Toni Räsänen

Proseduraalinen sisällön generointi

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Tradenomi
Tietojenkäsittely
16.5.2011

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Luonnontieteiden ala	Koulutusohjelma Tietojenkäsittelyn koulutusohjelma
Tekijä(t) Toni Räsänen	
Työn nimi Proseduraalinen sisällön generointi	
Vaihtoehtoiset ammattiopinnot Peliohjelmointi	Ohjaaja(t) Janne Koponen
	Toimeksiantaja
Aika 16.5.2011	Sivumäärä ja liitteet 46

Tässä opinnäytetyössä kerrotaan proseduraalisesta sisällön generoinnista, erityisesti videopelien näkökulmasta. Aluksi kerrotaan, että mitä proseduraalinen generointi yleensäkin on ja siitä siirrytään itse proseduraaliseen sisältöön. Proseduraalisesta generoinnista esitellään kaksi algoritmia: keskipisteen siirto algoritmi ja Perlin-kohina.

Proseduraalisen sisällön generoinnin erilaisia tyyppejä tutkitaan ja esitellään esimerkkien avulla joista joitakin esimerkkejä tutkitaan tarkemmin. Proseduraalisesta sisällön generoinnista esitellään myös kaksi algoritmia: luolaston generointi ja rytmipohjainen kentän generointi. Rytmipohjaista kentän generointia on myös käytetty tämän työn toiminnallisessa osuudessa.

Opinnäytetyön toiminnallisessa osassa tehtiin tasoloikkapeli yhdessä muutaman muun opiskelijan kanssa. Tämän pelin tärkeässä osassa oli proseduraalisesti generoitu sisältö, erityisesti proseduraalisesti generoitu kenttä. Työssä kerrotaan peliin suunnitellusta ja toteutetusta proseduraalisesta sisällöstä ja sen käyttämistä algoritmeista. Mainitun rytmipohjaisen kentän generoinnin lisäksi pelin kentissä on käytetty myös keskipisteen siirto algoritmia.

Kieli	Suomi
Asiasanat	proseduraalinen, sisältö, generointi, peli, ohjelmointi
Säilytyspaikka	☒ Verkkokirjasto Theseus ☒ Kajaanin ammattikorkeakoulun kirjasto

Kajaanin ammattikorkeakoulu
University of Applied Sciences

THESIS
ABSTRACT

School Business	Degree Programme Business Information Technology
Author(s) Toni Räsänen	
Title Procedural Content Generation	
Optional Professional Studies Game programming	Instructor(s) Janne Koponen
	Commissioned by
Date May 16th 2011	Total Number of Pages and Appendices 46

The topic of this thesis is procedural content generation, especially from the point of view of video games. The thesis begins by introducing the general concept of procedural generation. There are also two examples of procedural generation algorithms: midpoint displacement algorithm and Perlin noise.

In the procedural content generation part, the thesis will show different types of procedural content and of some of those there are examples in a form of games and algorithms. Two algorithms are introduced as examples: cave generation and rhythm-based level generation. Rhythm-based level generation is also used in the practical part of the thesis.

After procedural content generation is introduced, the thesis also studies procedural content generation in three example games. These examples are Elite, Spelunky and Dwarf Fortress.

The practical part is about designing and creating procedural content generation for a platformer game. The focus of procedural content in that game was creating a procedural level generator for the game. The game uses the aforementioned rhythm-based level generation and midpoint displacement algorithm to create levels.

Language of Thesis	Finnish
Keywords	procedural, content, generation, game, programming
Deposited at	☒ Electronic library Theseus ☒ Library of Kajaani University of Applied Sciences

SISÄLLYS

1 JOHDANTO 1

2 PROSEDURAALINEN GENEROINTI 3
2.1 Taustaa 3
2.2 Algoritmeja 4
2.2.1 Keskipisteen siirto algoritmi 5
2.2.2 Perlin-kohina 6

3 PROSEDURAALINEN SISÄLLÖN GENEROINTI 8
3.1 Taustaa 8
3.2 Algoritmeja 20
3.3 Hyödyt ja haitat 23

4 ESIMERKIT 26
4.1 Elite 26
4.2 Spelunky 27
4.3 Slaves to Armok: God of Blood Chapter II: Dwarf Fortress 28

5 PROJEKTI 33
5.1 Proseduraalinen sisältö 34
5.1.1 Kenttä 35
5.1.2 Viholliset 37
5.1.3 Esineet 37
5.2 Pelin algoritmit 38

6 POHDINTA 40

LÄHTEET 44

SYMBOLILUETTELO

Demoscene	Digitaalisen taiteen alakulttuuri jossa tekijät luovat niin kutsuttuja demoja. Demot ovat reaaliaikaisia audiovisuaalisia esityksiä.
Iteraatio	Iteroinnin yksi vaihe. Iterointi on työvaiheen toistoa niin kauan, kun haluttu tulos on saavutettu.
Korkeuskartta	Heightmap. Korkeuskartalla kuvataan alueen korkeussuhteita. Sen avulla voi esimerkiksi peleihin luoda pelialueita.
Mockup	Peleissä yleensä graafinen luonnos siitä miltä valmiin pelin pitäisi näyttää.
Pseudokoodi	Algoritmin perusrakennetta kuvaavaa oikean ohjelmointikielen tapaista koodia.
Riippumaton pelintekijä	Pelintekijä joka kehittää pelejä ilman ulkopuolisen julkaisijan rahallista tukea.
Roguelike	Tietokoneroolipelien alagenre. Sen erityispiirteitä ovat satunnaisuus, uudelleenpelattavuus, lopullinen kuolema ja vuoropohjainen liikkuminen. Nimi juontuu vuonna 1980 julkaistusta pelistä Rogue.

1 JOHDANTO

Pelit, joissa käytetään proseduraalista sisältöä, ovat pitkään kiinnostaneet minua, joten tästä kiinnostuksesta lähdin tekemään opinnäytetyötä proseduraalisesta sisällön generoinnista. Proseduraalinen sisällön generointi on ollut mukana tietokonepelien kehityksessä niiden alku ajoista lähtien, mutta sitä ei kuitenkaan ole käytetty erityisen paljoa ja se on enimmäkseen pysynyt samantyylisten pelien valttikorttina. Viime aikoina kuitenkin erityisesti riippumattomat pelifirmat ja pelinkehittäjät ovat alkaneet tehdä pelejä jotka hyödyntävät proseduraalista sisällön generointia. Proseduraalisen sisällön avulla he ovat helposti saaneet peleihinsä paljon erilaista sisältöä.

Proseduraalista generointia on käytetty myös muihin kuin peleihin. Esimerkiksi demoscene on hyödyntänyt tehokkaasti proseduraalista generointia ja sitä on käytetty vakavissakin ohjelmissa, kuten kaupunkisuunnittelun apuna.

Proseduraalinen sisällön generointi on proseduraaliselle generoinnille jatkumoa sillä peleihin sisältöä generoidaan käyttäen samoja algoritmeja. Proseduraalisella sisällön generoinnilla tarkoitetaan pelillisesti tärkeän sisällön, kuten kenttien ja juonen, generointia. Proseduraalista generointia voidaan käyttää peleissä muuhunkin, vähemmän pelillisesti tärkeisiin asioihin, kuten tekstuureihin ja animaatioihin.

Tässä työssä keskitytään proseduraaliseen sisällön generointiin, mutta aluksi käydään läpi myös mitä itse proseduraalinen generointi on. Proseduraalista sisältöä esitellään tarkemmin tutustuen siihen miten proseduraalista sisältöä esiintyy peleissä. Näihin asioihin ovat tarkemmin tutustuneet Andrew Doull sekä Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley ja Cameron Browne.

Pelejä, ja joissakin tapauksissa myös algoritmeja, käytetään esimerkkinä eri proseduraalisen sisällön generoinnin kategorioista. Niiden avulla havainnollistetaan kuinka kyseinen proseduraalinen sisältö voi esiintyä pelissä. Proseduraalisen sisällön generointia on myös tutkittu tarkemmin kolmen pelin kohdalla, joiden erilaisista proseduraalisista sisällöistä kerrotaan yksityiskohtaisemmin.

Opinnäytetyön toiminnallisessa osassa toteutettiin yhdessä muutaman muun opiskelijan kanssa vahvasti proseduraaliseen generointiin nojaava peli. Peli on toimintatasoloikka Mega

Man -pelien tyyliin, mutta siinä on vain yksi kenttä. Kenttä on teoriassa loputon ja pelaaja kohtaa siellä enemmän tai vähemmän vihamielisiä robotteja. Pelaajan on tarkoitus päästä mahdollisimman pitkälle. Loputtoman kentän generoinnin perustana käytettiin Gillian Smithin, Mike Treanorin, Jim Whiteheadin ja Michael Mateaksen kehittämää rytmipohjaista kentän generointia. Peli oli ryhmän ensimmäinen yhteinen kunnollinen projekti joka toteutettiin alusta asti eli se aloitettiin ideoinnista ja suunnittelusta. Näiden pohjalta lähdettiin tekemään peliä.

2 PROSEDURAALINEN GENEROINTI

Proseduraalinen generointi on tapa jolla luodaan esimerkiksi grafiikkaa tai ääntä algoritmien avulla. Erityisesti tätä on käytetty peleissä ja demoscenessä, mutta sitä on käytetty myös elokuvissa ja niin sanotuissa vakavammissakin ohjelmissa. Enimmäkseen näissä on keskitytty grafiikan, äänen ja pelien kohdalla myös pelillisen sisällön luontiin. (Wikipedia 2011 b.) CityEngine on eräs vakavampi ohjelma joka käyttää proseduraalista generointia. Se on ohjelma jolla voi proseduraalisesti generoida kaupunkeja. Sitä hyödynnetään arkkitehtuurissa, arkeologiassa, kaupunkisimulaatioissa jne. CityEngineä käytetään esimerkiksi tulevan Masdar-kaupungin suunnitteluun. (Procedural, Inc. 2011.)

2.1 Taustaa

Satunnaisuus on oleellinen osa proseduraalista generointia, mutta se ei ole kuitenkaan täysin satunnaista. Proseduraalisella generoinnilla tarkoitetaan sitä, että luodaan jotain sääntöjen perusteella, satunnaisuudella lisätään mukaan vaihtelevuutta. (Doull 2008 a.) Joissakin tapauksissa voidaan kuitenkin haluta aina samanlainen lopputulos, kun käytetään tiettyjä siemenlukuja (Togelius 2010, 3).

Proseduraalinen generointi lähtee liikkeelle niin kutsutuista siemenluvuista (seed number), vähän niin kuin puu saa alkunsa siemenestä. Siemenlukuja voi olla käytössä yhdestä niin moneen kuin näkee tarpeelliseksi. Useampi siemenluku antaa enemmän kontrollia sisällön yksityiskohtiin. Siemenluku voi olla satunnainen, ennalta määrätty tai se voidaan ottaa esimerkiksi käyttäjän syöttämistä tiedoista. (Doull 2008 a.)

Proseduraalista generointia käytetään esimerkiksi grafiikan, kuten tekstuurien ja monikulmioista muodostuvien mallien (polygon mesh), luontiin. Grafiikkaa luodaan algoritmeilla niin sanotusti lennossa. Tällöin malleja ei tarvitse luoda etukäteen valmiiksi, jolloin ohjelman koko pysyy pienenä. (Wikipedia 2011 c) Kun malli generoidaan ohjelman ajon aikana, se vie tilaa vain tietokoneen keskusmuistista. Demoscenessä tätä on käytetty hyödyksi demojen mahduttamiseen pieneen kokoon. (Wikipedia 2011 b.)

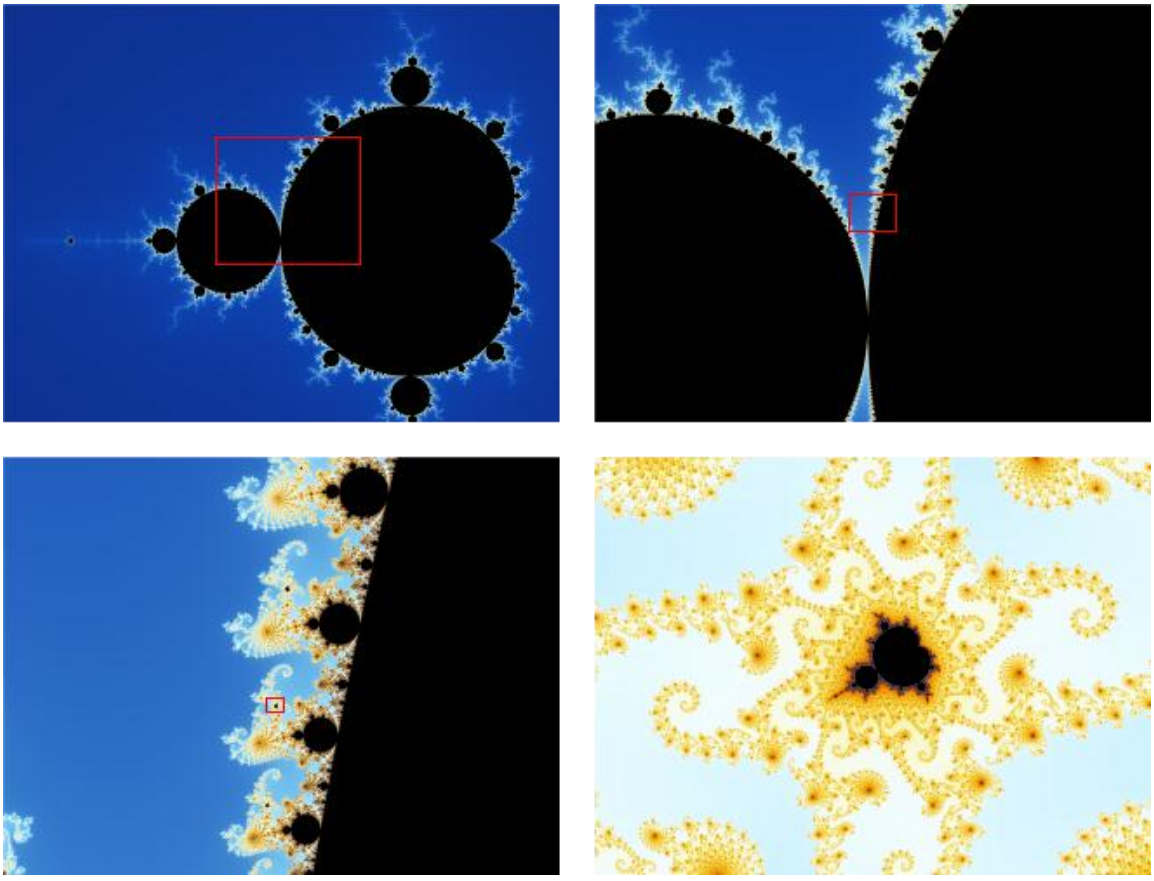
Äänet ja musiikki on myös mahdollista generoida proseduraalisesti. Äänien generoinnissa proseduraalisuutta voi käyttää yksinkertaisista äänistä aina puhesynteesiin asti.

Puhesynteesillä tarkoitetaan esimerkiksi tekstistä puheeksi -ohjelmia, joissa ohjelma puhuu kirjoitetun tekstin. (Wikipedia 2011 b.)

2.2 Algoritmeja

Proseduraalisen generoinnin joitakin algoritmeja voi kutsua fraktaaleiksi. Niissä luodaan esimerkiksi grafiikkaa fraktaalien avulla. Lähtökohtana on yksinkertainen muoto jota tarkennetaan jokaisella algoritmin iteraatiolla. (Martz 1997.)

Vuonna 1975 Benoît Mandelbrot keksi termin fraktaali, joka tarkoittaa geometrista muotoa jonka osat ovat pienennettyjä kopioita kokonaisuudesta. Esimerkiksi, kun Mandelbrot fraktaalia (Kuvio 1.) zoomaa lähemmäksi, toistuvat samat muodot kokoajan. Fraktaalien tarkkuus riippuu siitä kuinka monta iterointiaskelta siihen käytetään. Fraktaalien luonti lähtee yksinkertaisesta muodosta ja tarkentuu jokaisella iteraatiolla. (Mandelbrot 1982.)



Kuvio 1. Mandelbrotin fraktaali (Wikipedia 2011 a)

2.2.1 Keskipisteen siirto algoritmi

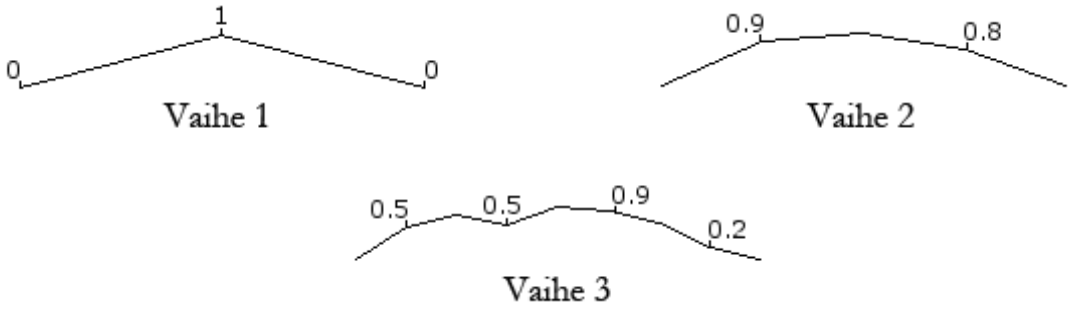
Keskipisteen siirto algoritmilla luodaan niin sanottuja korkeuskarttoja (height map). Korkeuskartat ovat esimerkiksi kaksiulotteisia taulukkoja joissa yhteen alkioon tallennetaan tieto sen pisteen korkeudesta. Eli esimerkiksi x, y ja z. Kaksiulotteisen taulukon x ja y toimivat koordinaattina kaksiulotteisella pinnalla ja alkion arvo z on korkeus sen koordinaatin kohdalla. Taulukosta voi sitten luoda peliin maaston ulkoilmakenttiä varten. Tämä toimii hyvin kolmiulotteisissa peleissä joissa korkeuskarttoja onkin käytetty. (Martz 1997.)

Keskipisteen siirtoa voi myös käyttää sivultapäin kuvatuissa peleissä, kuten tämän opinnäytetyön toiminnallisen osan projektissa. Sen sijaan, että luodaan kaksiulotteinen taulukko, luodaan yksiulotteinen taulukko jossa yhden alkion arvo on sen pisteen korkeus: x ja y. Tämä toimii ainakin tile-pohjaisissa peleissä, koska alkion arvo voidaan kertoa tilen leveydellä jolloin saadaan tilen paikka pelimaailmassa. Kertomalla y:n arvo tilen korkeudella, saadaan tilen paikka y-koordinaatilla.

Algoritmi

Keskipisteen siirto algoritmi on fraktaali eli siinä lähdetään liikkeelle epätarkasta muodosta ja tarkennetaan muotoa ennalta määrättyyn tarkkuuteen asti. Funktiolle annetaan jana ja sen janan päiden y-arvoista otetaan keskiarvo joka sijoitetaan janan keskimmäiseen kohtaan (Kuvio 2). Uutta arvoa myös siirretään satunnaiseen suuntaan, jotta mukaan saadaan satunnaisuutta. Keskipisteen kohdalta saadaan kaksi uutta janaa ja näille tehdään sama keskipisteen siirto. (Shankel 2000, 503.)

Algoritmia toistetaan niin monta kertaa kuin halutaan. Mitä useammin keskipisteen siirtoa kutsutaan, sitä tarkempi muoto janasta kehittyy. Keskipistettä siirrettäessä voi muodon karkeuteen vaikuttaa käyttämällä esimerkiksi jotain vakiolukua, vaihtoehtoisesti lukua voi siirtää ihan vain satunnaisesti. (Shankel 2000, 503.)

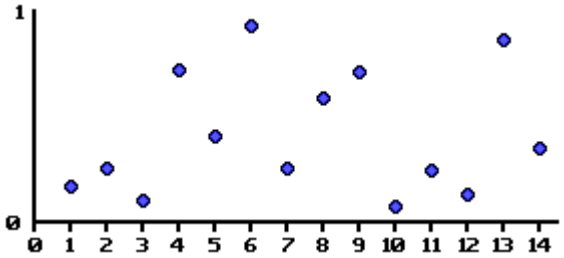


Kuvio 2. Keskipisteen siirto algoritmin vaiheita

2.2.2 Perlin-kohina

Kaikki proseduraalisen generoinnin algoritmit eivät välttämättä ole fraktaaleja. Esimerkiksi Perlin-kohina (Perlin noise) ei itsessään ole fraktaalialgoritmi, mutta sitä voi käyttää fraktaalialgoritmin tapaan. Tällöin tuloksista tulee yksityiskohtaisempia. (Perlin 1999.) Perlin-kohinaa käytetään, kun halutaan luonnollista vaihtelevuutta esimerkiksi tekstuureihin tai animaatioihin. Perlin-kohinaa voi käyttää myös kokonaan proseduraalisten tekstuurien luontiin. (Perlin 1985, 287.)

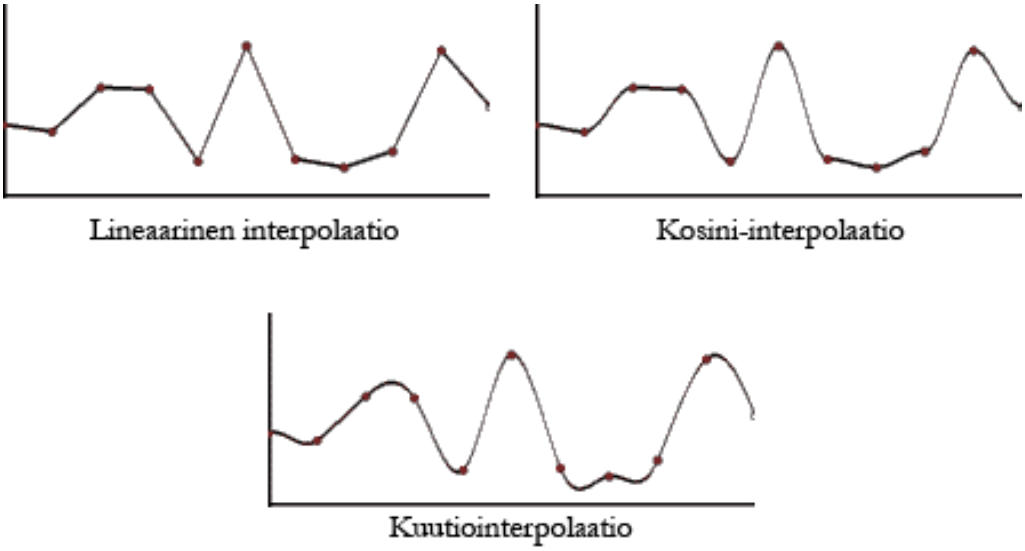
Perlin-kohinan tekeminen on yksinkertaisimmillaan satunnaislukujen generointia. Kohina näyttää tällöin karkealta (Kuvio 3.), koska satunnaisuus ei reagoi viereisten pisteiden arvoihin. Tätä varten voidaan käyttää interpolointia joka tekee muutoksista sulavampia ja luonnonmukaisempia. Interpoloinnin avulla Perlin-kohinaa voi käyttää myös maaston generointiin. (Elias 2003.)



Kuvio 3. Yksiulotteinen Perlin-kohina (Elias 2003)

Interpolaatiofunktioita on monia erilaisia (Kuvio 4.). Esimerkiksi lineaarinen interpolaatio, kosini-interpolaatio (cosine interpolation) ja kuutiointerpolaatio (cubic interpolation).

Jokainen funktio tuottaa erilaisia tuloksia, kuten kuviosta voi huomata. Eri funktioita pitää testailla, jotta löytää omaan tarkoitukseen sopivan. (Elias 2003.)



Kuvio 4. Erilaisia interpolaatioita (Elias 2003)

3 PROSEDURAALINEN SISÄLLÖN GENEROINTI

Nykyaikana yleisin tapa luoda sisältöä peleihin on tehdä sitä käsin. Esimerkiksi pelien kentät tehdään niin, että kentäntekijä asettelee kaikki kentän osat käsin paikalleen. Se vie paljon aikaa ja erityisesti isoissa projekteissa se on kallista. (Doull 2008 a.)

Proseduraalinen sisällön generointi on niin sanotusti jatkumoa proseduraaliselle generoinnille. Proseduraalisessa generoinnissa käytettyjä tekniikoita käytetään hyväksi pelin sisällön luontiin. (Doull 2010.) Joitakin tekniikoita käytetään hyväksi esimerkiksi animaatioiden, tekstuurien ja äänien generointiin eli niin sanotusti pelillisesti vähemmän tärkeisiin asioihin. Proseduraalisella sisällön generoinnilla tarkoitetaan pelillisesti merkittävän sisällön generointia. Esimerkiksi kentät, hahmot, juonet jne. (Doull 2008 a.)

Proseduraalinen sisällön generointi on tapa, jolla sisältöä luodaan satunnaisesti tai pseudosatunnaisesti algoritmien avulla. Kyse ei ole täysin satunnaisesta generoinnista, vaan sisältö generoidaan ennalta määrättyjen sääntöjen mukaan. (Introversion Software 2007, 1.)

3.1 Taustaa

Proseduraalista sisällön generointia voi käyttää monella eri tavalla peleissä. Näistä eri tavoista on kirjoittanut Andrew Doull artikkelissaan The Death of Level Designer: Procedural Content Generation in Games. (Doull 2008 a.) Lisäksi aiheesta on kirjoittanut Julian Togelius yhdessä Georgios N. Yannakakiksen, Kenneth O. Stanleyn ja Cameron Brownen kanssa artikkelissa Search-based Procedural Content Generation (Togelius 2010.).

Doull määrittelee proseduraalisen sisällön generoinnin peleissä seitsemään eri kategoriaan:

- ajonaikainen satunnaisten kenttien generointi
- kentän sisällön suunnittelu
- dynaaminen maailman luonti
- pelin sisäisten entiteettien instantiointi

- käyttäjien sovittelema sisältö
- dynaamiset järjestelmät
- proseduraalinen pulman ja juonen generointi. (Doull 2008 a; Doull 2008 b.)

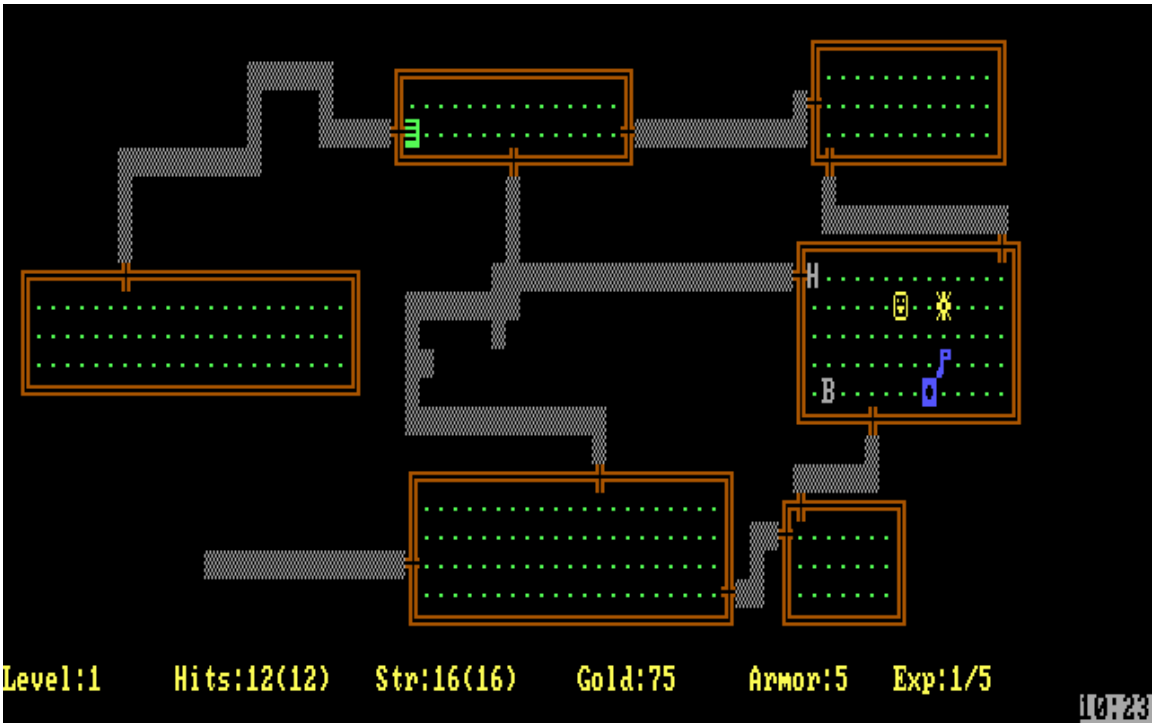
Togelius määrittelee proseduraalisen sisällön viiteen kategoriaan:

- yhteydessä oleva (online) vs. yhteydetön (offline)
- tarpeellinen vs. vapaaehtoinen sisältö
- satunnainen siemen vs. parametri vektorit
- stokastinen vs. deterministinen generointi
- rakenteellinen vs. generoi ja testaa. (Togelius 2010.)

Yhteydessä oleva vs. yhteydetön proseduraalinen sisältö on periaatteessa sama kuin Doullin ajonaikainen satunnaisten kenttien generointi ja kentän sisällön suunnittelu. Lisäksi Togelius ja kumppanit määrittelevät proseduraalisen sisällön piirteitä hieman alemmalla tasolla kuin Doull. (Togelius 2010.)

Ajonaikainen satunnaisten kenttien generointi

Tunnetuimpana proseduraalisena sisällön generointina voidaan pitää ajonaikaista satunnaisten kenttien generointia. Kyse on kenttien luonnista samalla, kun peli on käynnissä. Tämä on tuttu jo vanhasta klassikosta nimeltä Rogue (1980, Kuvio 5.) sekä Beneath Apple Manor (1978). Se on pysynyt mukana samantyylisissä peleissä tähänkin päivään saakka ja näitä pelejä kutsutaan yleensä roguelike-peleiksi, Roguen mukaan. Tämän tyylisissä peleissä liikutaan maanalaisissa luolastoissa jotka peli luo asettelemalla huoneita satunnaisesti ja yhdistämällä niitä käytävillä. (Doull 2008 a.)



Kuvio 5. Roguen DOS-versio (MobyGames 2000)

Tämä proseduraalisen generoinnin ilmentymä on erityisen suosittu juuri näiden roguelike-pelien kohdalla. Näitä pelejä tehdään edelleen ja monet suosituimmista ovat olleet tekeillä pitkään ja niiden kehitys jatkuu vieläkin. (Doull 2008 a.)

Nykyaikaisempia pelejä joissa tätä tapaa käytetään, ovat esimerkiksi Spelunky (Kuvio 6.) ja Diablon tyyliset pelit. Perinteisesti roguelike-peleissä kuvakulma on ylhäältä alas, mutta Spelunky muuttaa tämän konseptin sivulta kuvatuksi. Eli Spelunky yhdistää Super Mario Bros. -tyylistä tasoloikkaa proseduraalisesti generoituihin kenttiin. (Yu 2009.)

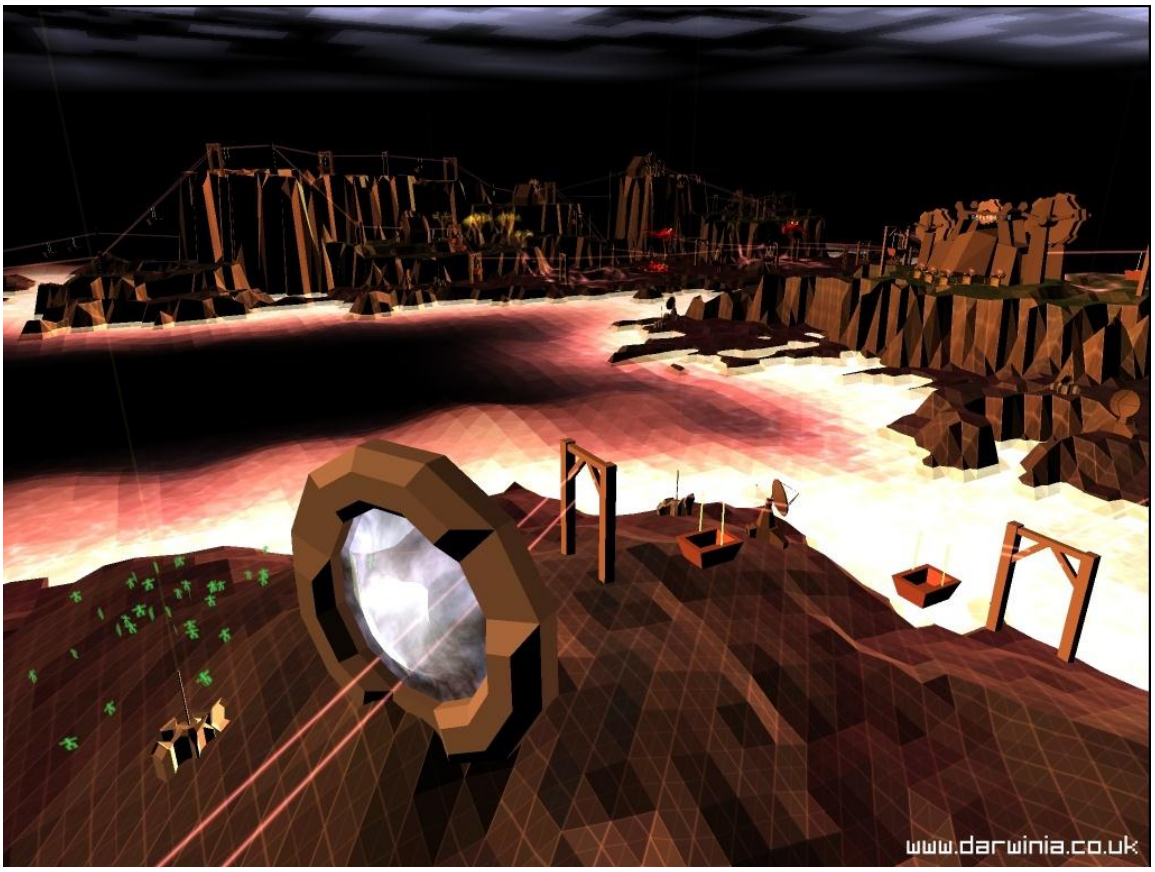
Kuvio 6. Spelunky (Yu 2009.)

Kentän sisällön suunnittelu

Proseduraalista generointia voidaan käyttää myös niin, että luodaan työkalu joka automaattisesti luo erilaista sisältöä kenttiin tai kenttiä itsessään. Tämä eroaa ajonaikaisesta generoinnista niin, että pelintekijä valitsee ennakkoon generoidusta sisällöstä sopivimmat ja tallentaa siemenluvat muistiin. Pelissä sitten algoritmit generoivat ennakkoon valitun sisällön tallennettujen siemenlukujen perusteella. Pelin tekijän tarvitsee vain muokata algoritmeja ja valitsee miellyttävimmän vaihtoehdon. (Doull 2008 a.)

Suosittu käyttökohde on niin sanotut korkeuskartat tai kokonaisen pelimaailman maaston generointi. Korkeuskartat ovat yksi osa pelimaailman luontia. Niiden avulla määritellään esimerkiksi, kuinka mäkinen maasto on. Kun tähän otetaan vielä huomioon vesi, joka asetetaan esimerkiksi korkeuskartassa tietylle korkeudelle, niin saadaan mielenkiintoisempia maailmoja. Tämän lisäksi vaiheittain voidaan maailmaa muokata lisäämällä jokia ja vaikka eroosion vaikutusta. (Johnston 2009 b.) Esimerkki korkeuskartan generoinnista löytyy kappaleesta 2.2.1 .

Hyvä esimerkki tätä tapaa hyödyntävästä pelistä on Darwinia (Kuvio 7.). Pelin maastot ovat proseduraalisesti generoituja, koska muuten kenttien tekijän olisi pitänyt sijoitella maaston kaikki pisteet käsin ja se on todella työlästä. Rakennukset ja kenttien juonelliset tapahtumat taasen Darwiniassa ovat käsin luotuja ja sijoiteltuja. Täysin proseduraalisesti luotuja kenttiä Introversion Software yritti tehdä, mutta tämä osoittautui kuitenkin vaikeaksi tavaksi luoda pelattavuudeltaan hauskoja kenttiä. (Introversion Software 2007, 2.)



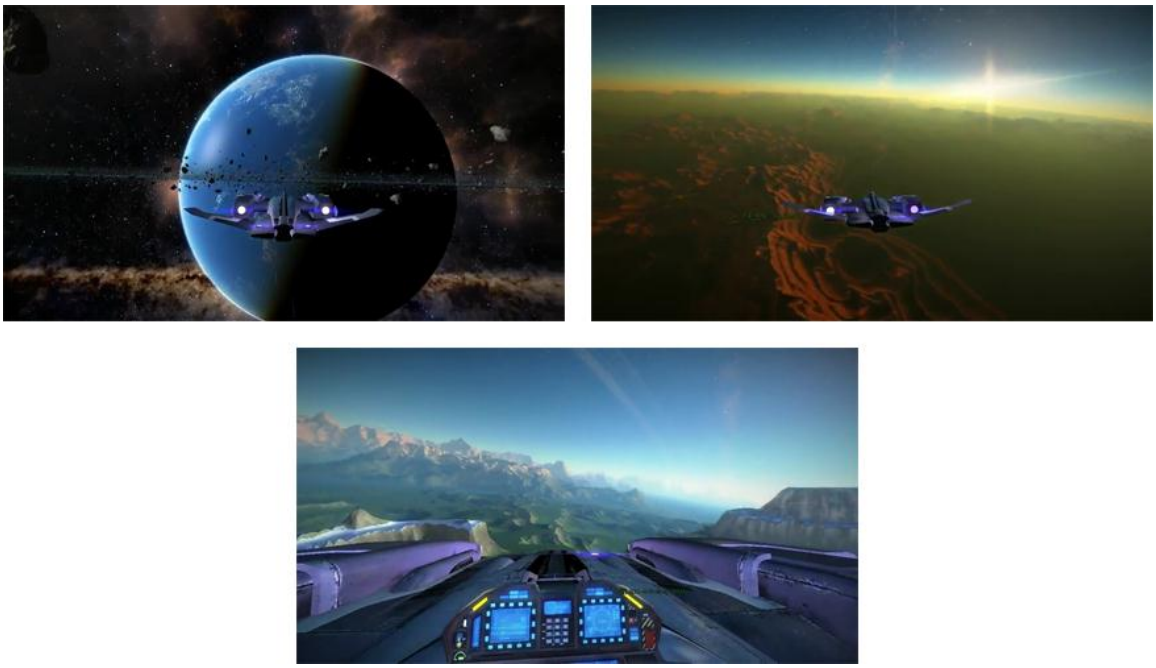
Kuvio 7. Darwinia (Introversion Software 2005)

Tätä proseduraalista generointia hyödyntävät strategiapelit, joissa pelaaja voi antaa tiettyjä parametreja pelille. Peli generoi näiden parametrien perusteella kartan ja pelaaja voi halutessaan pelata kyseisellä kartalla tai pyytää peliä generoimaan uuden kartan. Tyypillisiä parametreja ovat esimerkiksi kartan koko (pieni, keskikokoinen, suuri jne.) ja veden sekä maan määrä. Yleensä tämä generointimahdollisuus tarjotaan varsinaisen pääpelin lisäksi. Esimerkiksi Age of Empires -sarja hyödyntää tätä. (Ensemble Studios 2005.)

Dynaaminen maailman luonti

Joidenkin pelien maailmat ovat sen verran isoja, että niitä eivät sen hetkiset tietokoneet yksinkertaisesti pysty käsittelemään kokonaisuudessaan. Silloin proseduraalista generointia voi käyttää hyväksi hieman kentän sisällön luonnin tyyliin, niin että käytetään lähtöarvona jotain siemenlukua, jonka perusteella maailmaa generoidaan sen mukaan, missä pelaaja sijaitsee maailmassa. Tarvittaessa vanhaa aluetta poistetaan muistista. Näin maailma näyttää samalta aina tietyssä paikassa kaikille pelaajille, mutta jos pelaaja ei ole kyseisellä alueella niin se ei vie tehoja pelaajan koneelta, koska kyseistä aluetta ei pitäisi tarvita piirtää tai muuten käsitellä. (Doull 2008 a.)

Kyseistä tekniikkaa käytettiin aikoinaan jo vuonna 1984 Elite-pelissä, jolloin kyseiselle tekniikalle oli erityisesti tarvetta, koska koneet eivät olleet tehokkaita (Spufford 2003). Nykyaikaisemmista peleistä, tosin vielä tekeillä oleva, Infinity: Quest For Earth (Kuvio 8.) käyttää myös tätä tekniikkaa. Siinä avaruutta on jo yhden universumin verran ja planeettojenkin pinnat ovat proseduraalisesti generoituja. (Doull 2008 a.)

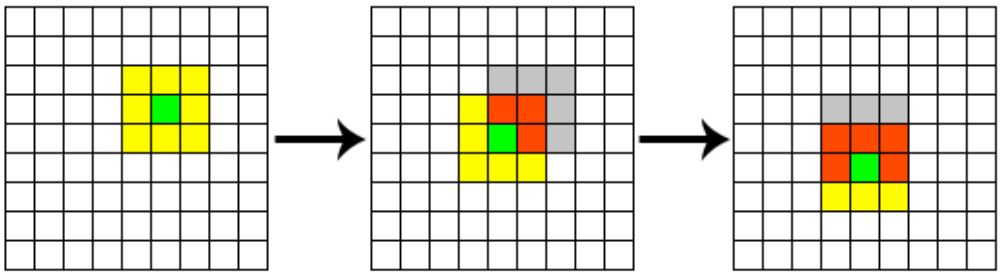


Kuvio 8. Infinity: Quest For Earth (InfinitySupport 2010)

Tässä tekniikassa ongelmallisinta on tosin se, että joidenkin asioiden kanssa se toimii heikosti. Esimerkiksi tiet ja joet voivat olla niin pitkiä, että ne yltävät kauemmaksi kuin mitä pelaajan sen hetkinen alue käsittelee ja teiden sekä jokien generointiin voidaan tarvita tietoa,

joka tulee nykyisen alueen ulkopuolelta. Tämän takia nämä osat maailmasta voivat näyttää erilaisilta eri paikoissa. (Doull 2008 a.)

Kuvio 9 havainnollistaa dynaamisen maailman luonnin periaatetta. Kuviossa maailma on jaettu 81 alueeseen. Pelaajan liikkuessa alueelta toiselle generoidaan uusia alueita sekä poistetaan alueita jotka ovat jääneet liian kauaksi pelaajasta sen mukaan mihin pelaaja on liikkunut. Pelaaja on vihreällä värillä osoitetulla alueella. Keltaiseksi värjätyt alueet merkitsevät uusia alueita jotka on dynaamisesti generoitu muistiin. Oranssilla värjätyt alueet ovat jo valmiiksi muistissa joten niitä ei tarvitse generoida uusiksi. Harmaat alueet ovat poistoa varten merkattuja alueita.



Kuvio 9. Dynaaminen maailman luonti

Pelin sisäisten entiteettien instantiointi

Proseduraalinen sisällön generointi ei rajoitu vain maailmojen ja kenttien luontiin. Sitä voidaan käyttää myös pelin sisäisten tavaroiden tai hahmojen luontiin. Normaalisti peliin luodaan monia erilaisia hahmoja, jotka kaikki vievät oman osansa tilaa ja yksilöllisten hahmojen määrä jää vähiin. Jos hahmot luodaan proseduraalisesti esimerkiksi käyttämällä ennalta tehtyjä osia – kuten erilaisia käsiä, jalkoja, päitä jne. – saa yksilöllisten hahmojen määrän huomattavasti suuremmaksi. Uudelleen käytetyt resurssit eivät myöskään vie niin paljoa tilaa kuin kokonaiset yksittäiset hahmot jotka saattavat sisältää samoja osia. Parhaassa tapauksessa myös yksittäiset osat on luotu proseduraalisesti. (Doull 2008 b.)

Esimerkiksi Borderlands (Kuvio 10.) käyttää tätä tekniikkaa aseiden luontiin. Pelissä pitäisi tekijöiden mukaan olla yli 500 000 erilaista asetta ja tähän päälle vielä muut varusteet. Sisällön määrän ei pitäisi ihan heti loppua. (Brudvig 2007.)

Kuvion oikeassa alalaidassa näkyy pelaajan käytössä oleva ase. Aseen ominaisuudet kuten vahinko (damage), tarkkuus (accuracy) ja ampumisnopeus (fire rate) ovat satunnaisesti arvottu tietyltä väliltä. Lisäksi, jos kyseessä on erikoisempi ase (kuten kuviossa), generoidaan siihen tiettyjä erikoisominaisuuksia jotka on lueteltu vahingon, tarkkuuden ja ampumisnopeuden alapuolelle.

Kuvio 10. Borderlands (Brudvig 2007)

Käyttäjien sovittelema sisältö

Proseduraalisesti generoitu sisältö voi helposti näyttää huonolta ihmisen silmissä. Tässä käyttäjä voi auttaa niin, että hän valitsee generoidusta sisällöstä omasta mielestään sopivammat. Normaalisti proseduraalisesti generoituun sisältöön voi vaikuttaa vain epäsuorasti, mutta käyttäjien sovitellessa sisältöä saa mukaan ihmisälykkyyttä. Tällä tavalla käyttäjä voi tutkia proseduraalista sisältöä ilman suurempaa tietotaitoa. (Doull 2008 b.)

Pelien kohdalla tunnetuin esimerkki tästä on Spore (Kuvio 11.). Siinä pelaaja voi luoda täysin omanlaisensa olion käyttämällä erilaisia osia, kuten jalkoja, käsiä, päitä ja muita ulokkeita. Peli tarjoaa osista erilaisia versioita, joista pelaaja valitsee mieleisensä ja sen jälkeen muokkaa

niiden kokoa, asentoa, muotoa jne. Sporen proseduraaliset algoritmit varmistavat, että pelaajan luoma hahmo myös animoituu luontevasti. (Doull 2008 b.) Kuvion vasemmalla puolen näkyy valikko josta pelaaja voi valita erilaisia osia joilla hahmon ulkonäköä voi muokata.

Kuvio 11. Spore Character Creator (Kohler 2008)

Dynaamiset järjestelmät

Pelin dynaamisuus on yksi tapa lisää saada pelimaailmasta erilainen jokaisella pelikerralla. Sää, yö-päivä-rytmitys ja tekoäly ovat yleisiä dynaamisen järjestelmän käyttökohteita. Esimerkiksi dynaaminen tekoäly reagoi pelaajan tekemiin asioihin jolloin mikään kohtaaminen vihollisen kanssa ei ole täysin samanlainen. Dynaaminen tekoäly ei ole täysin proseduraalista sisällön generointia, mutta siinä on paljon samaa. Sää lisää tunnelmaa pelimaailmaan ja proseduraalisuudella siihen saadaan luonnollista siirtymistä tilasta toiseen, ettei pääse käymään niin, että ensiksi paistaa aurinko ja yhtä äkkiä onkin myrsky. Myös musiikkia voidaan käyttää pelissä dynaamisesti. Dynaaminen musiikki reagoi pelin tapahtumiin.

Esimerkiksi, kun pelaaja on taistelussa, musiikki muuttuu nopeatempoisemmaksi ja jännittävissä kohdissa musiikki hiljenee. (Doull 2008 b.)

S.T.A.L.K.E.R.: The Shadow of Chernobyl hyödyntää dynaamista tekoälyä. Se käyttää myös proseduraalisuutta generoimaan vihollisia pelimaailmaan, jolloin samassa paikassa ei ole aina samanlaisia kohtaamisia. Tämä yhdistettynä vielä dynaamiseen tekoälyyn niin kohtaamiset eivät varmasti ole samanlaisia. (Doull 2008 b.)

Façade (Kuvio 12.) on kokeellinen peli, joka hyödyntää dynaamisuutta niin, että se reagoi pelaajan puheisiin (tosin pelissä ne varsinaisesti kirjoitetaan) ja keskustelu siirtyy suurimmaksi osaksi luonnollisesti aiheesta toiseen. (Doull 2008 b.) Kuviossa pelaaja on kirjoittamassa omaa reaktiotaan pelin tapahtumiin.

Kuvio 12. Façade (Procedural Arts 2008)

Proseduraalinen pulman ja juonen generointi

Yksinkertaisimmillaan proseduraalinen pulman generointi tarkoittaa esimerkiksi ovikoodien vaihtamista lukituissa ovissa. Proseduraalista generointia voidaan kuitenkin käyttää monipuolisemminkin. Periaatteessa sen avulla voidaan luoda lähes ääretön määrä tapoja

ratkaista jokin pulma tai ääretön määrä erilaisia pulmia. Tätä voidaan hyödyntää joko niin, että yhdellä pelikerralla on mahdollista jokaisen pelaajan ratkaista pulma omalla tavallaan tai niin, että pulman ratkaisu vaihtelee jokaisella pelikerralla. Ensimmäisessä tavassa jokainen pelaaja pääsee ratkaisemaan pulman omalla tavallaan, mutta pulma on aina sama. Jälkimmäisessä tavassa jokaisen pulman ratkaisu vaihtelee jokaisella pelikerralla. (Doull 2008 b.)

Peleissä, joissa ei käytetä proseduraalista pulman tai juonen generointia, on ongelmana se, että jos epäonnistuu, samoja kohtia joutuu aina yrittämään uusiksi ilman muutoksia kohtaukseen. Proseduraalinen juonen generointi auttaa samalla tavalla pelin juoneen. Sen avulla juonen käänteet eivät aina ole samanlaisia. Nykyaikana on yleistä, että peleissä on monta erilaista loppua, mutta näihin ei käytetä proseduraalisia menetelmiä, jolloin erilaiset loput määräytyvät tiettyjen tapahtumien perusteella ja ovat aina samanlaisia. Proseduraalisuuden avulla voitaisiin myös mahdollistaa monen erilaisen alun luonti. Tämä lisäisi jo heti pelin aloitukseenkin erilaisuutta. Kun pelin juoneen lisätään dynaamisuutta proseduraalisilla menetelmillä, pelaajan tekemiin valintoihin saadaan oikeanlaista merkitystä joilla voi olla merkittäviäkin vaikutuksia pelin tarinaan. (Doull 2008 b.)

Juonen proseduraalinen generointi on vielä lapsen kengissä ja sitä ei ole kokeiltu kovin monessa pelissä. Proseduraalista pulman generointia käytetään peleissä ovikoodeista myös monimutkaisempiin. (Doull 2008 b.)

Harrison Kingsleyn vuonna 1990 tekemä Murder! on Cluedo-lautapelin tyylinen murhamysteeri, jossa pelin alussa pelaaja voi muokata päivämääriä ja nimiä. Näiden parametrien perusteella peli generoi mysteerin. Syöttämällä samat tiedot uudestaan voi pelaaja yrittää ratkaista saman mysteerin uudestaan. Tekijöiden mukaan pelissä pitäisi olla lähes kolme miljoonaa erilaista mysteeriä. (Tew 2009.)

BlockRogue on Stan Pattonin vuonna 2009 tekemä pulmapeli jossa yhdistyy Sokobanin tyylinen pulmapeli ja Roguen tyylinen luolaston generointi. Pelissä on siis tarkoitus työnnellä laatikoita oikeisiin paikkoihin proseduraalisesti generoiduissa kentissä. (Patton 2009.)

Tarpeellinen vs. vaihtoehtoinen sisältö

Proseduraalista sisältöä suunniteltaessa pitää tietää, onko generoitava sisältö pelin etenemisen kannalta tarpeellista vai ei. Tämä määrittelee sen, että missä määrin sisällön pitää olla

toimivaa. Sisällön ollessa tarpeellista pitää sisällön myös olla toimivaa, jotta pelaaja ei pysty jäämään jumiin esimerkiksi jonkun väärin sijoitetun esteen takia. Tätä varten tarvitaan jonkinlainen oikeellisuuden tarkistus, esimerkiksi reitinhakualgoritmi. Tarpeelliseen sisältöön kuuluu esimerkiksi kenttä tai kentän osa, vaikka luolasto, jonka läpi pelaajan on pakko mennä, että hän pääsee seuraavalle alueelle. (Togelius 2010.)

Sisältö voi myös olla vaihtoehtoista joka tarkoittaa sisältöä jota pelaaja ei välttämättä koskaan näe. Tällöin sisällön toimivuutta ei välttämättä tarvitse tarkistaa, koska sisältö on vain lisätäytteenä elävöittämään maailmaa. Tähän kuuluu esimerkiksi vaihtoehtoiset luolastot jotka on luotu vain elävöittämään pelimaailmaa. (Togelius 2010.)

Satunnaiset siemenluvut vs. parametrivektorit

Proseduraalisessa generoinnissa voidaan käyttää yksinkertaisesti yhtä satunnaisesti valittua siemenlukua. Tällöin sisällön generointia ei ole mahdollista hallita niin paljoa, joka voi tietyissä tapauksissa olla suotavaakin, kuten ajonaikana satunnaisesti generoiduissa kentissä. (Togelius 2010.)

Generointiin voidaan käyttää myös monimutkaisempia parametreja. Kun generoinnissa käytetään enemmän parametreja, on sisältöön enemmän kontrollia. Näin voidaan vaikuttaa tarkemmin tiettyihin pelialueen osiin. (Togelius 2010.)

Stokastinen vs. deterministinen generointi

Proseduraalisessa generoinnissa käytetään yleensä satunnaisuutta. Satunnaisuudella varmistetaan esimerkiksi, että sisältöä on aina vaihtelevaa. Generointia joka tuottaa aina erilaista sisältöä vaikka olisi samat parametrit kyseessä, kutsutaan stokastiseksi generoinniksi. (Togelius 2010.)

Deterministinen generointi on stokastisen generoinnin vastakohta joka generoi aina samanlaista sisältöä, kun antaa saman siemenluvun tai -lukuja. Tästä on hyötyä, kun halutaan varmistaa esimerkiksi samanlainen sisältö kaikille pelaajille. (Togelius 2010.) Esimerkiksi Elite käyttää determinististä generointia, koska se generoi aina saman maailman jokaisella pelikerralla.

Rakentava vs. generoi-ja-testaa

Kuten tarpeellisen sisällön kohdalla jo huomioitiin, tarpeellinen sisältö pitää tarkistaa jotta sisältö olisi toimivaa. Tätä varten voidaan käyttää generoi-ja-testaa metodia jolloin generaattori luo sisältöä kahdessa vaiheessa. Ensimmäinen vaihe on itse generointi, jonka jälkeen se siirtyy testausvaiheeseen. Testausvaiheessa generaattori käy läpi generoidun sisällön ja tarkistaa sen oikeellisuuden. Jos sisältö ei vastaa laatuvaatimuksia, generaattori poistaa sen kokonaan tai osittain ja siirtyy taas generointi vaiheeseen. Rakentavassa generoinnissa sisältö generoidaan kerran ja oikeellisuus tulee generoinnin säännöistä. Maaston generointi fraktaalien avulla on tällaista. (Togelius 2010.)

3.2 Algoritmeja

Proseduraalista sisällön generointia varten voi käyttää monenlaisia algoritmeja. Jotkut algoritmeista voi olla tehty proseduraalista sisällön generointia varten, mutta joitakin algoritmeja voi myös soveltaa sisällön generointiin.

Luolaston generointi

Klassinen esimerkki ajonaikaisesta generoinnista on roguelike-pelien luolastojen generointi. Niissä luolastot generoidaan esimerkiksi, kun pelaaja siirtyy tasolta toiselle. Luolastojen generointiin voi käyttää hyvin monimutkaisiakin algoritmeja, mutta yksinkertaisilla algoritmeilla voi myös saada toimivia luolastoja (Kuvio 13.).

Esimerkkinä Chevy Ray Johnstonin kehittämä yksinkertainen algoritmi, jonka vaiheet menevät seuraavanlaisesti:

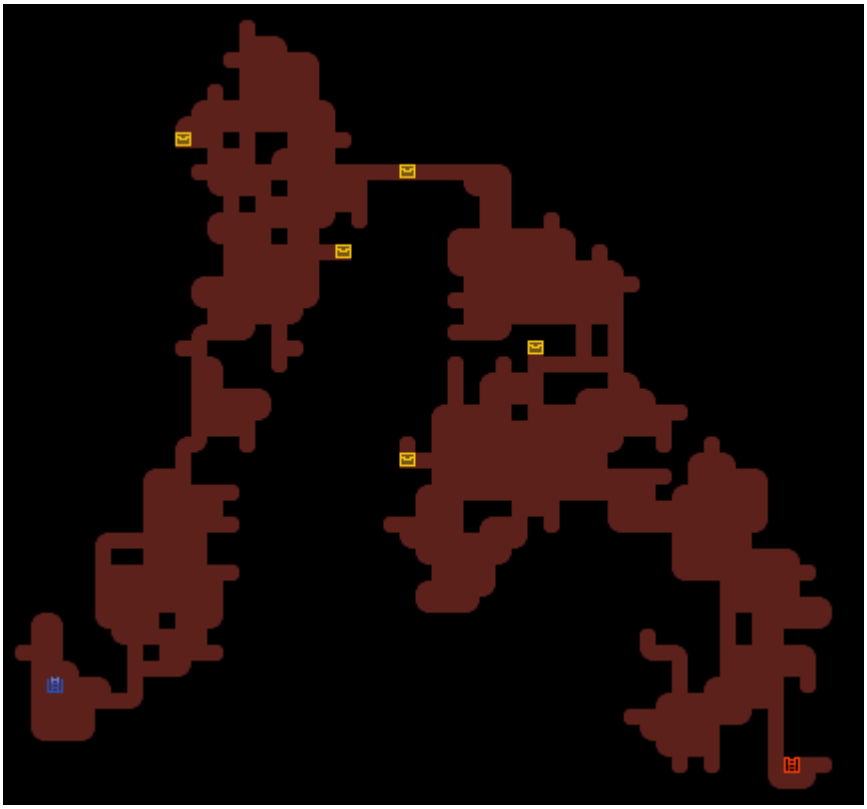
1. Kaiverra aloitusruutu ja lisää siihen rappuset ylöspäin.
2. Lisää ruutu listaan.
3. Aloita listan läpikäyminen ensimmäisestä jäsenestä.
4. Arvo, mitkä viereiset ruudut kaiverretaan.
5. Lisää kaiverretut ruudut listaan.
6. Siirry listan seuraavaan jäseneen.

7. Arvo, kaiverretaanko tälle ruudulle yhtään viereistä ruutua.

8. Arvo, lisätäänkö tähän kohtaan aarre, vihollinen, ansa jne.

9. Toista kohtia 4 - 8, kunnes listan viimeisellä ruudulla ei ole kaiverrettavia ruutuja vieressä.

10. Lisää rappuset alaspäin. (Johnston 2009.)



Kuvio 13. Esimerkkiluolasto (Johnston 2009.)

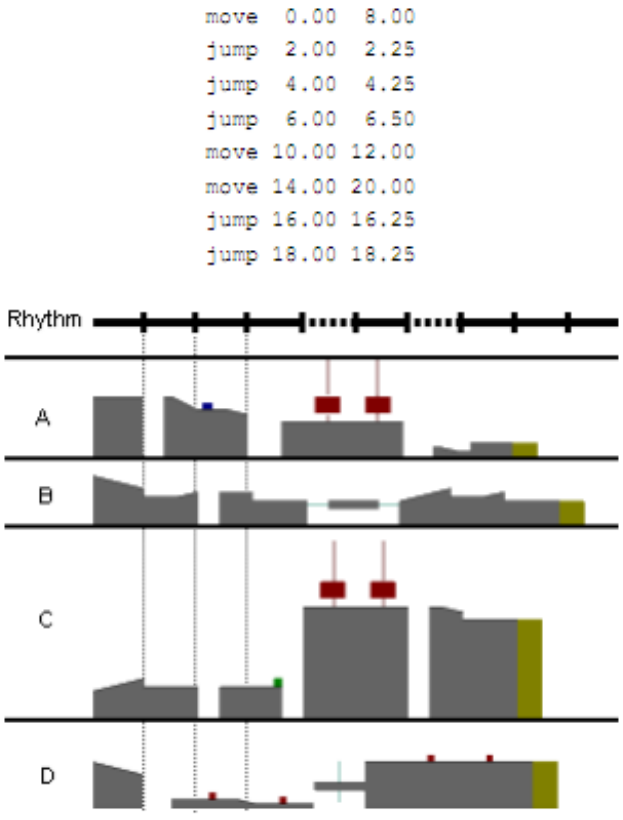
Rytmipohjainen kentän generointi

Rytmipohjainen kentän generointi (Rhythm-Based Level Generation) on Gillian Smithin, Mike Treanorin, Jim Whiteheadin ja Michael Mateasin kehittämä algoritmi jonka avulla saa luotua tasoloikkapelien tyylisiä kenttiä. (Smith 2009, 1.)

Rytmipohjaista generointia voi käyttää rakentavana tai generoi-ja-testaa algoritmina. Generoi-ja-testaa metodi on hyvä, kun luo yksittäisiä kenttiä, mutta rytmipohjainen kentän generointi toimii myös hyvin rakentavana. Toiminnallisen osan projektissa oli käytössä rakentava versio tästä generaattorista. Generaattorille annetaan fysiikkarajoitteet jonka perusteella se generoi esimerkiksi kuilun leveyden tai tason korkeuden, jotta pelaaja pystyy tekemään hypyn eikä pelaajan eteneminen jää siihen. (Smith 2009, 4.)

Rytmipohjainen kenttä koostuu monista perättäisistä rytmiryhmistä. Rytmiryhmät sisältävät vaihtelevan määrän rytmejä. Rytmissä on erilaisia toimintoja tai iskuja kuten esimerkiksi liiku ja hyppää. Toiminnalle merkitään aloitusaika sekä loppumisaika. Toimintojen määrän ja pituuden lisäksi rytmillä voi olla erilaisia iskutyyppejä (beat type). Iskutyyppi määrittelee, miten toiminnat esiintyvät rytmissä. Iskutyyppejä voi olla esimerkiksi tasainen, satunnainen sekä swing eli rytmit tapahtuvat useammin ryhmän alussa sekä lopussa ja harvemmin keskellä. (Smith 2009, 3.) Rytmipohjaisen generoinnin tuottamat rytmit voi nähdä kuin musiikin nuottikirjoitukset. Erilaiset nuotit esittävät rytmipohjaisen kentän eri toimintoja.

Generaattori on kaksivaiheinen. Se generoi ensiksi kentän rytmit ja rytmien pohjalta geometrian. Rytmit toimivat siemeninä geometriageneroinnille. Geometrian generointi toimii stokastisesti, jotta samoista rytmeistä saadaan erilaisia geometrioita. Kentän rytmitys tuntuu samalta, mutta geometrisesti kenttä on aina erilainen. Esimerkiksi jos tahtoisi käyttää aina samaa rytmitystä, ainakin geometria olisi vaihtelevaa. Kuvio 14 on esimerkki neljästä erilaisesta geometriasta jotka on luotu samasta rytmistä. Esimerkiksi lyhyt hyppy voi geometriana olla hyppy toiselle tasolle – joka voi myös olla eri korkeuksilla – vihollisen päälle, ilmassa olevaa kerättävää kohti jne. (Smith 2009, 3.)



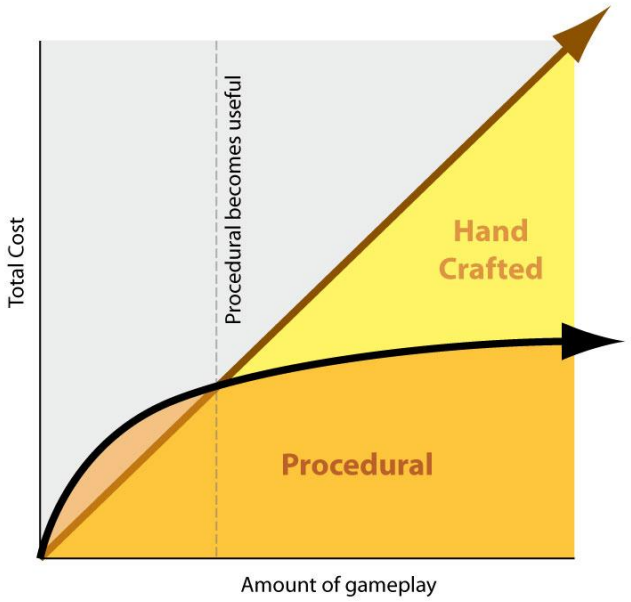
Kuvio 14. Neljä erilaista geometriaa samasta rytmistä (Smith 2009, 4.)

Rytmipohjaisessa generoinnissa käytetään myös tyyliparametreja. Tyyliparametrit ovat suunnittelijan muokattavissa jolloin suunnittelija voi myös vaikuttaa siihen mitä tiettyihin rytmiryhmiin generoidaan. (Smith 2009, 5.)

Geometriageneraattori ottaa rytmien toiminnat ja luo niistä tulkinnan geometriaksi kentälle. Generaattori voi luoda useamman ehdokkaan, jos ehdokas ei täytä kaikkia kriteereitä. Kriteerien avulla tarkistetaan, että generoitu rytmiryhmä on myös globaalilla tasolla tarpeeksi hyvä. (Smith 2009, 5.)

3.3 Hyödyt ja haitat

Pelien tekemiseen tarvittava rahamäärä kasvaa yhtä nopeasti kuin pelien koko, kuten kuviosta voi huomata (Kuvio 15.). Peleihin luodaan nykyaikana aina vain suurempia ja hienompia kokonaisuuksia. Tämän takia peleihin tarvitaan tarkempia tekstuureja, yksityiskohtaisempia hahmoja, enemmän pelattavaa jne. Kun tätä sisältöä tehdään käsin niin työmäärä alkaa kasvaa todella suuriin mittoihin ja samalla rahaa kuluu enemmän ja enemmän. Isojen pelitalojen tekemät pelit maksavat helposti nykyään miljoonia. (Danc 2007.)



Kuvio 15. Käsintehdyn ja proseduraalisesti generoidun sisällön tuoton kustannukset suhteessa sisällön määrään. (Danc 2007)

Toimivien ja sopivien algoritmien löytäminen voi olla aluksi hankalaa ja aikaa vievää, mutta kun algoritmit on löydetty, sisällön luonti peliin on periaatteessa vain napin painalluksen päässä. (Kuvio 15.) (Introversion Software 2007, 1.)

Pelien kaikkea sisältöä ei kuitenkaan välttämättä kannata jättää algoritmien tehtäväksi, koska proseduraalinen generointi on hyvin matemaattisesti painottunutta. Tämän takia sisältö voi pidemmän päälle alkaa pelaajasta tuntua matemaattiselta. Esimerkiksi vaikka tehtäviä pelissä olisi teoriassa ääretön määrä, niiden erilaisuus ei välttämättä ole tarpeeksi selkeää, jotta tehtävien suorittaminen pelaajasta tuntuisi mielenkiintoiselta. Kun David Braben ja Ian Bell vuonna 1984 tekivät Eliteä, heidän proseduraalisesti luotu universumi oli aluksi niin iso, että sinne mahtui 2^{48} eli 281 474 976 710 656 galaksia. He päättivät kuitenkin rajata galaksien määrän kahdeksaan, koska alkuperäisissä määrissä pelaaja varmasti huomaisi pelimaailman keinotekoisuuden. (Spufford 2003.)

Projektin alussa pitää miettiä, että mikä pelin sisällöstä on kertakäyttöistä ja uudelleenkäytettävää. Kertakäyttöinen sisältö tarkoittaa kaikenlaista merkityksetöntä, jolla pelaaja ei varsinaisesti tee mitään. Se on vain täytteenä pelin maailmassa. Esimerkiksi keskustelut vähemmän tärkeiden pelihahmojen kanssa ovat kertakäyttöisiä. Uudelleenkäytettävä sisältö on asioita pelimaailmassa, johon pelaaja palaa tai törmää uudestaan sen hyödyllisyyden takia. Esimerkiksi kauppa jonne pelaaja palaa kauppaamaan löytämäänsä tavaraa. (Danc 2007.)

Käsityöllä tehty sisältö ei ole helposti muokattavaa. Esimerkiksi, jos ensiksi peliä varten tekee vaikka pellehahmon ja toteaa myöhemmin, että sen tilalla toimisi paremmin jokin toinen hahmo, koko hahmo pitäisi suunnitella kokonaan alusta asti uusiksi. Tämä on erityisesti ongelmallista, jos uusi hahmo eroaa täysin alkuperäisestä, esimerkiksi, jos ihmisen tilalle pistettäisiin joku nelijalkainen hahmo. Mitä enemmän kertakäyttöistä sisältöä peliin tehdään tällä tavalla, sitä kömpelömpi projektista tulee ja siihen on vaikea tehdä muutoksia. Proseduraalisella generoinnilla kertakäyttöistä sisältöä voi tehdä nopeasti, halvemmin ja ketterämmin. Koska sisältö on muutenkin merkityksetöntä, proseduraalisuuden tuoma sieluttomuus sisällössä ei varsinaisesti tuota niin paljoa ongelmaa kuin merkityksekkäämpää sisältöä generoidessa. (Danc 2007.)

Uudelleenkäytettävän sisällön kohdalla proseduraalisuus tuo sisältöön lisävivahteita. Esimerkiksi roguelike-peleissä, joissa tutkitaan jokaisella peli kerralla luolia, proseduraalisuus tekee luolista aina erilaisia, jolloin tutkiminen ei ole niin tylsää. Luolien muotojen lisäksi

luolissa kohdatut viholliset ja löydetyt aarteet eivät ole samoja joten jokainen pelikerta tarjoaa erilaisia riskejä ja haasteita sekä palkintoja. (Danc 2007.)

Proseduraalisuus ei kuitenkaan tuo pelkkää hyvää. Sen lisäksi, että proseduraalinen sisältö on vaikea saada näyttämään ainutlaatuiselta – kuten Eliten kohdalla jo huomattiin – sen generointi on aikaa vievä prosessi. Yksinkertaisten ja yksittäisten objektien generoiminen onnistuu nopeasti, mutta jos alkaa generoida esimerkiksi kokonaista kaupunkia monien erilaisten rakennusten ja tiejärjestelmien kanssa, se vie helposti paljon aikaa. Esimerkiksi peli .kkrieger on pienikokoinen, mutta koska koko peli on kokonaan proseduraalisesti generoitu niin pelin käynnistyminen on hidasta. (Introversion Software 2007, 2.)

4 ESIMERKIT

Pelintekijät ovat jo pitkän aikaa tehneet pelejä joissa käytetään proseduraalista sisältöä. Tähän on valittu kolme esimerkkiä erilaisista proseduraalisista peleistä. Nämä kolme peliä ovat Elite, Spelunky ja Slaves to Armok: God of Blood Chapter II: Dwarf Fortress.

Elite on yksi ensimmäisiä proseduraalisia pelejä ja erityisesti se taitaa olla ensimmäinen joka käyttää dynaamista maailman luontia. Tämä oli erityisesti pelin ilmestymisaikana tärkeää, koska sen ajan tietokoneet eivät olleet tarpeeksi kykeneviä käsittelemään Eliteä, jos sen olisi tehnyt ilman proseduraalisia menetelmiä.

Spelunky on nykyaikaisempi esimerkki proseduraalisesta generoinnista. Se käyttää ajonaikaista kenttien generointia roguelike-pelien tyyliin ja yhdistää sen kaksiulotteiseen tasoloikkaan.

Slaves to Armok: God of Blood Chapter II: Dwarf Fortress on myös nykyaikaisempi peli. Se tunnetaan sen lähes kokonaisvaltaisesta ja yksityiskohtaiselta vaikuttavasta proseduraalisesta sisällöstä.

4.1 Elite

Elite on Ian Bellin ja David Brabenin vuonna 1984 tekemä avaruustaistelu- ja kaupankäyntipeli. Pelin julkaisi Acornsoft. Elitessä pelaaja ohjaa avaruusalusta (aikaisekseen) suhteellisen suuressa pelimaailmassa. Pelaajalla on vapaus toimia niin kuin itse näkee parhaaksi. Elite on yksi ensimmäisistä proseduraalisuutta käyttävistä peleistä.

Eliten maailma on niin suuri, että se ei ilman proseduraalista generointia olisi mahtunut vuonna 1984 käytetyille tallennusmedioille. Sen ajan tietokoneet eivät olisi myöskään pystyneet käsittelemään pelimaailmaa kokonaisuudessaan johon Elite hyödyntää dynaamista maailman luontia. Siemenluvuksi he valitsivat Fibonaccin lukujonoa jäljittelevän lukujonon. Fibonaccin lukujonoa ei tarvitse tallentaa muistiin vaan sitäkin voi generoida aina tarvittaessa lisää. Tarvitaan tietää vain ensimmäiset kaksi numeroa ja algoritmi luo tästä aina saman lukujonon. Eliten lukujono koostui pelkästään numeroista väliltä 0 - 9. Kun numerot

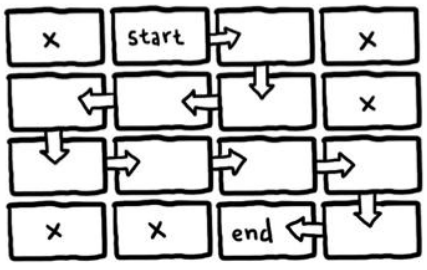
lukujonossa olivat suurempia kuin 9, niistä poistettiin aina kaikki ylimääräiset numerot. Esimerkiksi 16 on kuusi, 123 on kolme jne. (Spufford 2003.)

Braben ja Bell suunnittelivat aluksi, että Elitessä olisi jopa 2^{48} (eli noin 282 000 000 000 000, 282 biljoonaa) galaksia. Acornsoft totesi kuitenkin, että galaksien matemaattisuus selviäisi pelaajille ajan myötä niin suurella määrällä, kun pelaaja kohtaa vain vähän erilaisia alueita. Galaksien määrä päätettiin laskea kahdeksaan ja jokaisessa galaksissa olisi 256 tähtijärjestelmää. (Spufford 2003.)

4.2 Spelunky

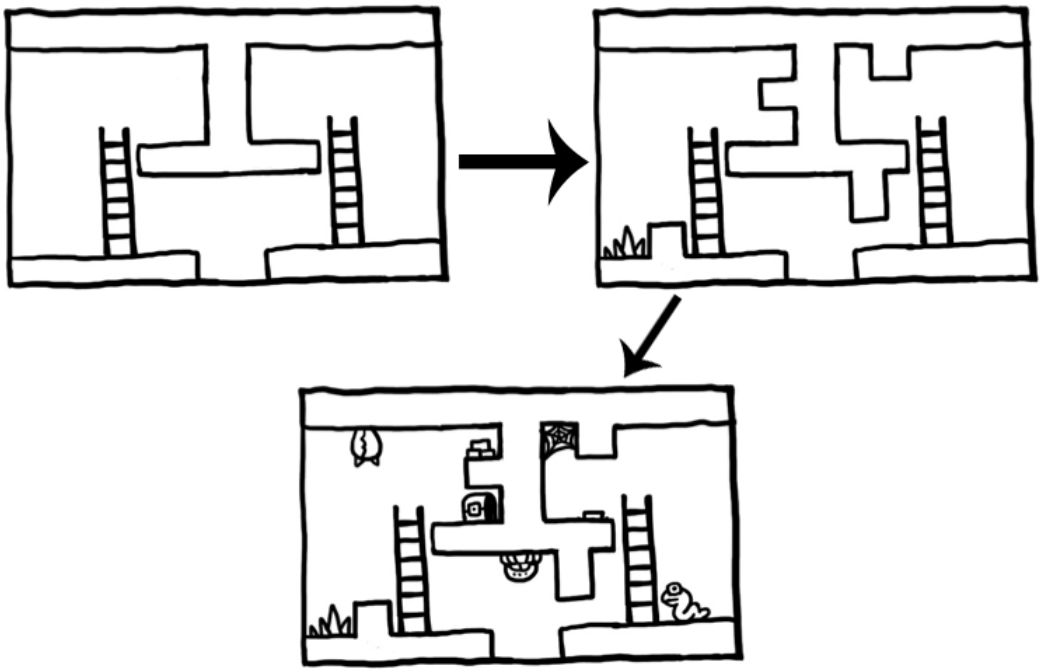
Spelunky (Kuvio 6.) on Derek Yun vuonna 2008 tekemä tasoloikka, jossa pelaaja etsii maanalaisista kaivoksista aarteita. Kaivokset ovat täynnä häijyjä otuksia ja ansoja sekä pelastettavia neitoja. Kaikkea tätä vastaan pelaajalla on käytössä pommeja, köysiä, ampuma- ja lyömäaseita ynnä muuta sellaista. Spelunky on hyvä esimerkki proseduraalisen sisällön generoinnin toimivuudesta kaksiulotteisissa tasoloikissa.

Spelunkyssa kentät generoidaan ajonaikana. Pelin kenttä koostuu niin kutsutuista huoneista. Yksi pelin kenttä on jaettu 16 huoneeseen joista jokainen valitaan valmiista mallipohjista. Ensimmäisenä huoneista sijoitetaan aloitushuone ja loppuhuone. Näiden välille luodaan reitti ja reitille luodaan huoneet (Kuvio 16.). Reitti rakennetaan niin, että pelaaja voi kulkea sen läpi ilman, että hänen tarvitsee käyttää tavaroita joita on rajallinen määrä.



Kuvio 16. Spelunkyn kentän generointia (Yu 2011)

Huoneita muokataan hieman satunnaisesti jotta niissä olisi vaihtelevuutta (Kuvio 17.). Reitti jonka generaattori luo on kulkureittien suhteen avoin niin, että pelaajan ei tarvitse käyttää yhtään tavaraa päästäkseen kentän loppuun. Kenttään generoidaan vihollisia, ansoja ja aarteita jotka tuovat haasteita ja vaihtelevuutta. Generaattori sijoittaa myös erilaisia erikoishuoneita, kuten kauppoja, uhrialttareita, onkaloita jne. Huoneet jotka eivät kuulu reittiin määritellään satunnaisesti erilaisiksi huoneiksi joihin ei ole suoraa pääsyä. (Yu 2011.)



Kuvio 17. Spelunkyn huoneen generoinnin vaiheita (Yu 2011)

4.3 Slaves to Armok: God of Blood Chapter II: Dwarf Fortress

Slaves to Armok: God of Blood Chapter II: Dwarf Fortress tunnetaan myös lyhyemmin nimellä Dwarf Fortress. Se on fantasia yksinpeli jossa pelaaja joko kontrolloi kääpiölinnoitusta (Fortress mode) tai vaihtoehtoisesti pelaa yksinäisellä kääpiöllä samaan tyyliin kuin roguelike-pelit (Adventure mode). Sitä ovat tehneet Zach ja Tarn Adams vuodesta 2002 ja ensimmäinen versio julkaistiin 2006. (Harris 2008 b.) Lisäksi pelissä on Legends-moodi jossa pelaaja voi tutkia luodun maailman historiaa. Pelaajan teot muissa pelimoodeissa näkyvät myös Legends-moodissa. (Meeks 2010.)

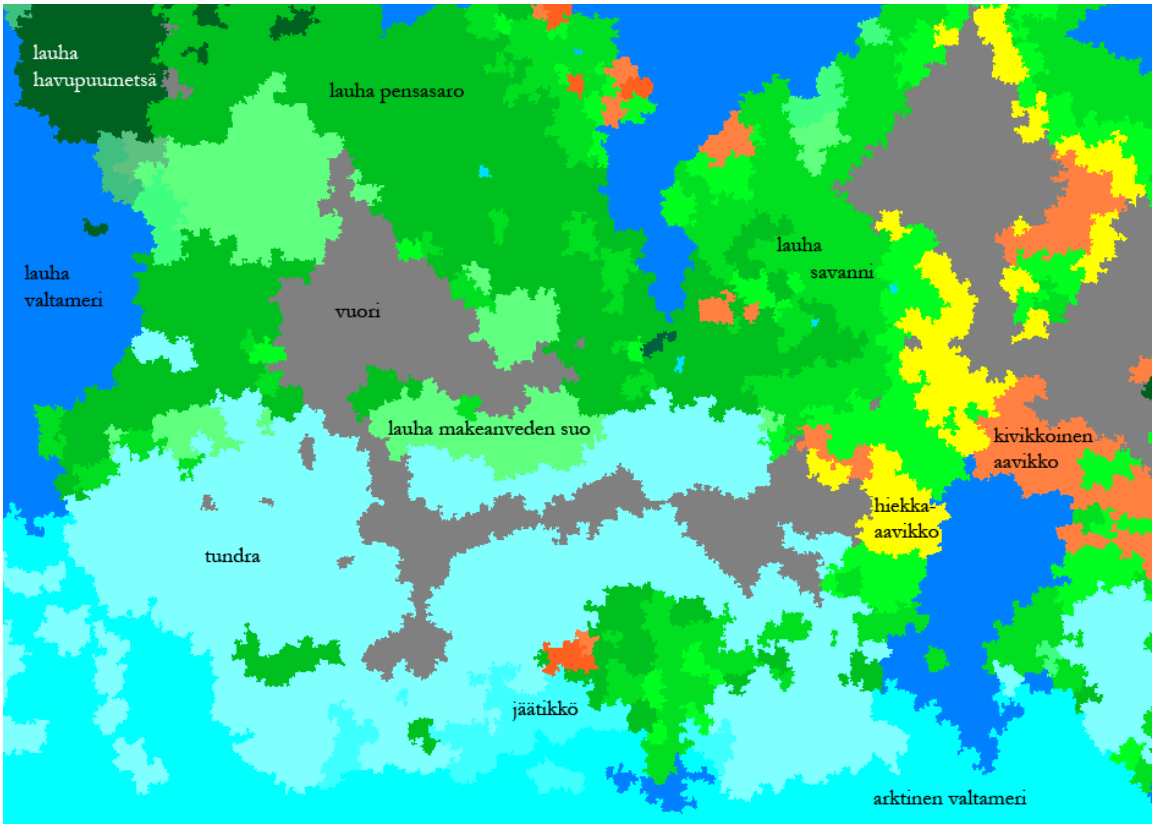
Dwarf Fortressia pidetään nykyään malliesimerkkinä proseduraaliselle sisällön generoinnille. Pelin maailmassa kaikki, kartasta sen historiaan, luodaan proseduraalisesti. Sen generaattori simuloi luonnon perusosia joiden perusteella maailmasta on saatu luonnonmukainen. (Harris 2008 a.)

Generaattori käyttää generoi-ja-testaa metodia maailman luonnissa, kuten kuviosta voi huomata. Kuvion tapauksessa generaattori on hylännyt 4 maailmaa ennen kuin se on aloittanut tarkemmin luomaan maailmaa. (Kuvio 18.)

Kuvio 18. Dwarf Fortressin maailmageneraattori (Adams 2011)

Ensimmäisenä pelissä luodaan sen maailma. Dwarf Fortress käyttää hyödyksi biomeja eli eloyhteisöjä. Biomien avulla maailmassa alueet vaihtelevat luonnollisesti niin, että maailmasta ei löydy äkkinäisiä luonnottomia siirtymiä erilaisten alueiden välillä. Biomit muodostuvat Dwarf Fortressissa ainakin lämpötilasta, sateesta, korkeudesta, vedenpoistosta ja suolaisuudesta. (Harris 2008 a.)

Kuvio 19 on tulostus Dwarf Fortressissa luodun maailman biomikartasta. Erilaisia biomeja kyseisessä maailmassa on 36. Niihin kuuluu esimerkiksi: vuori, lauha makeanveden järvi, trooppinen suolaisenveden järvi, tundra, jäätikkö, hiekka-aavikko, trooppinen savanni jne. (Adams 2011.)



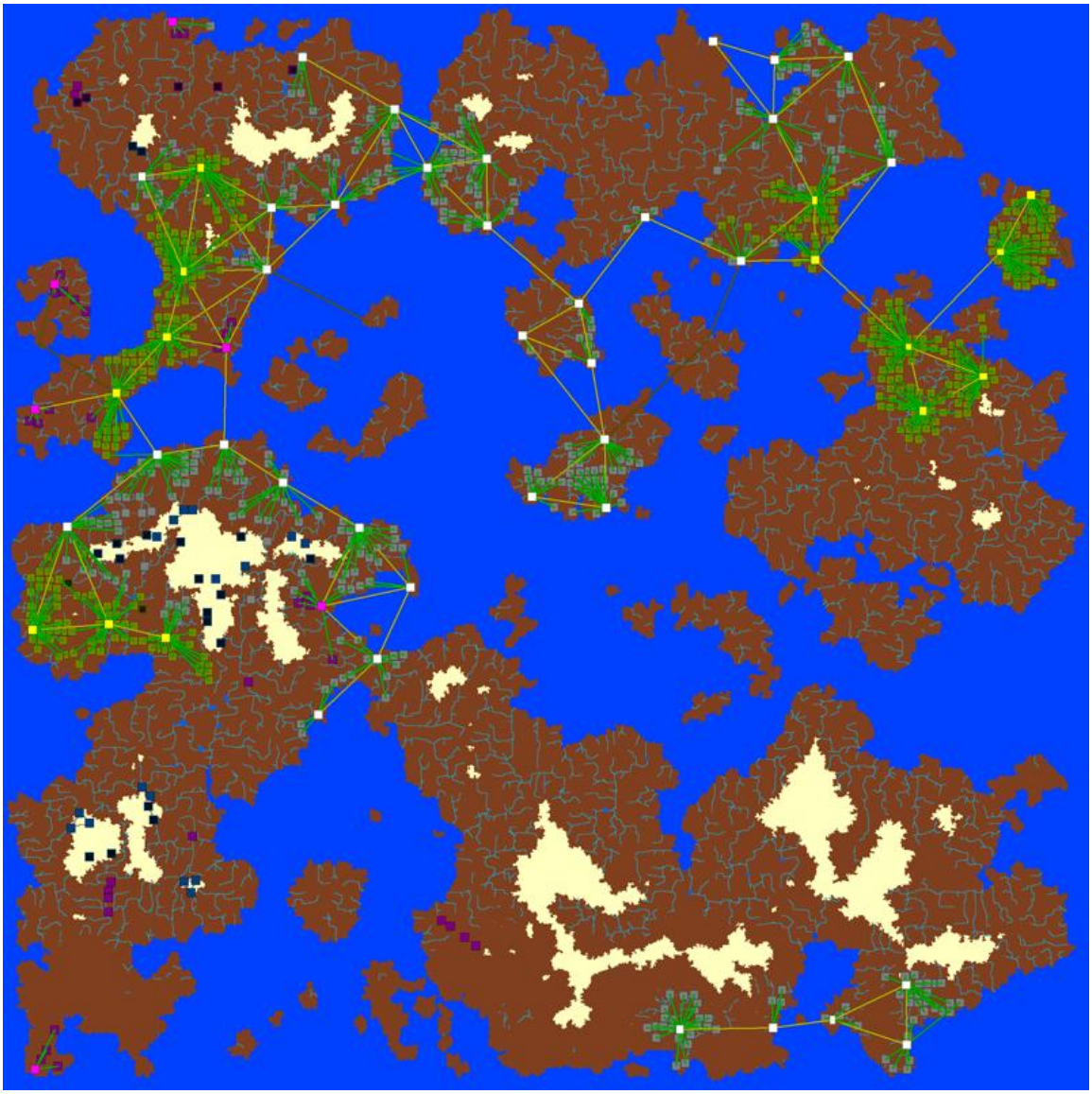
Kuvio 19. Otos Dwarf Fortressin biomikartasta (Adams 2011)

Biomien jälkeen generaattori käy läpi eroosiovaiheen. Generaattori luo vuoriin väliaikaisia jokia joita se vie vuoria alaspäin niin kauan kuin joki osuu mereen tai jää jumiin. Jumiin jäädessään joki yrittää kaivautua. Tämän jälkeen generaattori asettaa näihin oikeat joet. Jos joki ei pääse mereen asti niin generaattori joko pakottaa reitin mereen tai muodostaa järven. Isoimmille joille generaattori antaa nimet. (Harris 2008 a.)

Tässä vaiheessa maailmassa ei ole vielä elämää. Generaattori käy läpi maailman biomit ja muodostaa näistä alueita jotka se nimeää ja asettaa niihin kasvustoa ja eläimiä. Tämä toimii perustana maailman historialle (Kuvio 20.). (Harris 2008 a.)

Kuvio 20. Generaattori luomassa maailman historiaa (Adams 2011)

Maailman historiaa varten generaattori luo satunnaisesti maailmaan erilaisia otuksia ja kulttuureja. Generaattori sen jälkeen simuloi näiden eloa maailmassa luoden historiaa maailmalle. Esimerkiksi sodat rotujen välillä, kaupan käyntiä (Kuvio 21) jne. (Meeks 2010.) Esimerkkimaailmaan generaattori loi esimerkiksi muurahaisihmisiä, käärmeihmisiä, lepakkoihmisiä jne. Lisäksi mukana ovat vakiona ihmiset, kääpiöt, haltiat, hiidet ja koboldit. Näille roduille generaattori luo tarkempaa historiaa, kuten palvotut jumalat, hallitsijoiden ja tärkeimpien henkilöiden teot, valta-ajat, nimet jne. (Adams 2011.)



Kuvio 21. Kaupankäyntikartta (Adams 2011.)

5 PROJEKTI

Opinnäytetyön käytännönosuutena oli yhteinen peliprojekti kolmen muun opiskelijan kanssa. Tämän opinnäytetyön tekijän lisäksi projektissa oli mukana Otto Hietala, Sasu Kemppainen ja Markku Wiik. Hietala ja Kemppainen tekivät myös opinnäytetyöt omista osistaan. Hietala kehitti opinnäytetyönään pelimoottorin RoboDashiin Simple DirectMedia Layer ohjelmakirjastoa hyödyntäen (Hietala 2011). Kemppainen toimi projektin suunnittelijana ja johtajana, josta hän kertoo enemmän opinnäytetyössään (Kemppainen 2011). Peli ei olisi mitään ilman grafiikkaa, joten mukana oli myös Wiik graafikkona. Hän ei kuitenkaan tehnyt omasta osastaan opinnäytetyötä.

RoboDash on toimintatasoloikka jossa pelaajan on tarkoitus ohjata robotti mahdollisimman pitkälle vasemmalta oikealle. Pelaaja kulkee läpi tuhoutuneen maailman jossa hän kohtaa muita enemmän tai vähemmän vihamielisiä robotteja. Pelaaja pystyy myös vaihtamaan osia itsestään, jotka antavat uusia ominaisuuksia käytettäväksi kuten korkeampi hyppy tai erilaiset aseet. Pelistä, ja projektista yleisemmin, voi lukea tarkemmin Kemppaisen opinnäytetyössä.

Kuvio 22 on RoboDashin mockup eli näkemys siitä miltä peli näyttäisi. Pelaajan ohjattavissa oleva robotti näkyy suurin piirtein keskellä, vihreällä laatikkopäällä varustettuna. Muut robotit ruudulla ovat enemmän tai vähemmän vihamielisiä. Vasemmassa yläkulmassa näkyy prosenttina pelaajan sen hetkinen elinvoima. Oikean yläkulman lukema osoittaa pelaajan kulkeman matkan, jolla mitataan pelissä pelaajien paremmuus. Vasemmasta alalaidasta pelaaja näkee robottinsa varustuksen. Kentät koostuvat tuhoamattomasta maasta sekä tuhottavista roskapalikoista. Mockup-kuvassa tuhottavat roskapalikat näkyvät harmaina palikoina.

Kuvio 22. RoboDash mockup

RoboDash ja sen pelimoottori ovat avointa lähdekoodia ja ne ovat jaossa SourceForgessa. Projektin osoite: http://sourceforge.net/projects/vohweli/. Pelin kehityksestä pidettiin myös blogia joka löytyy osoitteesta: http://sourceforge.net/apps/wordpress/vohweli/.

5.1 Proseduraalinen sisältö

RoboDashin pääidea on sen loputon kenttä joka luodaan proseduraalisesti. Muissa tasoloikissa, joissa on loputon kenttä, on yleensä automaattinen liikkuminen ja pelaaja huolehtii vain hyppimisestä ja joissakin myös ampumisesta. RoboDashissa pelaaja ohjaa hahmoa niin kuin perinteisissä toimintatasoloikissa eli liikkumista ei ole automatisoitu. Tämän lisäksi pelissä oli tarkoitus käyttää proseduraalista generointia muihinkin asioihin, kuten robotin ulkonäköön, robotin nimeen ja vihollisten sekä lisävoimien sijoitteluun (joka tosin kuuluu osittain myös kentän generointiin).

5.1.1 Kenttä

RoboDashissa on vain yksi kenttä joka on teoriassa jokaisella kerralla erilainen ja jatkuu loputtomiin. Pelin ei kuitenkaan ole tarkoitus olla niin helppo, että pelaaja pystyisi pelaamaan sitä loputtomiin. RoboDashin kenttää ei heti alussa luoda kokonaan vaan sitä generoidaan vähän kerrallaan pelaajan edistymisen mukaan.

Kenttää ei generoida kokonaan alussa, koska teoriassa loputtoman kentän generointi veisi paljon muistia sekä sen generointiin menisi paljon aikaa. Kentän generointi ei myöskään pystyisi järkevästi reagoimaan pelaajan löytämiin sekä käyttämiin lisävoimiin. Tällöin generaattorin pitäisi ottaa huomioon kaikki mahdolliset tavat edetä joka taasen olisi paljon aikaa vievää.

Kun kenttiä generoidaan pelaajan etenemisen mukaan, pystyy peli reagoimaan siihen, mitä ominaisuuksia pelaajalla on. Generaattori voi näin luoda pelaajan sen hetkisille ominaisuuksille sopivia haasteita jolloin pelin pitäisi pysyä haastavana ja mielenkiintoisena kokoajan. Näin pelissä ei myöskään pitäisi tulla vastaan haasteita joita pelaajan on mahdotonta ohittaa.

RoboDash generoi kenttää vähän kerrallaan, kun pelaaja on tarpeeksi lähellä kentän sen hetkistä loppua. Generointi tapahtuu ajonaikana niin, että uutta aluetta luodaan vähitellen, jotta peli ei pysähdy edes lyhyeksi ajaksi. Generaattori ajetaan samassa säikeessä pelin kanssa jolloin generointiin käytetään oma osansa jokaisesta päivityskierroksesta.

Kentän generoinnissa luodaan välillä kohtia joissa pelaaja ei voi palata enää taaksepäin. Näitä kohtia käytetään hyödyksi muistinhallintaan. Kun pelaaja on liikkunut tällaisen kohdan ohi, poistetaan kaikki aikaisemmat osat kentästä. Näin varmistetaan se, että pelin kenttä ei koskaan vie liikaa muistia. Geometriana nämä kohdat ovat pääasiassa tarpeeksi korkeita esteitä, jotta pelaaja ei voi hypätä niiden ohi.

RoboDashin kenttä on variaatio dynaamisesta maailman luonnista, koska kenttä ei ole koskaan kokonaan ladattuna muistiin ja sitä generoidaan vähitellen lisää. Kenttä ei kuitenkaan ole deterministinen, koska se ei ole aina samanlainen vaikka olisi samat parametrit käytössä. Näin ollen kenttään hyödynnetään samalla ajonaikaista generointia.

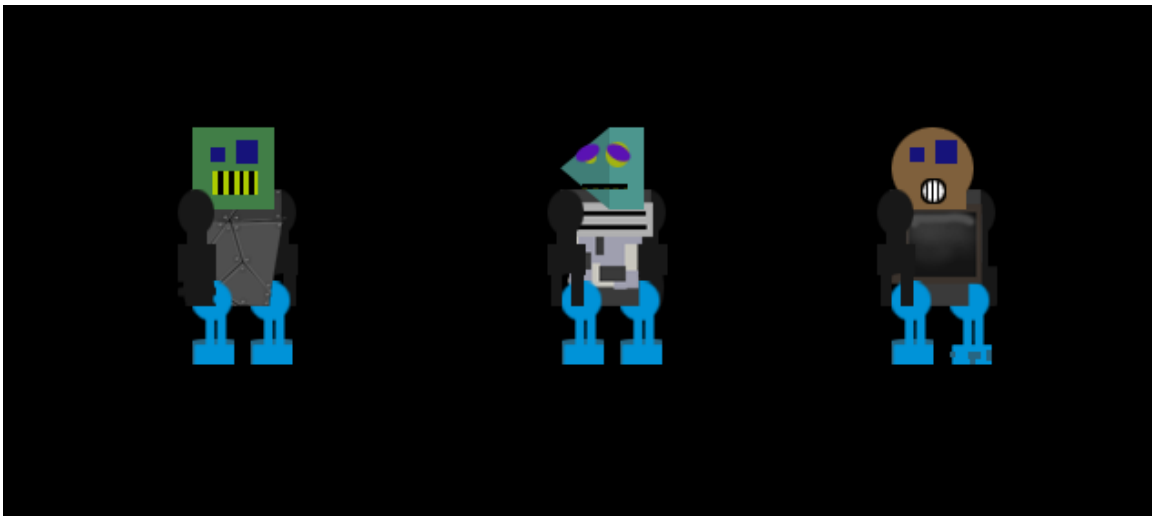
Kentässä on tarkoitus vaihdella alueita esimerkiksi aavikosta kivikkoiseksi jolloin peliin tulisi vaihtelevuutta. Vaihtelevuus vaikuttaa sekä alueen ulkonäköön että generaattorin tuottamiin

geometrisiin muotoihin. Näin aavikko olisi hieman tasaisempaa ja kivikkoinen alue olisi poukkoilevampaa ja vaatisi enemmän hyppelemistä.

Myös pelimaailman tausta generoidaan joka vaihtelee alueiden mukaan. Taustaa varten käytetään valmiita kuvia, joista valitaan mitä näytetään missäkin vaiheessa. Pelissä on joitakin erikoisempia artefakteja joita näkyy myös taustalla, mutta niiden sijoittelu on vähän enemmän ennalta päätetty. Esimerkiksi jokin tietty esine näkyy vain, kun pelaaja on edennyt pelissä tarpeeksi pitkälle. Tähän lisätään vaihtelua sijoittamalla esineen ilmestymiskohta satunnaisesti jollekin tietylle välille sen sijaan, että se ilmestyisi aina samassa kohdassa. Robotti

Jokaisen pelikerran alussa pelaajan robotti luodaan satunnaisesti. Robotilla vaihtelee sen pää, vartalo, jalat ja kummatkin kädet. Kädet voivat olla erilaisia, koska robotilla voi olla erilaiset aseet eri käsissä. Muuten kuin käsien ja jalkojen kohdalla, vaihtelut ovat enemmänkin vain kosmeettisia.

Itse RoboDashiin ei ehditty toteuttamaan erilaisia robotteja, mutta yksinkertaisella hahmogeneraattoriohjelmalla pystyi tarkastelemaan robotteja. Kuvio 23 on kolme esimerkkiä tämän generaattorin roboteista. Robottien grafiikat ovat näissä esimerkeissä vanhat testaukseen tarkoitetut grafiikat. Itse generointi tapahtuu yksinkertaisella satunnaisgeneraattorilla.



Kuvio 23. Esimerkki robotteja RoboDashista

5.1.2 Viholliset

RoboDashin viholliset ovat myös robotteja. Niitä on erilaisia ja jokainen robotti käyttäytyy eri tavalla. Vihollisrobotteja ei koota samalla tavalla osista kuten pelaajaa vaan kaikki samantyyppiset viholliset ovat samannäköisiä. Esimerkkivihollisia ovat:

- lentävä robotti joka pudottaa pommeja
- paikallaan pysyvä robotti joka ampu useita pieniä ammuksia ylöspäin
- vartijarobotti joka liikkuu edestakaisin määrätyllä alueella
- iso, rauhallinen robotti joka ei hyökkää, mutta sillä on käytössä iso pommi, jonka robotti räjäyttää, jos pelaaja ampuu sitä.

Jotkut robotit eivät välttämättä ole vihamielisiä. Viholliset generoidaan pelimaailmaan kentän generoinnin yhteydessä. Generaattori tarvitsee erilaisia sääntöjä vihollisten sijoitteluun, jotta niitä ei sijoitella pelimaailmaan ihan miten sattuu. Vihollisten sijoittelun pitää myös olla mukautuva siinä määrin, että se osaa luoda sopivia haasteita pelaajan sen hetkisille ominaisuuksille. Lisäksi generaattorin pitää osata generoida pelaajan etenemisen myötä pelistä haastavampi niin, että peli ei ole alussa liian vaikea, mutta pelin pitää muuttua vaikeammaksi mitä pidemmälle pelaaja pelissä pääsee. Tätä voidaan ajatella eräänlaisena dynaamisena järjestelmänä.

5.1.3 Esineet

Vihollisroboteilta, kun niitä tuhoaa, voi pudota erilaisia esineitä pelaajan käytettäväksi. Esineisiin kuuluu jalkoja, käsiä, energialisiä ja pattereita robotin kehoon. Jalat vaihtelevat pelaajan liikkumista. Esimerkiksi renkaalla liikkuminen on erilaisempaa kuin jaloilla. Vaihdeltavat kädet ovat erilaisia aseita. Aseet vaihtelevat erilaisista lähitaisteluaseista ampumisaseisiin. Energialisä palauttaa pelaajan menettämää energiaa. Patteri antaa pelaajalle erilaisia lisäominaisuuksia kuten tuplahyppy ja maksimi energiamäärän lisäys.

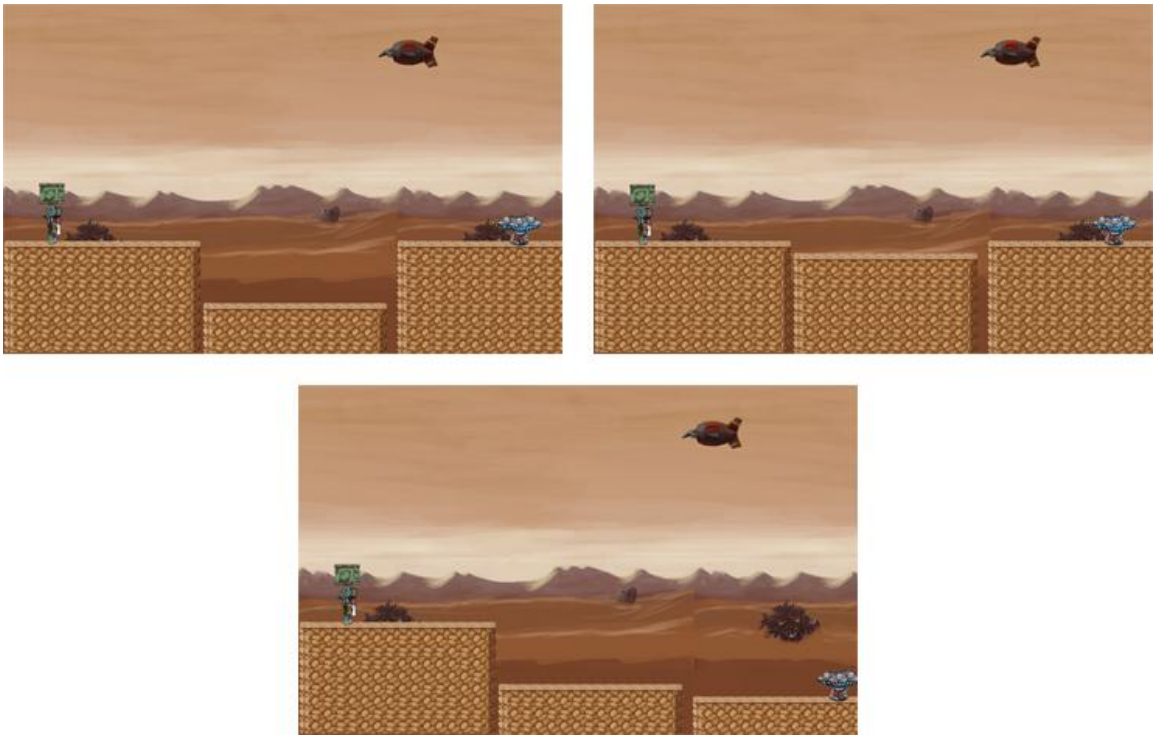
Esineet tippuvat vihollisilta, kun pelaaja tuhoaa niitä. Esineiden tippuminen on satunnaista eli siinä käytetään pelin sisäisten entiteettien instantiointia. Jotkut viholliset eivät välttämättä

aina pudota mitään. Esineiden antamat ominaisuudet vaihtelevat satunnaisesti. Tähän vaikuttaa myös se, että keneltä esine on pudonnut ja kuinka pitkällä pelaaja on.

5.2 Pelin algoritmit

Kentän generoinnissa käytetään kahta eri algoritmia. Ensisijainen algoritmi on rytmipohjainen kentän generointi ja toissijainen on keskipisteen siirto algoritmi. Rytmipohjaisella generoinnilla luodaan tasoloikkapelimäinen perusta pelin kentälle. Keskipisteen siirto algoritmilla luodaan peliin luonnollisen näköisiä romukasoja, jotka rikkovat välillä perinteistä palikkamaista ulkonäköä.

Kuvio 24 on havainnollistettu miltä sama rytmi näyttää RoboDashissa kolmella eri geometrialla. Näkyvissä oleva rytmi koostuu kahdesta hypystä. Vihollisten sijoitteluun ei generaattori vaikuttanut. Ensimmäinen alusta, jolla pelaaja seisoo, on sijoitettu aina samaan kohtaan, joka toimii eräänlaisena aloitusalueena.

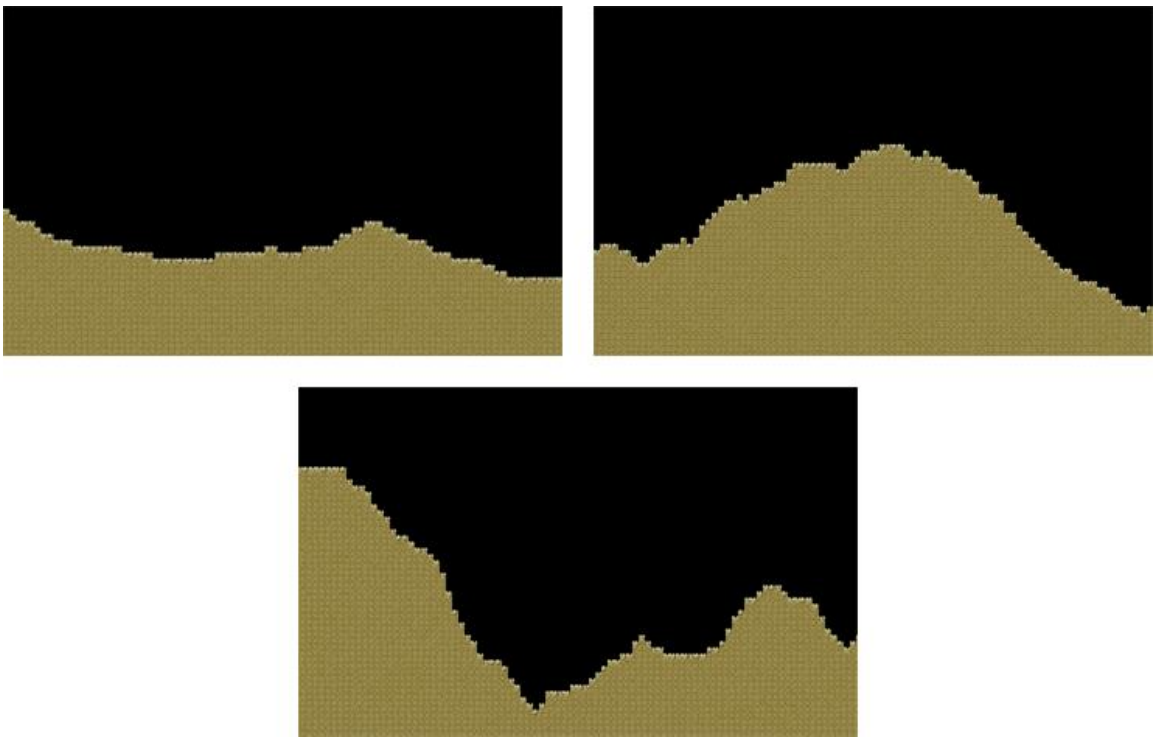


Kuvio 24. RoboDash ja rytmipohjainen generointi

Alueiden muutoksien lisäksi vaihtelevuutta tuovat satunnaiset romukasat. Tätä varten RoboDash käyttää keskipisteen siirto algoritmia. Välillä peliin generoidaan vaiheita jolloin

romuista on muodostunut isoja kasoja, jopa vuoria, kentälle. Keskipisteen siirto algoritmin avulla romukasoista saa tarpeeksi luonnollisen kaltaisia. Pelin romut ovat tuhottavia palikoita joten varsinaisesti romukasojen oikeellisuutta ei tarvitse tarkistaa erikseen. Koska vaikka pelaaja ei pääsisi hyppimällä romukasojen ohi, on hänellä käytössä aina jokin ase jolla romut voi tuhota.

Kuvio 25 havainnollistaa kolmella eri tulostuksella RoboDashia varten tehtyä keskipisteen siirto algoritmia erilaisilla karkeusarvoilla. Ensimmäisessä kuvassa (vasen ylhäällä) arvo on asetettu 1000:een. Toisessa kuvassa (oikealla) arvo on 2000 ja viimeisessä kuvassa (alhaalla) 3000. Itse algoritmia ei täysin ehditty implementoimaan itse peliin joten kuviossa näkyy vain testiohjelman tulostuksia.



Kuvio 25. Esimerkki RoboDashin keskipisteen siirto algoritmista

6 POHDINTA

Proseduraalinen sisällön generointi on mielenkiintoinen osa-alue pelien tekemisessä. Sitä voidaan soveltaa moniin eri asioihin, kuten kenttien luontiin satunnaisesti ajonaikana ja ennakkoon esimääritetyillä niin kutsutuilla siemenluvuilla. Laajamittaisemmin vielä nykyään proseduraalista generointia käytetään enimmäkseen vain kenttien ja erilaisten esineiden luontiin. Jotkut ovat kuitenkin uskaltautuneet yrittää käyttää proseduraalisuutta erikoisemmissakin asioissa, kuten luomaan keskusteluihin epälineaarisuutta ja dynaamisuutta. Nämä uskaliaat yritykset ovat niitä jotka vievät alaa eteenpäin. Ei pidä kuitenkaan muita väheksyä, koska niilläkin osa-alueilla kehitystä tarvitaan.

Erityisesti pienemmille pelifirmoille proseduraalinen sisällön generointi on hyvä tapa luoda sisältöä, koska heillä ei ole samoissa määrin rahaa luoda käsin suuria määriä sisältöä, kuin isoilla firmoilla. Isotkin firmat hyötyisivät paljon proseduraalisen sisällön generoinnista, koska vaikka heillä varaa olisi, ei turhaan kannata samasta maksaa liikaa.

Proseduraalista sisällön generointia ei kuitenkaan pidä ajatella jonkinlaisena pelastajana, joka ratkaisee kaikki pelin sisällön luontiin liittyvät ongelmat. Proseduraalisesti generoitu sisältö voi helposti vaikuttaa keinotekoiselta. Algoritmeja voi aina tietenkin kehittää, mutta jos sisältöä ei jotenkin rajaa käsin, se väistämättä jossain vaiheessa alkaa näyttää matemaattisesti luodulta.

Käsintehty sisältö on hyvä yhdistää proseduraalisesti generoidun sisällön kanssa. Ihminen kykenee vielä luomaan paljon mielekkäämpiä ja mieleenpainuvampia kohtauksia, tilanteita ja paikkoja kuin matemaattiset algoritmit. Proseduraalisesti generoimalla voidaan kuitenkin helposti luoda kaikki muu sisältö, kuten vaikka pelimaailma kokonaisuudessaan ja lisätä näin vaihtelevuutta peliin. Pelintekijä voi sitten itse lisätä sinne käsintehtyä sisältöä. Proseduraalisella generoinnilla voi myös pelien tapahtumiin tuoda vaihtelevuutta jolloin pelin uudelleenpeluuarvo kasvaa. Jos pelaaja joutuu yrittämään samaa kohtausta monta kertaa peräkkäin, tekee vaihtelevuus uudelleen yrittämisestä mielekkäämpää.

Tulevaisuudessa proseduraalisessa generoinnissa käytettävät algoritmit kehittyvät edelleen. Tällöin proseduraalisesti saadaan tehdyksi monimutkaisempiakin asioita ja pelien juoniin voidaan saada enemmän dynaamisuutta ja epälineaarisuutta. Pelien tarinat ovat olleet tähän

mennessä hyvin elokuvamaisen lineaarisia, mutta pelien pitäisi pystyä parempaan, koska pelit ovat luonnostaan interaktiivisia.

Projekti

RoboDashin tekeminen alkoi heinäkuussa 2010. Kyseessä oli ryhmän ensimmäinen yhteinen peliprojekti joka tehtäisiin alusta asti yhdessä ja sen pitäisi sopia jokaisen oman opinnäytetyön aiheeseen. Proseduraalinen sisällön generointi rajasi eniten mahdollisia toteutettavia pelejä. Alustavasti päätimme myös aikatauluttaa projektin kolmelle kuukaudelle.

RoboDash itsessään on melko yksinkertainen peli: liiku vasemmalta oikealle mahdollisimman pitkälle väistellen ja tuhoten robotteja. Näin ollen se vaikutti semmoiselta, että sen voisi helposti toteuttaa kolmessa kuukaudessa. Mukana oli kuitenkin sen verran uusia asioita, ainakin koodillisesti, että pelin tekeminen oli hidasta.

Ohjelmoijia projektissa oli yhteensä kaksi. Tämä ei muuten olisi ollut iso ongelma, mutta koska olimme tekemässä opinnäytetyönä omia osiamme, se velvoitti kumpaakin keskittymään omaan osa-alueeseensa. Mutta peli kaipasi myös itse pelattavaa ja koska mukana ei ollut kolmatta ohjelmoijaa tekemään nimenomaan pelimekaniikkaa, joutuivat kummatkin ohjelmoijat jakamaan myös nämä tehtävät.

Alkuperäinen kolmen kuukauden aikataulu venyi lopulta noin viiteen kuukauteen. RoboDashissa ei vielä siinä vaiheessakaan ollut kuin ihan perusosat paikallaan. Rytmipohjainen kentän generointi toimi jossain määrin. Se osasi generoida kenttää lisää vähitellen, kun pelaaja liikkui kentässä pidemmälle. Tässä vaiheessa generaattori loi lisää rytmejä edellisten rytmien perään. Generaattori katsoi rytmeistä, missä kohti hypyt alkoivat ja näiden perusteella loi kenttään sopivan kokoisiaasdasdas palasia (Kuvio 26.).

Kuvio 26. RoboDash

Geometriageneraattori loi rytmeistä vain yhdenlaisia esteitä hypyille. Maapalikoiden korkeutta vain vaihdeltiin fysiikkarajoitteiden mukaan. Kuiluja geometriageneraattori ei vielä luonut. Pelin kenttä koostui yksinkertaisista nelikulmioista joissa kaikissa oli samanlaiset grafiikat.

Kun RoboDashia oli tehty noin viisi kuukautta, lopetettiin sen tekeminen. Projekti oli jo venynyt alkuperäisestä aikataulusta ihan kiitettävästi ja jossakin vaiheessa on vain osattava lopettaa.

Aluksi kentän generoinnissa oli käytössä pelkästään keskipisteen siirto algoritmi, mutta sen heikkous mielenkiintoisena tasoloikkakenttien generaattorina oli nähtävissä nopeasti. Tätä varten piti etsiä jokin toinen ratkaisu. Rytmipohjainen generointi löytyikin nopeasti ja sen toimivuus teoriassa oli selkeää. Keskipisteen siirto algoritmia ei kuitenkaan heitetty roskakoriin vaan sitä hyödynnettiin kentän generoinnissa muihin osiin.

Rytmipohjaisen generaattorin toteutus osoittautui hankalammaksi kuin oli aluksi oletettu. Sen verran mitä generaattoria peliin kuitenkin saatiin aikaiseksi, se osoitti olevansa suhteellisen toimiva ratkaisu proseduraalisten kenttien generointiin tasoloikkapeliä varten. Yleistä proseduraalisen generoinnin kanssa onkin, että itse algoritmien kehittämiseen voi mennä paljon aikaa. Ainakin selkeää oli, että rytmipohjainen ratkaisu oli lähempänä toimivia

kenttiä kuin vain satunnainen maaston generointi, koska tasoloikkapelejä – erityisesti hyviä – pelatessa voi huomata, että niiden kentissä on selkeää rytmitystä jonka perusteella kentät on rakennettu.

Pelien kentät toimivat tiettyjen sääntöjen mukaan joilla kentistä tehdään mielenkiintoisia. Tällä periaatteella on ymmärrettävää, että generoinninkin pitäisi noudattaa ennalta määrättyjä sääntöjä, sen sijaan, että generoitaisiin vain satunnaisesti. Tämä auttaa myös siinä, että kenttiä ei välttämättä tarvitse erikseen testata niiden toimivuutta varten. Mutta proseduraalista sisältöä suunniteltaessa on mietittävä, että miten paljon tähän on tarvetta. Näin on mahdollista luoda vähitellen, pelaajan etenemisen mukaan, generoitava loputon kenttä. Jos kenttä generoidaan vain satunnaisesti, kuten esimerkiksi korkeuskartan generointi, pitää sen oikeellisuus tarkistaa, jotta pelaaja pystyy pelaamaan sitä.

Loputonta kenttää tehdessä on tärkeää olla mukana myös hyvä muistinhallinta. Tämän tarpeellisuutta on tietenkin hyvä miettiä, koska jos pelaaja ei pelissä edes voi päästä kovin pitkälle, ei muistia varsinaisesti tarvitse erityisemmin hallita. RoboDashin kohdalla kuitenkin muistin hallintaa olisi tarvittu. Kentän generointi on hyvä jakaa sopiviin palasiin joita on helppo lisätä ja poistaa. Tämä sopii hyvin rytmipohjaisen generoinnin kanssa, koska rytmiryhmiä voi helposti hyödyntää muistinhallinnassa.

LÄHTEET

Adams, T. & Z. 2011. Slaves to Armok: God of Blood Chapter II: Dwarf Fortress. Versio 0.31.21. Saatavilla: http://www.bay12games.com/dwarves/.

Brudvig, E. 2007. More guns than could ever imaging. Yahoo! Games. Saatavilla: http://videogames.yahoo.com/ps3/borderlands/preview-1071343. (Luettu 28.4.2011).

Danc. 2007. Content is Bad. Lost Garden. http://lostgarden.com/2007/02/content-is-bad.html. (Luettu 28.4.2011).

Doull, A. 2008 a. The Death of the Level Designer: Procedural Content Generation in Games - Part One. ASCII Dreams. http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html. (Luettu 29.4.2011).

Doull, A. 2008 b. The Death of the Level Designer: Procedural Content Generation in Games - Part Two. ASCII Dreams. http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural_14.html. (Luettu 29.4.2011).

Doull, A. 2010. Proceduralism: Part One (Revision). ASCII Dreams. http://roguelikedeveloper.blogspot.com/2010/05/proceduralism-part-one-revision.html. (Luettu 29.4.2011).

Elias, H. 2003. Perlin Noise. The good-looking textured light-sourced bouncy fun smart and stretchy page. http://freespace.virgin.net/hugo.elias/models/m_perlin.htm. (Luettu 13.4.2011).

Ensemble Studios. 2005. Age of Empires III. Saatavilla demo muodossa: http://www.ageofempires3.com/age3/. (Luettu 26.4.2011).

Harris, J. 2008 a. Interview: The Making Of Dwarf Fortress. Gamasutra. http://www.gamasutra.com/view/feature/3549/interview_the_making_of_dwarf_.php?page=7. (Luettu 29.4.2011).

Harris, J. 2008 b. Interview: The Making Of Dwarf Fortress. Gamasutra. http://www.gamasutra.com/view/feature/3549/interview_the_making_of_dwarf_.php?page=2. (Luettu 29.4.2011).

Hietala, O. 2011. Developing Game Engine with Simple DirectMedia Layer. Kajaanin ammattikorkeakoulu. Opinnäytetyö.

InfinitySupport. 2010. Infinity Tech Demo Video 2010. Saatavilla: http://www.youtube.com/watch?v=fO7XhaTGDYg. (Luettu 26.4.2011).

Procedural Arts. 2008. InteractiveStory.net. http://www.interactivestory.net/. (Luettu 28.4.2011).

Introversion Software. 2005. Darwinia. Saatavilla: http://www.introversion.co.uk/darwinia/about/screenshots.html. (Luettu 26.4.2011).

Introversion Software. 2007. Procedural Content Generation. Gamasutra. Saatavilla: http://www.gamecareerguide.com/features/336/procedural_content_.php. (Luettu 29.4.2011).

Johnston, C. 2009. a. Procedural Generation 1. Proper Undead. Saatavilla: http://properundead.com/2009/03/cave-generator.html. (Luettu 29.4.2011).

Kemppainen, S. 2011. Pelisuunnittelu pienessä peliprojektissa. Kajaanin ammattikorkeakoulu. Opinnäytetyö.

Kohler, C. 2008. Spore Creator Launches June 17. Wired. Saatavilla: http://www.wired.com/gamelife/2008/04/spore-creature/. (Luettu 4.4.2011).

Mandelbrot, B.B. 1982. The Fractal Geometry of Nature. W.H. Freeman and Company.

Martz, P. 1997. Generating Random Fractal Terrain. Game Programmer. http://www.gameprogrammer.com/fractal.html. (Luettu 7.4.2011).

Meeks, E. 2010. Procedural Humanities - An Interview With Tarn Adams, Creator Of Dwarf Fortress. http://www.hastac.org/blogs/elijahmeeks/procedural-humanities-interview-tarn-adams-creator-dwarf-fortress. (Luettu 29.4.2011).

MobyGames. 2000. Rogue. Saatavilla: http://www.mobygames.com/game/dos/rogue/screenshots/gameShotId,7951/. (Luettu 28.4.2011).

Olsen, J. 2004. Realtime Procedural Terrain Generation. Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. http://perso.telecom-paristech.fr/~bloch/P6/PRREC/terrain_generation.pdf. (Luettu 21.4.2011).

Perlin, K. 1985. An Image Synthesizer. SIGGRAPH 85 Computer Graphics. ACM. 19 (3), 287 - 296.

Perlin, K. 1999. Making Noise. Noise Machine. http://www.noisemachine.com/talk1/21.html. (Luettu 11.5.2011).

Procedural, Inc. 2011. Swiss Village for Masdar City. http://www.procedural.com/showcases/showcase/masdar-city.html. (Luettu 4.5.2011).

Shankel, J. 2000. Fractal Terrain Generation – Midpoint Displacement. Game Programming Gems. Charles River Media.

Smith, G., Treanor, M., Whitehead, J., Mateas, M. 2009. Rhythm-Based Level Generation for 2D Platformers. Saatavilla: http://www.sokath.com/papers/smith-fdg09.pdf (Luettu 29.4.2011).

Spufford, F. 2004. Backroom Boys: The Secret Return of the British Boffin. Faber. http://www.guardian.co.uk/books/2003/oct/18/features.weekend. (Luettu 29.4.2011).

Togelius, J., Yannakakis, G. Stanley, K., Browne, C. 2010. Search-based Procedural Content Generation. Saatavilla: http://julian.togelius.com/Togelius2010Searchbased.pdf. (Luettu 29.4.2011).

Wikipedia. 2008. Rhythm. http://en.wikipedia.org/wiki/Rhythm. (Luettu 11.4.2011).

Wikipedia. 2011 a. Fractal. http://en.wikipedia.org/wiki/Fractal. (Luettu 5.4.2011).

Wikipedia. 2011 b. Procedural Generation. http://en.wikipedia.org/wiki/Procedural_generation. (Luettu 8.4.2011).

Yu, D. 2011. The Full Spelunky on Spelunky. Make Games. http://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky. (Luettu 29.4.2011)