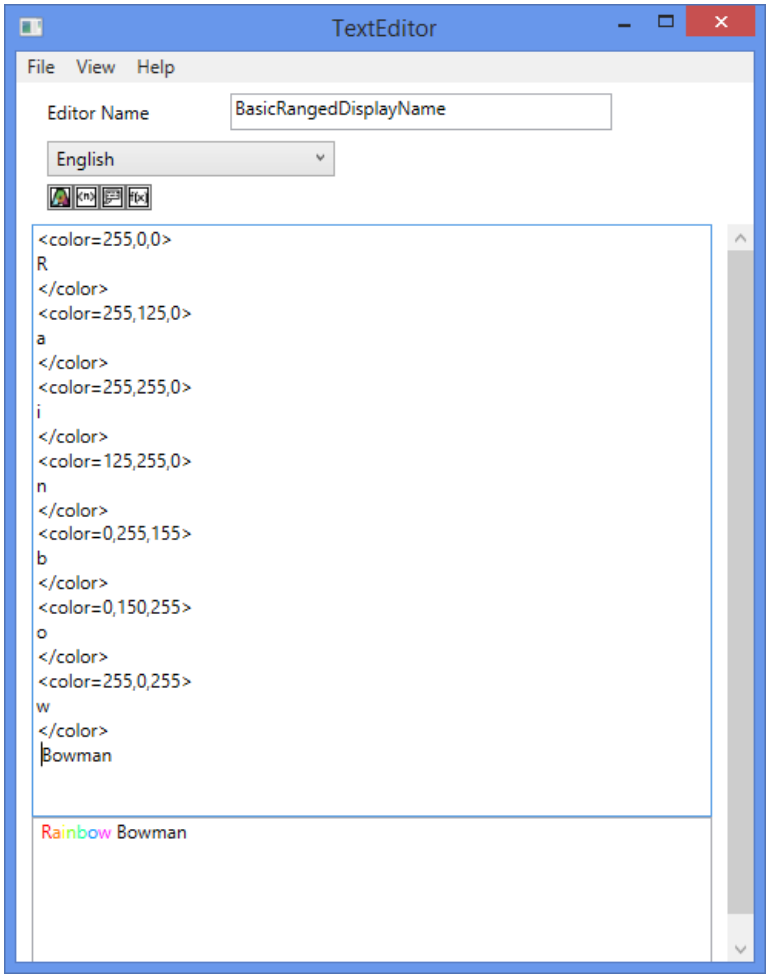
Editorissa on myös mahdollista laittaa tapahtumia tapahtumaan tietyistä syistä, kuten esimerkiksi jos vihollinen kuolee, voi se kuollessaan räjähtää tehden vahinkoa lähellä oleviin toimijoihin.

6.7 Vaikutuseditori

Vaikutuseditorissa voi tehdä toistuvia tapahtumia tai hetkellisesti voimassa olevia hahmon tilan muutoksia kuten liikkumisnopeuden hidastus, vahinkokertoimen nostamisen yms. Vaikutuseditorin avulla voi tehdä esimerkiksi myrkytysefektin hahmoihin, jolloin ne menettävät osumapisteitä vähän väliä.

6.8 Tekstieditori

Tekstieditorissa voi kirjoittaa käännettyjä tekstejä. Tekstiin saa myös lisättyä mukaan äänitiedoston, joka voidaan soittaa dialogia näytettäessä pelaajalle. Tekstieditori on ainut WYSIWYG editori, jossa on tekstinsyöttökentän lisäksi esikatselukenttä, johon syötetty teksti tulee lopullisen näköisenä. Useammassa editorissa olisi ollut suotavaa käyttää jonkin näköistä esikatselutapoja.



Kuva 6 Tekstieditori ja erivärinen teksti

Tekstieditori tukee tekstin värin vaihtamisen, rivinvaihdon, dialogi äänitiedoston sekä dynaamisen muuttujan lisäämistä.

6.9 Sisällön tallentaminen

Editoriohjelma on tehty niin ettei eri ikkunoiden sulkeminen poista tietoja muistista, vaan ne pystytään tallentamaan vielä kunkin editorin sulkemisen jälkeen. Koko ohjelmaa sulkiessa jokaisen editorin hallintaluokka katsoo, onko editoitavaan tiedostoon tehty muutoksia, ja jos on, ne ilmoittavat siitä käyttäjälle kysymällä, haluaako käyttäjä tallentaa tiedostot. Useassa editorissa on oma logiikkansa toimintojen kumoamiselle, koska WPF -työkalut tarjoavat sen vain kirjoitettuun tekstiin.

Päätimme käyttää tallentamiseen omaa tekstimuotoa, joka muistuttaa XML-muotoilua. Omassa tallennusmuodossamme jokaista sisällön datakenttää vastaa oma <tunniste>data</tunniste> tekstipätkä, jossa sana tunniste korvataan datakentän nimellä, ja datan tilalle kirjoitetaan datakentästä luettu arvo. Teimme oman sisällönkirjoitusmuodon, koska halusimme pitää mahdollisuuden optimointiin avoinna.

Ohjelmistosta pelimoottoriin sisältö viedään pakattuna ja salattuna ZIP-pakettina. ZIP-paketti on salattu yksinkertaisella eksklusiivinen disjunktio (XOR) -menetelmällä jossa paketin jokainen tavu käännetään avainta vasten

```
dataOutput[i + 0] = (byte)(encryptionKey[0] ^ dataInput[i + 0]);
dataOutput[i + 1] = (byte)(encryptionKey[1] ^ dataInput[i + 1]);
dataOutput[i + 2] = (byte)(encryptionKey[2] ^ dataInput[i + 2]);
dataOutput[i + 3] = (byte)(encryptionKey[3] ^ dataInput[i + 3]);
dataOutput[i + 4] = (byte)(encryptionKey[4] ^ dataInput[i + 4]);
dataOutput[i + 5] = (byte)(encryptionKey[5] ^ dataInput[i + 5]);
dataOutput[i + 6] = (byte)(encryptionKey[6] ^ dataInput[i + 6]);
dataOutput[i + 7] = (byte)(encryptionKey[7] ^ dataInput[i + 7]);
```

Kuva 7 Sisään tuleva data salataan eksklusiivisen disjunktion avulla

Tämä estää kaikista yksinkertaisimman käyttäjän tiedostojen palauttamisen yrittämisen ja on erittäin nopea toimenpide. Pelimoottorissa on sama salausavain ja algoritmi, jota kutsutaan pakettia avattaessa. Lopputulos alkuperäinen ZIP-paketti, koska kahden peräkkäisen eksklusiivisen disjunktion kutsuminen samalla operaattorilla on tulos muuttuja itse.

$$XOR(XOR(a,b),b) = a$$

Kaava 1 Kahden eksklusiivisen disjunktion peräkkäinen kutsuminen

Ohjelmisto tutkii sisältötiedostojen tilaa jatkuvasti muutosten varalta. Esimerkiksi jos Python-liitännäinen on kirjoittanut uutta tietoa kenttätiedostoon, ja kenttäeditori on päällä, osaa ohjelma kysyä käyttäjältä haluaako hän lukea tapahtuneet muutokset levyltä.

6.10 Satunnaisesti valitun sisällön lisääminen jälkikäteen

Pelimoottorissa oli tarkoitus, että jokainen sisältöpaketti, joka sisällytetään pelin mukana, voidaan aktivoida ja deaktivoida haluttaessa, jolloin satunnaisgeneraattorille näkyvän sisällön määrää voi helposti lisätä ja poistaa. Tämä mahdollistaa sisällön lisäämisen jälkikäteen niin käyttäjältä kuin kehittäjältä, joka auttaa modifikaatioyhteisön synnyssä sekä ladattavan sisällön tekemisessä. Kun uuden paketin tiedot luetaan, lisätään kaikki moduulit, kentät, kampanjat ja toimijat satunnaisgeneraattorin listoihin, jolloin kun listasta pitää valita moduuli, voi valituksi tulla uusi tai vanha moduuli ilman että vanha sisältö menisi rikki.

7 YHTEENVETO

Opinnäytetyön tarkoituksena oli luoda omaan pelimoottoriin sisällöntuottotyökalut, niin että sisällön luominen peliin sen julkaisun jälkeen oli mahdollisimman vaivatonta. Työssä kiinnitettiin huomiota sisällön uudelleenkäyttöön satunnaistuoton avulla, mikä olisi voinut kasvattaa pelattavan sisällön määrää, jos projekti olisi jatkunut pidemmälle. Ohjelmiston käytettävyyttä olisi voinut parantaa entisestään, varsinkin Python-liitännäinen olisi voinut toimia varmemmin, sillä ohjelmien kaatuilu vie paljon aikaa. Projektin lopputulos oli monipelattavan räiskintäpelimoottorin teknologiademo ja moottorin sisällönluontityökalut.

Työssä opittiin paljon ohjelmistokehityksestä, pelimoottorin ja pelin suunnittelusta sekä toteuttamisesta. Työn aikana opittiin, että nopea iterointi ja ideoiden nopeampi hylkääminen voi säästää paljon aikaa. Työssä opittiin, että itse tehdyt työkalut voivat olla hyödyllisiä tehdä, mutta ne kannattaa tehdä jo olemassa olevan pelimoottorin rinnalle. Uuden moottorin tekeminen ei tule olemaan todennäköistä vielä vuosiin. Työssä opittua on käytetty hyödyksi mm. Rockodile Gamesin Hear no Evil -projektissa, sekä Unity-moottorilla tehdyissä asiakasprojekteissa, joissa on tarvittu omia työkaluja.

LÄHTEET

Shaker, N.; Togelius, J. & Nelson, M. 2016. Procedural Content Generation in Games. Springer International Publishing.

Alarion99 2012. beautiful seed. Floating mountain, jungle temple, emeralds. Viitattu 31.5.2018 https://www.minecraftforum.net/forums/minecraft-java-edition/seeds/318821-1-4-2-beautiful-seed-floating-mountain-jungle

Green, D. 2016. Procedural Content Generation for C++ Game Development. Packt Publishing Ltd.

Conditt, J. 2014. Unity Asset Store top sellers earn $30K a month. Viitattu 27.5.2018 https://www.engadget.com/2014/11/05/unity-asset-store-top-sellers-earn-30k-a-month

Luton, W. 2009. Making Better Games Through Iteration. Viitattu 26.5.2018 https://www.gamasutra.com/view/feature/132554/making_better_games_through_.php?print=1

Shaefer, E. 2000. Postmortem: Blizzard's Diablo II. Viitattu 27.5.2018 https://www.gamasutra.com/view/feature/131533/postmortem_blizzards_diablo_ii.php?print=1

Ryan, T. 1999. Beginning Level Design Part 2: Rules to Design By and Parting Advice. Viitattu 26.5.2018 https://www.gamasutra.com/view/feature/131739/beginning_level_design_part_2_.php?print=1

Wikipedia 2004. Munchkin (video game). Viitattu 26.5.2018 https://en.wikipedia.org/wiki/Munchkin_(video_game)

Wikipedia 2003. BMP file format. Viitattu 26.5.2018 https://en.wikipedia.org/wiki/BMP_file_format

Creation Kit 2011. Blank Workspace. Viitattu 31.5.2018 https://www.creationkit.com/images/6/6b/BlankWorkspace.jpg