Proseduraalinen pelisisällön luominen

Jani Liikkanen

Opinnäytetyö
Lokakuu 2015
Tietojenkäsittely
Ohjelmistotuotanto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

LIIKKANEN, JANI:
Proseduraalinen pelisisällön luominen

Opinnäytetyö 33 sivua
Lokakuu 2015

Opinnäytetyön tavoitteena oli tutustua erilaisiin proseduraalisen sisällöntuotannon keinoihin ja hyödyntää niitä Iceflake Studiosille luotavassa sisällönluontityökalussa. Yritys tarvitsi työkalua helpottamaan ja nopeuttamaan työskentelyä Ice Lakes -pelinsä kanssa. Työkalulla on tarkoitus luoda pelin sisäisiä karttoja lähes valmiiksi, jonka jälkeen pelintekijä pystyy pienellä viimeistelyllä hiomaan ne pelattavaan kuntoon.

Työ jakautui kahteen osaan: eri metodeihin tutustumiseen sekä niiden käyttämiseen käytännön toteutuksessa. Opinnäytetyö käsitteli useita proseduraalisia metodeja, joista valittiin käyttöön työhön parhaiten soveltuvat menetelmät. Kaikista tutkituista tavoista ei ollut käytännön työn kannalta hyötyä, vaan ne soveltuivat paremmin erilaisiin tarpeisiin.

Opinnäytetyötä varten luotu karttageneraattori vastasi toimeksiantajan odotuksia. Sillä pystyy luomaan vaihtelevan näköisiä maastoja, jotka näyttävät luonnollisilta, ja niitä on mahdollista hioa persoonallisiksi. Generaattoria olisi mahdollista hioa vielä monipuolisemmaksi useammilla eri asetusvaihtoehdoilla.

Asiasanat: proseduraalisuus, sisällönluonti

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Software Development

LIIKKANEN, JANI:
Procedural Content Generation in Video Games

Bachelor's thesis 33 pages
October 2015

The aim of this thesis was to study different kinds of procedural content generation methods, and use the gained knowledge to build a map generation tool. The tool was built for a small game company Iceflake Studios. They needed the tool to speed up development of their upcoming ice fishing game Ice Lakes. The map generation tool automatically creates maps that resemble closely what the player will see in the finished game, and the developer can tweak and improve the results.

Work was divided into studying the theory and methods of procedurality, and creating the map generator tool. This thesis covers a large amount of procedural content generation methods. Since different methods work for different types of goals, some of the procedural methods that were covered in the thesis were not used in the map generator.

The outcome of the map generator satisfied the needs of the Iceflake Studios. It can create natural looking landscapes with a great deal of variation, and offers the possibility of manual polishing. The generator may still be improved by adding more settings for the user to tweak.

Key words: procedurality, content generation

SISÄLLYS

LYHENTEET JA TERMIT

Asset	Unity-editorissa assetit ovat eräänlaisia malleja, joista luodaan peliobjekteja, tai käytetään niiden osana. Esimerkiksi 3D-mallit, äänet, tekstuurit tai kooditiedostot.
Kenttä	Kentällä tarkoitan työssäni yhtä kartta-aluetta, missä pelaaja voi pelihahmollaan liikkua.
Partikkeli	Yleisin erilaisiin efekteihin, kuten esimerkiksi tuli, salama, kipinät, käytetty yksinkertainen kuva. Partikkelit ovat tietokoneelle mahdollisimman kevyesti toteutettu, jotta niitä pystytään näyttämään satoja tai tuhansia kerrallaan.
Peliobjekti	Mikä tahansa pelin sisäinen, näkyvä tai näkymätön, yksilöllinen esine tai asia. Esimerkiksi pelihahmo, puu, pistemittari tai kamera.
Pikseli	Tietokonegrafiikassa kuvat koostuvat pikseleistä. Yhdellä pikselillä on väri- ja läpinäkyvyysarvot.
Polygoni	Polygonit ovat 3D-mallin näkyviä pintoja. Polygonit muodostuvat kolmesta tai useammasta verteksistä, jotka muodostavat pinnan kulmapisteet.
Tekstuuri	Kuvatiedosto peligrafiikasta. Tekstuuri voi olla esimerkiksi selkeä kuva, jota käytetään sellaisenaan pelissä, tai se voi sisältää osia useista peliobjektien käyttämästä grafiikasta.
Terrain	Kolmiulotteisessa pelissä käytettävä yksinkertainen maastoobjekti.
Verteksi	3D-mallit koostuvat pisteistä kolmiulotteisessa tilassa. Näitä pisteitä sanotaan vertekseiksi, ja ne sisältävät tiedon omasta sijainnistaan.

1 JOHDANTO

Pelinkehityksessä yksi suurimpia osa-alueita on pelin sisällön luominen. Sisällöllä tarkoitan kaikkia pelihahmoja, esineitä, grafiikoita, ääniä, peliympäristöjä ynnä muita pelissä näkyviä ja kuuluvia asioita. Kaikkien näiden tekeminen vaatii paljon aikaa, mikä on niin pienillä kuin isoillakin tiimeillä vähissä. Mikäli sisältöä pitää luoda paljon, tarvitaan siihen myös hyviä apuvälineitä.

Opinnäytetyön toimeksiantaja Iceflake Studios on pieni Tamperelainen peliyritys. Heillä on monta julkaistua mobiilipeliä, mutta nyt he ovat siirtymässä muillekin alustoille. Yrityksen uusi peli, Ice Lakes, on kolmiulotteinen pilkkimispeli, joka olisi tarkoitus julkaista vuoden 2015 lopussa. Koska studio on pieni, tarvitsivat he apua luomaan peliin karttoja. Minulla oli hieman aiempaa kokemusta proseduraalisesta generoinnista, sekä mielenkiintoa tutkia sitä lisää, ehdotin että tutkisin aihetta, ja tekisin työkalun helpottamaan sisällön tuottamista.

Työn tavoitteena on perehtyä erilaisiin sisällöntuottotapoihin, ja hyödyntää opittuja asioita, jotta toimeksiantajalle saadaan mahdollisimman toimiva ja monipuolinen apuväline työhönsä. Käsittelen raportissani proseduraalisen sisällönluonnin eri keinoja, jotta lukija voi saada pienen käsityksen siitä, kuinka monikäyttöisiä eri tekniikat voivat parhaimmillaan olla. Tarkoituksena on suunnitella ja toteuttaa algoritmeja, joiden avulla pystytään luomaan luonnollista, sekä pelillisesti toimivaa maastoa.

Työni aiheesta ei ole juurikaan kirjallista tietoa, minkä vuoksi joudun turvautumaan esimerkiksi blogikirjoituksiin, tutoriaaleihin sekä tieteellisiin artikkeleihin. Itse työkalun toteutukseen tarvitsee paljon suunnittelua, sekä eri algoritmien kokeilua ja hiomista. Saan myös apua toimeksiantajalta sekä ohjelmointiin liittyvissä asioissa, että työkalun tuottaman lopputuloksen toimivuudessa pelin kannalta. Saan myös käyttööni peliin tulevat assetit, kuten 3D-mallit ja tekstuurit.

2 SISÄLLÖN LUOMINEN

Videopeliin sisältö luodaan useimmiten suoraan muilla ohjelmilla. Esimerkiksi pelihahmot ja esineet tehdään 3D-mallinnus- tai kuvankäsittelyohjelmalla. Näin luodaan usein kokonaisia objekteja, joita käytetään sellaisena kuin ne ovat. Esimerkiksi kenttäsuunnittelija asettelee objektit käsin paikoilleen, ja rakentaa näin kenttiä peliin.

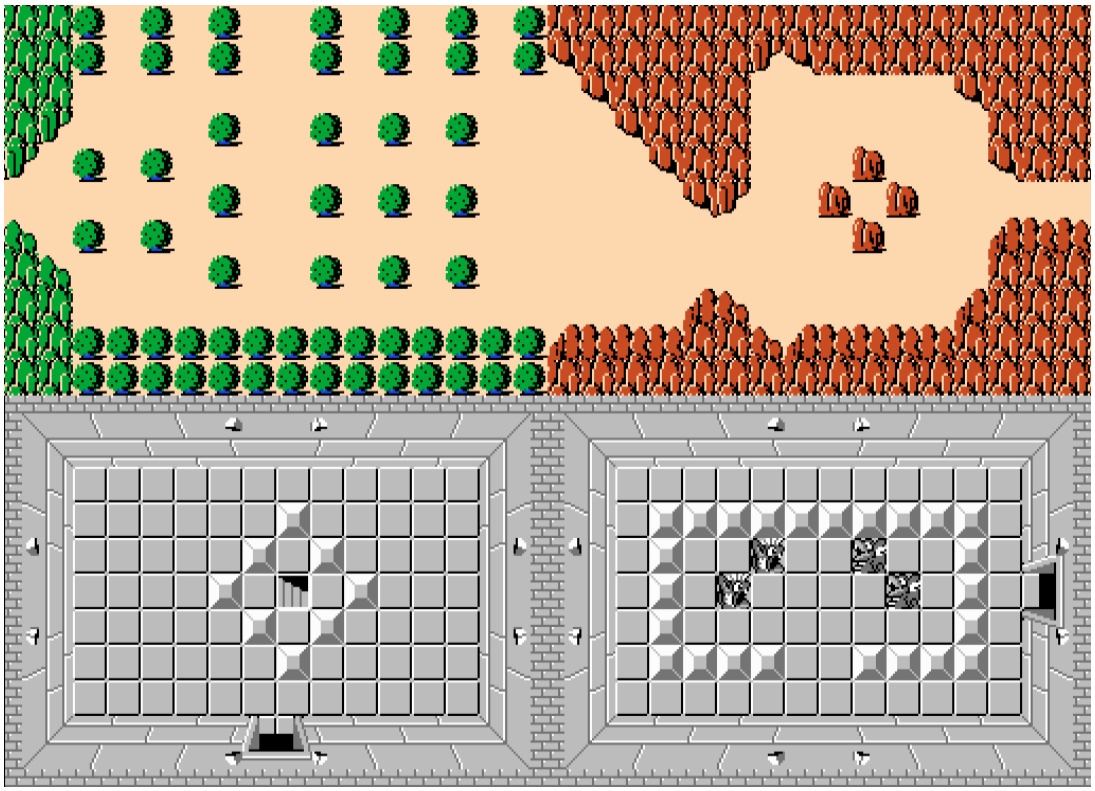
Sisällön tuottamista pystytään kuitenkin automatisoimaan ja tehostamaan eri tavoin käyttötarkoituksesta riippuen. Tietenkin myös näitä tapoja hyödyntämällä hyvin usein käytetään paljon käsin tehtyjä assetteja, kuten 3D-objekteja ja tekstuureita. Käsittelen lyhyesti kahta eri tapaa, jotka auttavat pelinkehittäjää tuottamaan peliinsä sisältöä. Nämä ovat modulaarisuus ja proseduraalisuus, joista jälkimmäiseen syvennyn enemmän työssäni.

2.1 Modulaarisuus

Modulaarisuus tarkoittaa eräänlaisten valmiiden palojen avulla rakennettua sisältöä. Nämä valmiit palat suunnitellaan niin, että ne sopivat toisiinsa sulavasti, ja niitä voi käyttää laajasti useissa paikoissa. Modulaarisessa suunnittelussa käytetään eräänlaisia rakennussarjoja, joidenka osista kasataan yhtenäinen kokonaisuus. Yksinkertaisena esimerkkinä Bethesda Game Studiosin pelissä Fallout 3 oli tehty neljästä putken palasta sarja. Nämä putket olivat erimuotoisia: t-putki, mutkaputki, suora putki sekä katkennut putki. Sarjan paloja käyttämällä pystyttiin nopeasti tekemään seinillä kulkevia putkistoja. (Burgess 2013.)

2D-pelien kenttäsuunnittelussa käytetään usein modulaarisia paloja. Pelin maailman rakentamiseen käytetään niin sanottua sprite sheet -tiedostoa (KUVA 1). Tässä yhdessä kuvatiedostossa on yhdistettynä monia pienempiä kuvia, tai kuvan osia, joita yhdistelemällä voidaan rakentaa selkeä pelialue (KUVA 2).

KUVA 1. Sprite sheet pelistä The Legend of Zelda (Jones, 2012)



KUVA 2. Sprite sheet tekniikalla toteutettuja alueita pelistä The Legend of Zelda (Jones, 2012)

Nykyisin myös pelihahmon toteuttaminen modulaarisena on yleistä. Tällöin pelihahmo koostetaan irrallisista paloista, joita voivat olla esimerkiksi pää, kädet, jalat, torso ynnä muut. Esimerkiksi kuvassa 3 näkyy pelihahmon irralliset palat, joista pelihahmo kootaan. Kuvan oikeassa reunassa on kuvakaappaus Unity-editorista, jossa pelihahmon liikkeitä voi animoida liikuttamalla esimerkiksi käsiä tai jalkoja. Modulaarisen hahmon animointi on huomattavasti sujuvampaa kuin kuva kuvalta piirretyn hahmon, sillä paloja pystyy vapaasti liikuttamaan, pyörittämään tai venyttämään. Animaatiosta tulee myös sulavaliikkeisempi. Modulaaristen hahmojen animoinnissa on myös mahdollista helposti vaihtaa osa hahmosta toiseen ilman, että animaatiota täytyy tehdä uudestaan. Hahmolta voi vaihtaa esimerkiksi hatun tai aseen animaation pysyessä samana. (Fessler 2014.)

KUVA 3. Modulaarisen hahmon sprite sheet, sekä animointi – näkymä Unity Editorissa (Kokkonen, 2014)

2.2 Proseduraalisuus

Modulaarinen sisällönluominen on siis täysin ihmisen toteuttamaa. Ensin tehdään palat, joista haluttua pelisisältöä voidaan koota, jonka jälkeen paloista luodaan pelimaailmoja, hahmoja tai mitä vaan. Vaikka modulaarisuus nopeuttaa sisällön luomista, halutaan joskus sisältöä niin paljon, että kaiken rakentaminen ei ole pelintekijöille ajan tai resurssien

puutteessa mahdollista. Toisaalta voidaan myös haluta, että peliin saadaan loputtomasti sisältöä, joka luodaan pelaajan koneella pelin aikana. Tällöin käytetään avuksi proseduraalista sisällönluomista. Proseduraalinen luominen tarkoittaa sitä, että tietokone luo peliin sisältöä erilaisien algoritmien ja menetelmien mukaisesti (Shaker, Togelius & Nelson 2015, 2).

Shakerin ym. (2015, 3) mukaan proseduraalisuuden selkein hyöty tulee siitä, että se ei vaadi ihmisen osallistumista luomisprosessiin. Ihmiset ovat hitaita ja kalliita, joten on hyödyllisempää, kun nopea tietokone tekee työn. Tietokone pystyy sopivilla algoritmeilla luomaan käytännössä loputtoman määrän erilaista sisältöä, mikä lisää pelin uudelleenpeluuarvoa huomattavasti. Tarvittaessa ihminen voi asettaa generaattoriin alkuasetuksia, mikäli halutaan juuri tietynlaista lopputulosta.

Proseduraalisessa sisällöntuottamisessa käytetään lähes aina apuna satunnaisuutta. Joskus sitä käytetään apuna pelin luomisvaiheessa, esimerkiksi, jos halutaan tuottaa puita muuten käsin tehdylle alueelle. Pelimaailma voidaan myös täysin arpoa joka käynnistyksen yhteydessä uudestaan. Mikäli halutaan, että sama arvottu sisältö voidaan saavuttaa uudelleen, käytetään proseduraalisessa generoinnissa niin sanottua siementä. Siemen on yleensä joko jokin numerosarja tai tekstinpätkä, josta luomistyö alkaa. Proseduraalisuudessa satunnaisuus ei varsinaisesti ole aina satunnaista, vaan tarvittaessa samalla siemenellä saadaan aina aikaan samanlainen lopputulos.

3 PROSEDURAALISIA KEINOJA

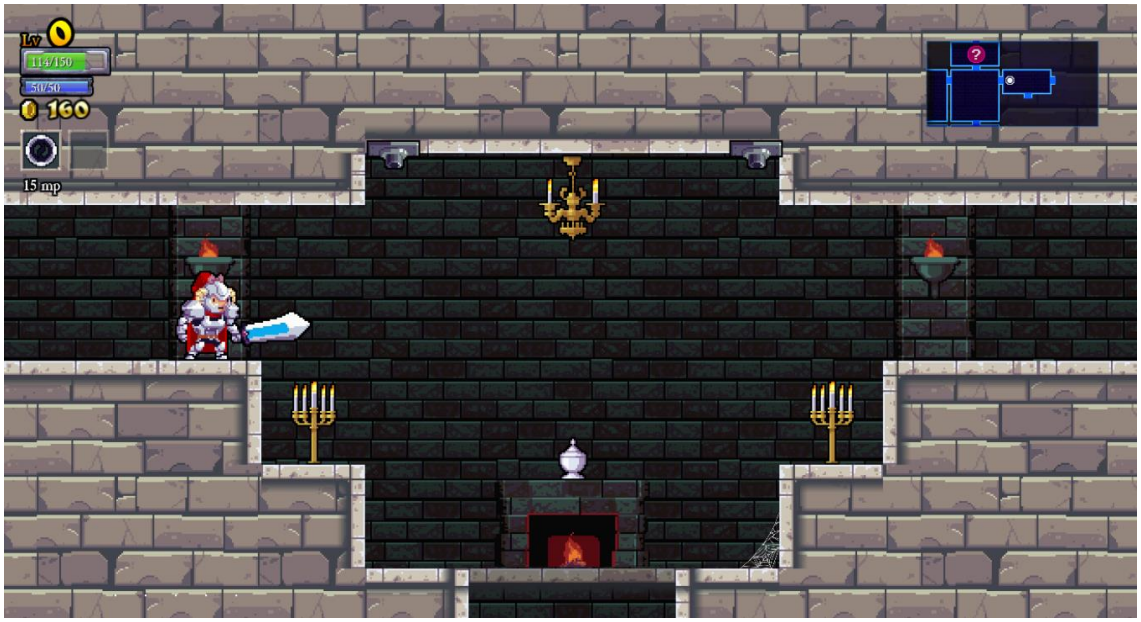
Proseduraalisen luomisen voi jakaa karkeasti neljään eri pääkategoriaan. Nämä ovat rakentavat metodit, kieliopit, rajoituksiin pohjautuvat systeemit sekä etsintäpohjaisuus (Smith & Togelius 2015). Eri tavoissa on omat vahvuutensa ja heikkoutensa, joidenka vuoksi niitä usein yhdistetään toisiinsa, jolloin voidaan paikata joidenkin metodien heikkouksia. Tässä luvussa esittelen lyhyesti tapojen eroja, ja kerron niiden eroista.

Käytän esimerkeissäni paljon tunneleita ja luolastoja, koska ne ovat helposti hahmotettava konsepti pelimaailman luomisesta. Peleissä erilaiset luolastot ovat myös tavanomaisia ympäristöjä, koska niihin on luontevaa sijoittaa esimerkiksi sokkeloita, hirviöitä tai muita haasteita pelaajalle. Myös monet ensimmäisistä tunnetuista proseduraalista generointia hyödyntäneistä peleistä, kuten Rogue (julkaistu vuonna 1980), sijoittuvat luolastoihin. Roguessa pelaaja laskeutui luolastoa alaspäin kerroksittain. Jokaisessa kerroksessa oli proseduraalisesti luotu huoneisto, josta piti etsiä tie seuraavaan kerrokseen. Pelin suosion takia syntyi uusi termi, roguelike, joka tarkoittaa peliä, jossa on proseduraalisesti luotua sisältöä (Johnson, 2015).

3.1 Rakentavat metodit

Rakentavissa metodeissa nimensä mukaisesti rakennetaan sisältö. Niissä voidaan käyttää apuna esimerkiksi huoneiden pohjapiirustuksia tai eräänlaisia sapluunoita, joita yhdistelemällä voidaan rakentaa luolasto. Esimerkiksi peli Rogue Legacy (julkaistu vuonna 2013) käyttää luolaston generoinnissa valmiita huoneita, joiden sisällä vaihtelevat erilaiset viholliset, aarteet, ansat ja esteet (KUVA 4). Pelissä voi siis olla ulkomuodoltaan samanlaisia huoneita, mutta eri pelikerroilla niihin arvotaan erilainen sisältö. Yhdellä pelikerralla huoneessa voi olla muutama esine tai aarrearkku, toisella kerralla vihollisia ja ansoja. Rakentavissa metodeissa pyritään erilaisin algoritmein ohjaamaan tietokoneen luomaa sisältöä haluttuun suuntaan, esimerkiksi huoneiden kokoa ja määrää, aarteiden sijaintia tai maaston esteitä. Koska tätä metodia käytetään usein valmiiden objektipohjien kanssa, pitää algoritmien suunnittelussa ottaa huomioon, että tuotetusta pelisisällöstä tulee tarpeeksi vaihtelevaa ja ettei se ala kyllästyttää pelaajaa. Rakentavat metodit ovat suoritukseltaan nopeita, minkä vuoksi niitä on mahdollista hyödyntää pelin aikana. Tämä tarkoittaa käytännössä sitä, että samalla kun pelaaja jatkaa pelaamistaan, luo generaattori

tämän eteen uutta sisältöä. Jotkin proseduraalisen sisällönluomisen tavat voivat olla liian hitaita tämänlaiseen työhön. (Smith & Togelius 2015.)



KUVA 4. Kuvankaappaus Rogue Legacy pelistä (Cellar Door Games, 2013)

3.2 Kohina

Kohina on hyvin monikäyttöistä proseduraalisessa sisällönluomisessa. Sillä voi tehdä esimerkiksi satunnaisia tekstuureita, maastoja tai partikkeliefektejä. Kohina on yksinkertaisimmillaan televisiostakin tuttua valkoista kohinaa, jossa jokainen pikseli on täysin satunnaisesti joko musta tai valkoinen. Tällainen kohina ei välttämättä ole kovin hyödyllistä paitsi erikoistapauksissa, koska lähes koskaan proseduraalisesti tuotettu sisältö ei ole täysin satunnaista. Erilaisia kohina-algoritmejä on kehitetty lukemattomia määriä. Tunnetuimpia proseduraalisia kohinatyyppejä ovat kehittäjiensä mukaan nimetyt Perlin kohina sekä Worleyn kohina.

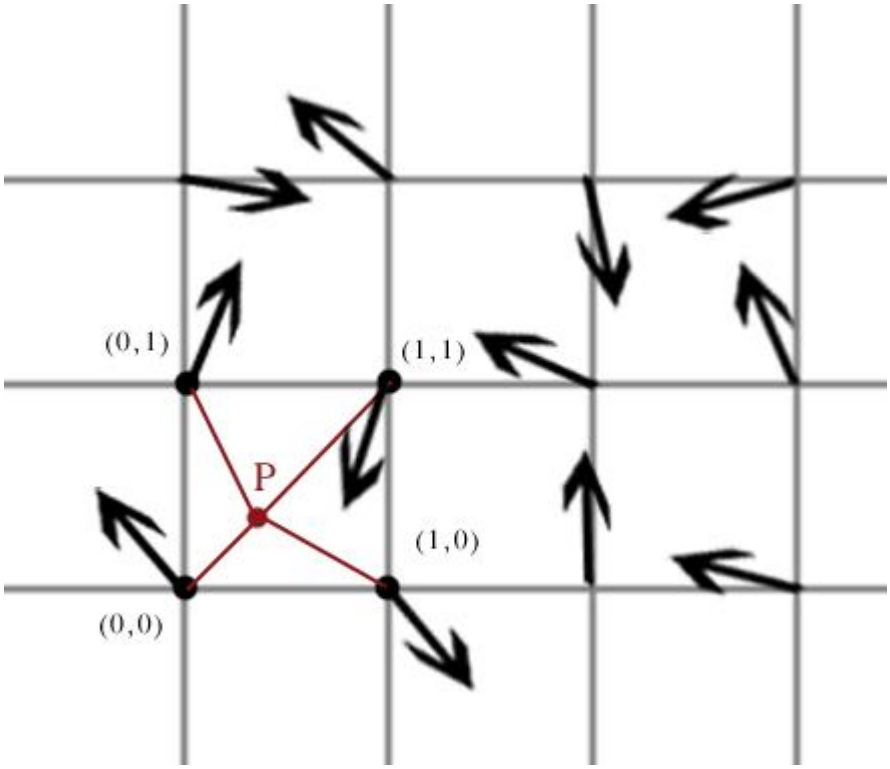
3.2.1 Perlin kohina

Tunnetuin kohinatyyppi on Ken Perlinin mukaan nimetty Perlinin kohina, jonka hän kehitti työskennellessään Tron-elokuvan parissa vuonna 1982. (Shaker ym. 2015, 61.) Tämä algoritmi tuottaa kohinaa, joka vaihtelee luonnollisesti ja toistuu ilman huomattavia terä-

viä muutoksia. Perlinin algoritmilla pystytään luomaan kohinaa niin monessa ulottuvuudessa kuin on tarvetta, mutta kohinan laskemiseen tarvittava aika kasvaa eksponentiaalisesti, minkä vuoksi yleisimmin käytetään yksi-, kaksi- tai kolmiulotteista kohinaa. Yksinkertaisimmillaan Perlinin kohinalla tuotetaan mustavalkoinen, epäsäännöllisesti vaihteleva tekstuuri. Kohinalla tuotettu kuva on yleensä helposti tunnistettavissa, mikäli siihen käytettyjä algoritmeja ei ole paljoa muokattu (KUVA 5).

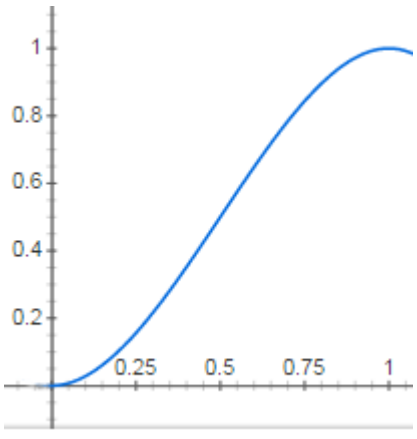
KUVA 5. Perinteisellä Perlinin kohinalla tuotettu tekstuuri

Kaksiulotteista tekstuuria Perlinin kohinalla luodessa tehdään ensin ruudukko, jonka jokaiseen ruutuun arvotaan vektorit. Jokainen pikseli tekstuurissa sijaitsee neljän vektorin muodostaman neliön sisällä (KUVA 6). Kuvassa pisteelle P saadaan arvo laskemalla yhteen jokaisesta kulmasta pikselin etäisyyden suhteen interpoloitu arvo. Interpolointi tarkoittaa sitä, että kahden tiedetyn luvun väliltä määritellään lukuarvo tietylle pisteelle. Esimerkiksi auto, jonka sijainti tiedetään kahdella eri ajanhetkellä. Interpoloimalla näiden pisteiden välillä, saadaan selville auton sijainti haluttuna aikana.



KUVA 6. Ruudukko, joka on täytetty arvotuilla vektoreilla

Arvo voidaan interpoloida lineaarisesti, eli kasvattamalla arvoa tasaisesti pisteen lähestyessä kulmaa. Käyttämällä funktiota, joka interpoloi etäisyyttä S-kirjaimen muotoisesti (KUVA 7), saadaan kuviosta pehmeämpi. Yksinkertainen funktio $y = -2x^3 + 3x^2$ on yleinen tähän tarkoitukseen, koska sillä saa arvon väliltä 0–1 kun x on välillä 0–1.

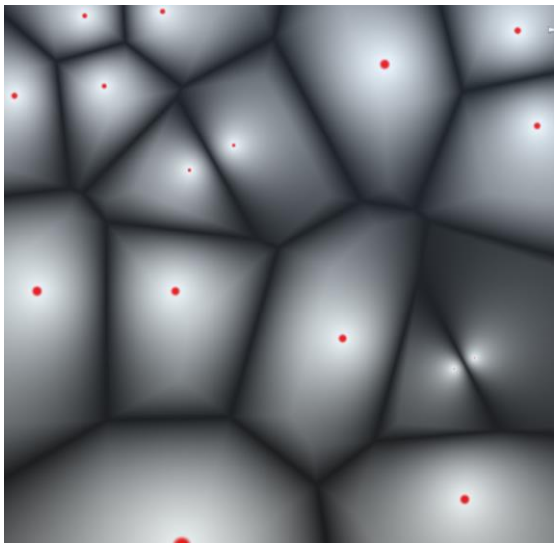


KUVA 7. S-kirjaimen muotoinen funktio yhtälöllä $f(x) = -2x^3 + 3x^2$

3.2.2 Worleyn kohina

Steven Worleyn kehittämä, myös nimellä cell noise (solukohina) tunnettu, Worleyn kohina on algoritmi, joka luo solumaisia rakenteita. Tekstuuri toteutetaan jakamalla sille ensin satunnaisesti pisteitä, jonka jälkeen jokaiselle pikselille etsitään lähin piste. Tämän jälkeen lasketaan pikselille väriarvo riippuen tämän etäisyydestä lähimpään pisteeseensä. Näin saadaan aikaiseksi solumaisia rakenteita sisältävä tekstuuri. (Rosén, 2006)

Kuvassa 8 näkyy Worleyn kohinalla toteutettu tekstuuri. Kuvaan on arvottu joukko pisteitä, jotka olen korostanut punaisella värillä. Kaikkia kuvan pikseleitä verrataan näihin pisteisiin. Pikseli saa väriarvonsa sen mukaan, millä etäisyydellä se on lähimmistä pisteistä.



KUVA 8. Worleyn kohinalla tuotettu tekstuuri

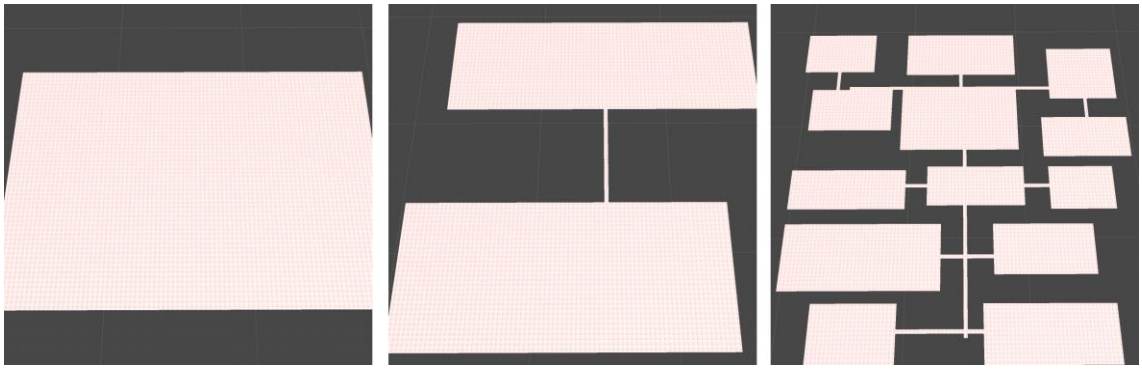
3.3 Binäärinen tilanjako

Nimensä mukaisesti binäärinen tilanjako ottaa tietyn tilan, ja jakaa sen kahteen osaan. Esimerkiksi luolasto tai kaupunki otetaan ensin kokonaisena käsittelyyn, ja jaetaan se aina kahteen pienempään, kunnes alkuperäinen tila on jaettu moneen sopivan kokoiseen alueeseen. Näitä alueita kutsutaan myös soluiksi (Shaker ym. 2015, 33). Aina tilaa jaettaessa, linkitetään uudet alueet jaettavana olevaan alueeseen. Näin saadaan puumainen rakenne, jonka avulla päästään käsiksi haluttuun soluun. Koska jokaista solua voidaan käsitellä erillisenä, on helppo määrittää kyseiseen soluun halutut ominaisuudet sen vaikuttamatta

muihin. Jos esimerkiksi käyttää binääristä tilanjakoa apuna kaupungin luomiseen, voi yhden solun määrittää keskusta-alueeksi ja toisen lähiöksi.

Usein binäärisen tilanjaon yksi selkeimmistä ominaisuuksista on se, että sillä tuotetut alueet ovat hyvin neliskulmaisia. Tämä on hyvä huomioida, kun miettii, minkälaista metodia tarvitsee omaan peliinsä. Esimerkiksi vankilaan sopii hyvin neliskulmainen pohjapiirustus, mutta se voi käyttötavasta riippuen näyttää liian kulmikkaalta luonnonmukaisissa ympäristöissä.

Yksinkertaisimmillaan binäärinen tilanjakaja aloittaa yhdellä huoneella. Tämä huone jaetaan kahteen pienempään huoneeseen, ja ne yhdistetään käytävällä. Huoneiden jakamista pienemmäksi jatketaan, kunnes on saavutettu haluttu lopputulos. Kuvassa 9 nähdään alkutilanteessa yksi iso alue, joka jaetaan ensin kahteen pienempään. Nämä kaksi aluetta on yhdistetty käytävällä. Solujen jakamista jatketaan, kunnes on saatu monen pienemmän solun kokonaisuus.



KUVA 9. Binäärinen tilanjako huoneilla ja käytävillä (Unity 5.1.1f1 2015, kuvankaappaus)

3.4 Agenttipohjainen luominen

Binäärinen tilanjako voidaan ajatella käsittelevän luotua aluetta kokonaisuudessaan. Siitä melkein päinvastaista toiminnallisuutta löytyy ohjelmistoagenteissa. Nämä robottimaiset algoritmit seuraavat niille annettuja ohjeita luodessaan pelisisältöä. Agenttia voisikin ajatella eräänlaisena robottina, joka toteuttaa tälle annettuja käskyjä. Agentti voi esimerkiksi olla luolaa kaivava robotti, joka osaa tehdä tunneleita ja huoneita. Aina huoneen jälkeen

se tekee tunnelia satunnaiseen suuntaan, jonka jälkeen se luo taas uuden huoneen. Tällaisen agentin ongelmana voikin olla, että se saattaa kaivaa huoneita päällekkäin, tai vain yhteen nurkkaan luolastoa. Binääriseen tilanjakoon verrattuna, agenttipohjaisuudella saavutetaan yleensä luonnollisempia muotoja (Shaker ym. 2015, 39).

Toisaalta agentit ovat juuri niin älykkäitä, kuinka älykkäiksi ne on ohjelmoitu. Jos agentti ohjelmoidaan välttämään päällekkäisiä tunneleita tai huoneita, ei se enää olekaan täysin sokea. Voi myös olla hyvä ajatus käyttää useita agentteja samanaikaisesti. Yksi voi toimia eräänlaisena johtajana, joka kaivaa päätunnelia. Tästä isommasta tunnelista johtaja lähettää useita pienempiä agentteja, jotka kaivavat yhden tunnelin, ja yhden huoneen. Näin saadaan rakennettua tunnelisto, jossa kaikki huoneet ovat yhteydessä toisiinsa, eikä suuria päällekkäisyyksiä pitäisi tapahtua.

3.5 Soluautomaatti

Soluautomaatti on yksinkertainen malli yksinkertaisen maailman tai objektien mallintamiseen. Yleisimmät soluautomaatit sisältävät yksi- tai kaksiulotteisen taulukon soluja, mutta se voi olla moniulotteisempikin. Soluautomaatissa on myös sääntöjä siitä, miten solut vaikuttavat toisiinsa, sekä joitain tiloja, joissa solu voi olla.

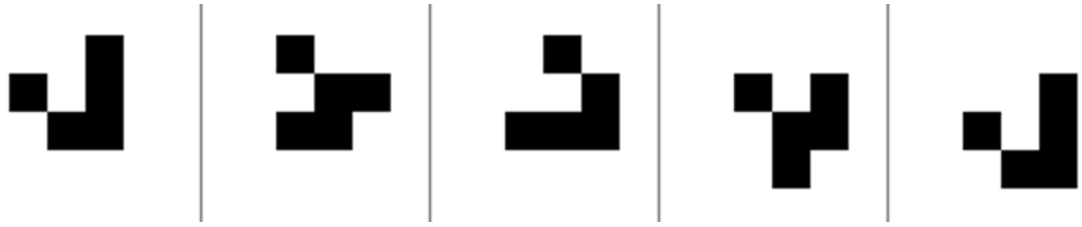
Soluautomaateilla on niin sanottuja sukupolvia. Nämä kuvaavat sen hetkistä tilaa, jossa automaatti on, ja jota käytetään seuraavan sukupolven luomiseen. Automaatti voi myös tarvittaessa käyttää useampia edellisiä sukupolvia seuraavan luomiseen.

Vaikka soluautomaatti on pohjimmiltaan melko yksinkertainen työkalu, voi sitä käyttää äärimmäisen monipuolisesti. Sen avulla on mahdollista esimerkiksi luoda proseduraalisia tekstuureita, luolastoja ja karttoja, mutta sitä pystyy hyödyntämään myös moneen muuhunkin asiaan. Sen ei välttämättä tarvitse käyttää taulukkoa, vaan se voisi esimerkiksi tutkia soluja näiden sijainnin suhteen kolmiulotteisessa avaruudessa.

3.5.1 Elämän peli

Tunnetuin soluautomaattia käyttävä sovellus on John Conwayn kehittämä, harhaanjohtavasti nimetty Game of Life (Elämän peli). Tässä epäpelissä on kaksiulotteinen taulukko, jossa solut voivat olla joko eläviä tai kuolleita. Peli aloitetaan täyttämällä halutut solut eläviksi, jonka jälkeen tarkistetaan solun naapurusto. Naapurustolla tarkoitetaan solun lähellä olevien solujen joukkoa, joka voidaan määritellä halutuin säännöin, esimerkiksi solu ja sen ympäröimät 8 solua, tai solun 4 ympäröivää solua. (Shaker ym. 2015, 42.) Seuraava sukupolvi soluja kuolee, elää tai syntyy riippuen naapuruston tilasta. Mikäli solu elää ja se koskettaa kahta tai kolmea elävää solua, se jää henkiin. Jos solu on kuollut ja se koskettaa täsmälleen kolmea elävää solu, syntyy se eloon.

Näillä säännöillä tehdään kuvioista uusia sukupolvia ja katsotaan, miten ne elävät. Peliin on kehitetty monia erilaisia yhdistelmiä ikiliikkujista eteenpäin mateleviin olioihin tai sykkiviin kuvioihin. Peliä on myös helppo tehdä monipuolisemmaksi lisäämällä siihen eri sääntöjä ja tiloja soluille, tai siitä voi tehdä moniulotteisemman. Kuvassa 10 näkyy eräänlaisen ikiliikkujan viisi sukupolvea. Eliön ensimmäinen ja viides sukupovi ovat samoja, mutta hieman eri kohdissa ruudukkoa. Tämä tarkoittaa käytännössä sitä, että eliö liikkuu eteenpäin niin kauan, kunnes törmää johonkin.



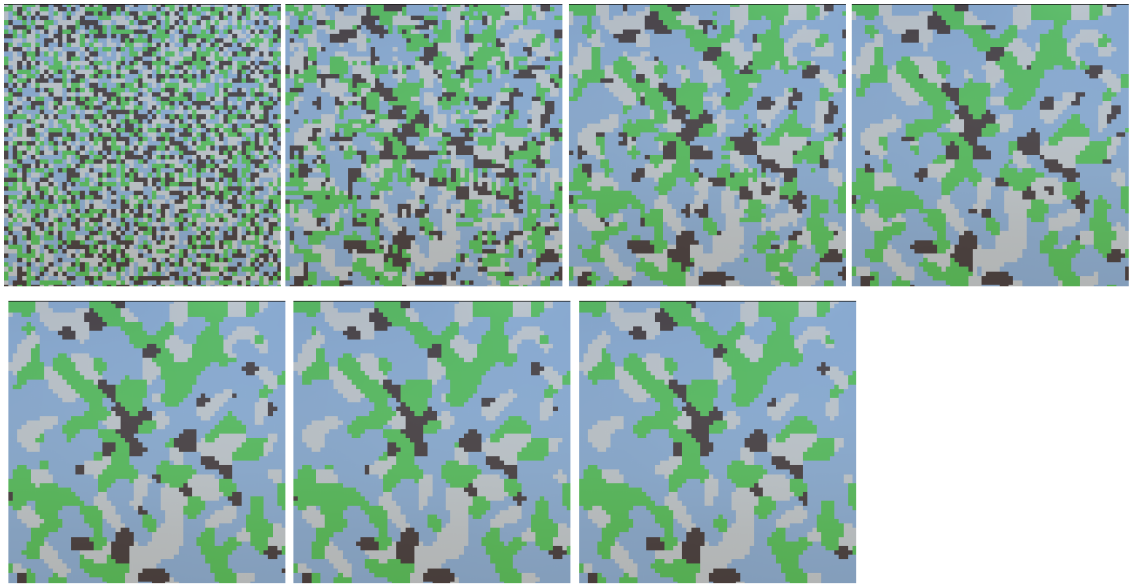
KUVA 10. Conwayn elämän pelin viisi sukupolvea

3.5.2 Luolaston tai maaston luominen soluautomaatilla

Yksinkertaisen luolaston rakentaminen soluautomaatin avulla on helppoudestaan huolimatta melko monipuolista. Solut voivat yksinkertaisimmillaan pelkkiä mustia tai valkoisia pikseleitä, mutta mikään ei estä, etteivätkö ne voisi olla muitakin objekteja. Mitä monipuolisempia solut ovat, sitä monipuolisempaa niiden käyttökin on. Tämä myös tarkoittaa sitä, että niiden käyttöön pitää miettiä tarkemmat algoritmit ja säännöt, jotta luotu

maailma on looginen kokonaisuus, jossa tiet eivät pääty järviin, eikä kaupungin keskelle ilmesty tulivuorta.

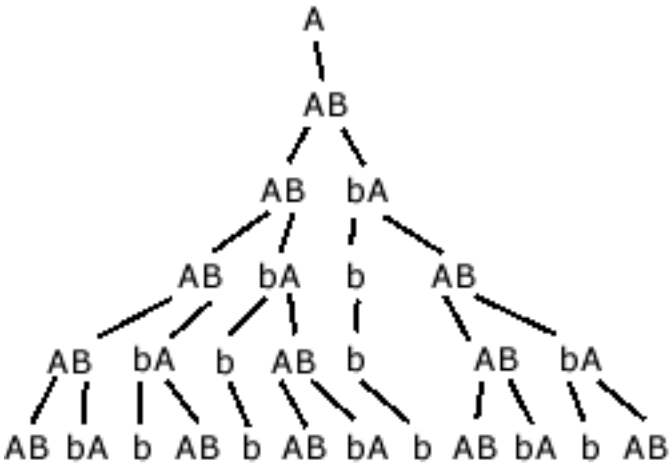
Vaihtelevan maaston luominen onnistuu esimerkiksi käyttämällä soluautomaattia, joka käyttää useampaa väriä. Esimerkiksi jo neljällä värillä saa paljon vaihtelua kuvioon (KUVA 11). Aluksi arvotaan kuvan jokainen pikseli yhdeksi neljästä väristä. Tämän jälkeen kuvaa käydään läpi haluttu määrä sukupolvia, kunnes lopputulos vastaa halutunlaista. On hyvin tavanomaista, että muutaman ensimmäisen sukupolven aikana tekstuuriin tapahtuvat muutokset ovat huomattavan suuria, kuten kuvan 11 ylärivistä voi nähdä. Soluautomaatin asetuksista riippuen, myöhempien sukupolvien muutokset voivat olla hyvin pieniä, tai niitä ei välttämättä tule enää ollenkaan. Kuvassa 11 lopullinen tekstuuri alkaa saamaan muotoaan selkeästi jo kolmannen tai neljännen sukupolven kohdalla, mutta alarivin sukupolvissa erot ovat hyvin pieniä.



KUVA 11: Soluautomaatti, joka käyttää neljää väriä

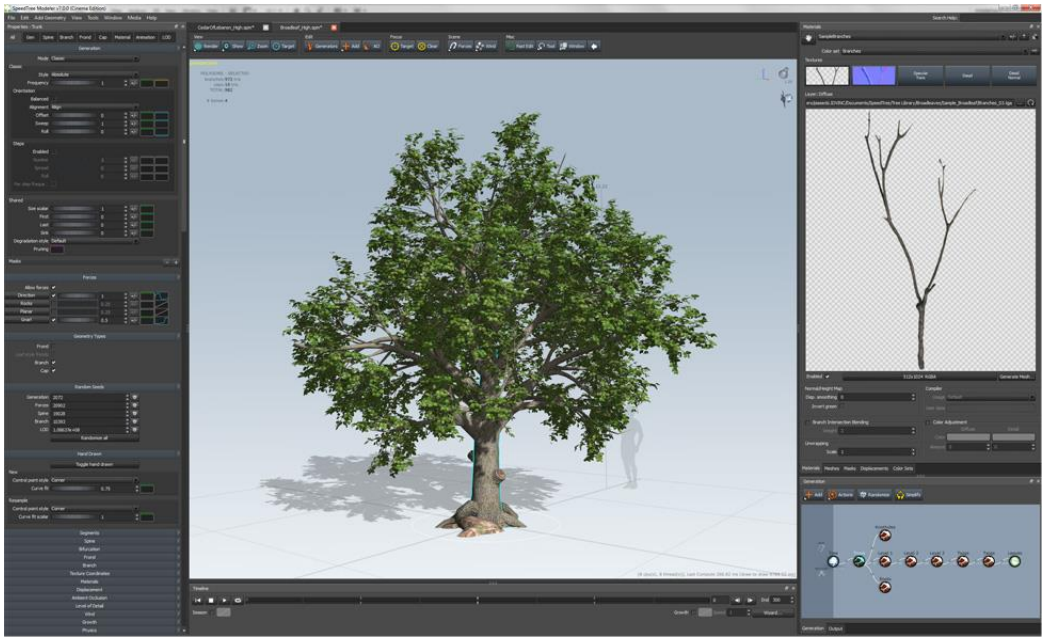
3.6 L-systeemi

Suosituimmat tavat kehittää realistisen oloista kasvustoa proseduraalisesti ovat erilaiset kieliopilliset systeemit. Yksi käytetyimmistä tavoista on L-systeemi, jolle yksinkertaisimmillaan annetaan pari sääntöä, esimerkiksi A:sta seuraa AB ja B:stä seuraa bA. Lähtöasetelma voi olla A, jolloin sarja kehittyy kuvassa 12 esiteltyjen sääntöjen mukaisesti.



KUVA 12. Esimerkki L-systeemistä

L-systeemiä käyttämällä on helppo saada aikaiseksi luontoa niin, että puut ja pensaat ovat tunnistettavissa määrin saman näköisiä, mutta eivät kuitenkaan identtisiä kopioita toisistaan. Esimerkiksi SpeedTree -ohjelma on kehitetty luonnollisen näköisten puiden ja pensaiden luomiseen (KUVA 13).



KUVA 13. SpeedTree ohjelman käyttöliittymä (SpeedTree 2015, kuvankaappaus)

3.7 Muita tapoja proseduraaliseen generointiin

Rajoituksiin perustuvat systeemit

Suhteellisen uusia tapoja, joita ei vielä ole tutkittu paljoa, ovat rajoituksiin ja sääntöihin perustuvat systeemit. Niiden idea on, että tietokoneelle annetaan korkean tason sääntöjä, joista työvälineelle kehitetyt algoritmit osaavat luoda pelialueita. Nämä säännöt voivat olla esimerkiksi: luo luolasto, jossa on kymmenen huonetta sekä aarrekammio. (Smith & Togelius 2015.)

Etsintäpohjaisuus

Toisella nimellä kutsuttu evoluutioalgoritmi eli etsintäpohjainen algoritmi on sitä, että halutusta sisällöstä tehdään useita erilaisia versioita. Näistä toisistaan hieman eroavista objekteista valitaan parhaat yksilöt tietyn pisteytyksen mukaan ja niiden ominaisuuksia yhdistellään uuteen sukupolveen. Tätä uutta sukupolvea jälleen arvioidaan ja valitaan parhaat yksilöt, joista jatketaan kehittämistä. Evoluutiokierroksia jatketaan, kunnes saavutetaan riittävän hyvä lopputulos. Etsintäpohjaisen systeemin ongelmia ovat sen hitaus ja epävarmuus: seuraava evoluution sykli ei välttämättä anna toimivaa tulosta. (Smith & Togelius 2015.)

3.8 Yhteenveto

Erilaisia tapoja ja algoritmeja proseduraaliseen sisällönluomiseen on siis paljon, enkä tässä lyhyessä pintaraapaisussa ehtinyt kuin nopeasti tutustumaan tunnetuimpiin niistä. Eri tavoilla on omat heikkoutensa ja vahvuutensa, joita yhdistelemällä voidaan toteuttaa mitä monipuolisimpia työkaluja. Saavuttaakseen parhaimman mahdollisen lopputuloksen, pitää tietää, minkälaisen lopputuloksen haluaa ja millä metodeilla se on mahdollista saavuttaa.

Esimerkiksi eri kohinametodeilla pystyy luomaan tekstuureja, joita pystytään hyödyntämään muun muassa 3D-malleissa ja maaston muodoissa. Näillä tekstuureilla on kuitenkin

vaikeaa, ellei mahdotonta, rakentaa vaikkapa taloja tai kasveja. Binäärisellä tilanjaolla on helppo määrittää alueita, esimerkiksi huoneita talossa tai kortteleita kaupungissa, ja tällä metodilla on helppo varmistaa, etteivät ne ole päällekkäin. Agenttien kanssa päällekkäisyys voi osoittautua ongelmaksi, mutta toisaalta nämä voivat olla huomattavasti monipuolisempia. Agentin voi ohjelmoida tekemään lähes mitä vain, se voi rakentaa kaupunkeja, taloja ja huoneita tai se voi tarvittaessa toimia vaikka pelihahmon tekoälynä.

Hyvän agenttipohjaisen generaattorin tekeminen vaatii kuitenkin paljon työtä, sillä sen toimivuus pitää testata tarkkaan. Sen sijaan yksinkertaisempi tapa toteuttaa sisältöä, on käyttää rakentavia metodeja, jotka voivat toimia kuin lego-palikat. Rakentavat metodit käyttävät valmiita rakennuspaloja ja yhdistelevät niitä.

4 UNITY

4.1 Unitystä lyhyesti

Unity on tietokoneohjelmisto, jota käytetään pääasiallisesti pelien tekemiseen. Kenties suurimpia valtteja Unityssä on sen helppokäyttöinen graafinen editori, sekä mahdollisuus kääntää pelit lähes mille tahansa modernille alustalle. Alun perin vuonna 2005 julkaistu Unity on viimeaikoina saanut suuren määrä huomiota ja se on nykyään yksi suosituimmista pelintekoon käytetyistä työkaluista. Unitystä on sekä ilmainen että maksullinen versio, minkä vuoksi se on kaikkien käytettävissä. Tämä on lisännyt sen suosiota huomattavasti esimerkiksi pienten pelistudioiden, pelialan opiskelijoiden sekä harrastelijoiden keskuudessa.

Unity-editorin käyttäminen on tehty hyvin helpoksi. Käyttäjä pystyy raahaamaan eri elementtejä haluamilleen paikoille, ja ainakin teoriassa niiden pitäisi toimia ilman suurempia ponnisteluja. Näin käyttäjä pystyy muun muassa tuomaan editoriin uusia assetteja, kuten ääniä, tekstuureita tai 3D-malleja. Jokainen pelissä oleva komponentti on osana GameObject-objektia. Yksinkertaisimmillaan tällainen peliobjekti sisältää vain tiedot sijainnistaan, koostaan sekä rotaatiostaan. Nämä peliobjektit ovat kaikkien pelissä olevien asioiden ydin ja niitä käytetään kaikessa. Kaikki pelissä olevat hahmot, rakennukset, valot ja äänet ovat GameObjecteja, ja yhdessä tällaisessa objektissa voi olla useita eri komponentteja. Editorissa avoimena olevaan kenttään pystyy raahaamalla lisäämään näitä peliobjekteja, objekteihin tekstuureja tai materiaaleja tai muita komponentteja, ja näihin komponentteihin niiden käyttämät assetit.

4.2 Unityn terrain

Terrainilla tarkoitetaan kolmiulotteisen maaston pinnanmuotoja, ja se voidaan esittää monella tavalla. Yleisin tapa terrainin esittämiseen on käyttää kaksiulotteista korkeuskarttaa, joka on kaksiulotteinen taulukko arvoja, jotka kuvastavat maaston korkeutta. Kuten kaikki muutkin kolmiulotteiset objektit, myös terrain koostuu vertekseistä sekä verteksien väleihin piirretyistä polygoneista. Terrainissa verteksit ovat tasaisin välimatkoin toisis-

taan muodostaen neliskulmaisen, taulukkomaisen levyn. Korkeuskarttaa käyttämällä jokaiselle verteksille katsotaan kartan taulukosta arvo, jonka perusteella määritellään, mille korkeudelle verteksi nostetaan. Tämänlaisessa korkeuskartassa huonona puolena on se, että yhdessä kohtaa karttaa voi olla vain yksi korkeusarvo, mikä tarkoittaa käytännössä sitä, että luolien tai luolamaisten muodostelmien tekeminen on mahdotonta. Helpoin tapa tallentaa korkeuskartta, on tehdä korkeusarvojen taulukosta mustavalkoinen kuva, josta jokaiselle verteksille katsotaan korkeusarvo tätä vastaavasta pikelistä.

Unityn terrain tukee myös useita yhtäaikaisia tekstuureja, minkä avulla maastosta saa vaihtelevan näköisen. Käyttämällä useampia erilaisia tekstuureja maastossa, voi se vaihdella esimerkiksi nurmikentästä hiekkaan tai asfalttiin. Maastoon voi myös luoda helposti vaihtelua käyttämällä hieman erilaisia tekstuureja päällekkäin.

5 PROSEDURAALISEN MAASTON LUOMINEN

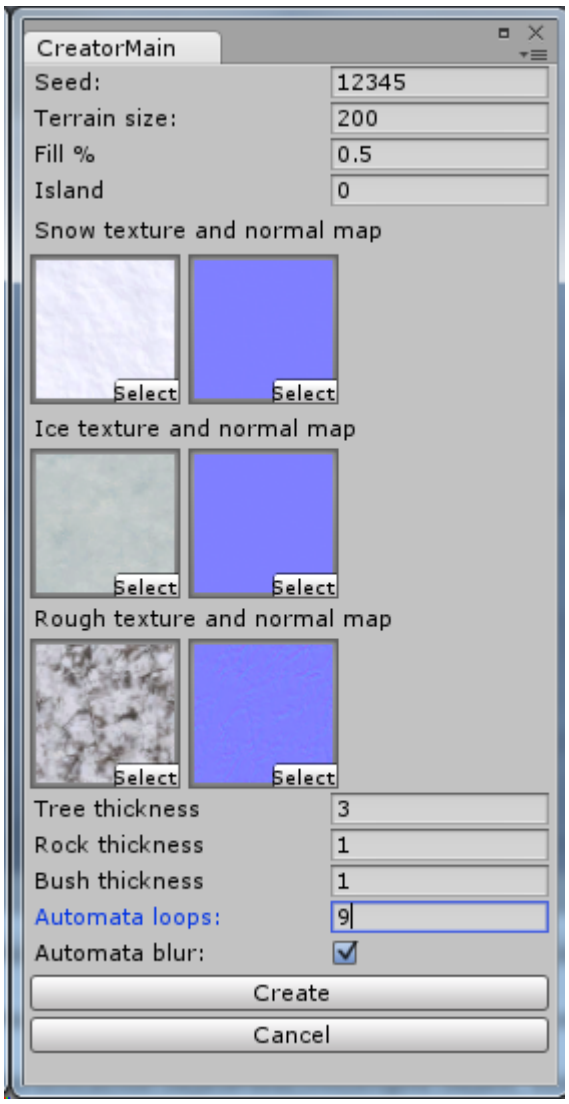
Toimeksiantajan antama tehtävä oli luoda työkalu, joka luo kolmiulotteisen pelialueen. Pelialueella pitää olla maata sekä vesialueita, ja sen pitää vaikuttaa luonnonmukaiselta. Koska pelin ajankohta on talvi, ovat vesialueet aina jään peitossa, minkä vuoksi ne ovat helppo toteuttaa. Pääasiassa maa-alue olisi metsää, joten siellä täytyisi olla puita, pensaita ja kiviä. Alueelle tarvitaan myös rakennuksia, kuten taloja tai laavuja, sekä järvien rannoille veneitä ja laitureita. Generaattorini käyttää peliin jo tehtyjä 3D-malleja sekä tekstuureja.

5.1 Työkalun asetukset

Tekemälläni työkalulla on helppo määritellä useita eri asetuksia, jotka vaikuttavat luodun maaston ulkoasuun. Kuvassa 14 näkyy eri asetukset, mitä maastogeneraattoriin voi määritellä. Työkaluun voi syöttää halutun siemenen (engl. *seed*). Samalla siemenellä saa aina saman lopputuloksen. Myös maaston koon määrittäminen metreissä onnistuu helposti, sekä sen, kuinka paljon pinta-alasta on maata ja kuinka paljon vettä. Tämä arvo on vain alkuasetus, ja se voi muuttua maaston pohjan saadessa lopullista muotoaan. Maastosta voi myös valita, haluaako siitä enemmän saaren mallisen, eli vesialueita luodaan enemmän kartan reunoilla, vai että järvi luodaan kartan keskelle.

Asetusikkunassa voi syöttää tekstuurin sekä normaalikartan[1] (engl. *normal map*) lumelle, jäälle sekä vaikeakulkuiselle maastolle. Näiden lisäksi asetuksista voi määrittää, kuinka tiheään kartalle luodaan puita, kiviä sekä pusikkoja. Viimeisinä asetuksina voi määrittää, kuinka monta sukupolvea soluautomaatti käy läpi, sekä halutaanko maastotekstuuriin sumennusta (engl. *blur*).

[1] Normaalikartta tarkoittaa tekstuuria, joka korostaa pinnan muotoja valon osuessa siihen



KUVA 14. Asetusikkuna luotavalle maastolle (Unity 5.1.1f1 2015, kuvankaappaus)

5.2 Maaston korkeudet

Pelimaaston luominen on selkeintä aloittaa maaston korkeuksista. Tässä tehtävässä käytin apuna kahta eri työkalua: soluautomaattia sekä Perlinin kohinaa. Soluautomaatilla sain hyvin luotua tekstuurin, joka sisältää suuria luonnollisia muotoja järvialueen pohjaksi. Kohinalla pystyy tuottamaan vaihtelevia maastonmuotoja ilman, että niihin tulisi liian äkkinäisiä ja satunnaisia muutoksia.

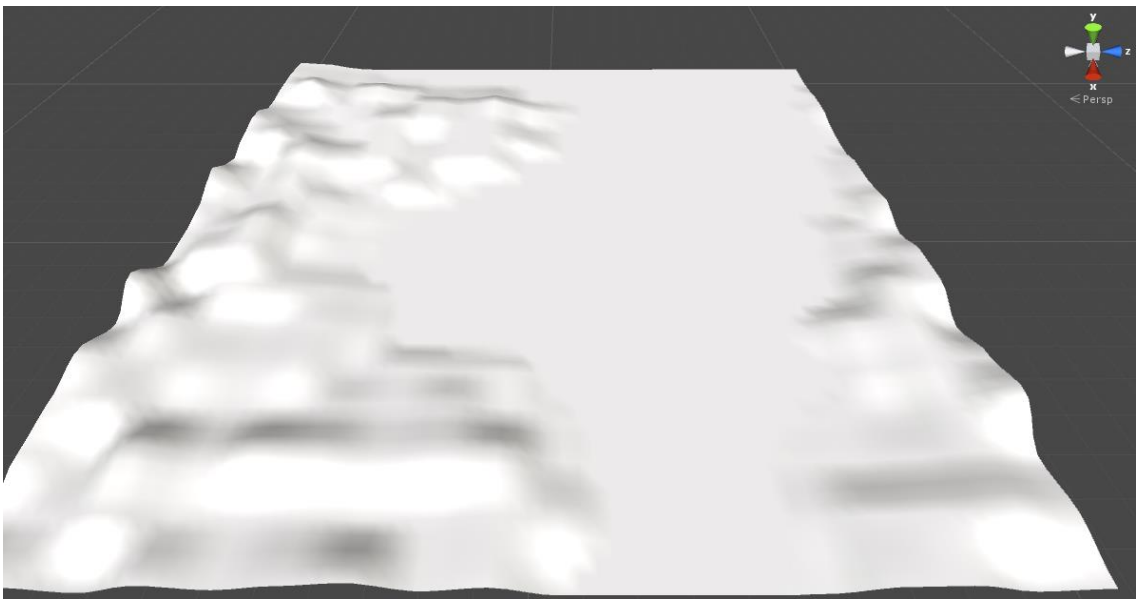
Soluautomaatilla loin karkean jaottelun maaston ja vesialueen välille luomalla mustavalkoisen tekstuurin, jossa mustat alueet olivat vettä ja valkoiset maata. Kartanluontityökalussani on myös mahdollisuus lisätä tekstuuriin sumennus, joka vähentää maaston ja ve-

sistön teräviä vaihteluja ja äkkijyrkkiä nousuja. Lisäämällä soluautomaatin luomien sukupolvien määrää, voi maastosta saada tasaisemman, mutta myös pienellä määrällä voi saada mielenkiintoisen, suota muistuttavan, lopputuloksen. Kuvassa 15 esiintyy soluautomaatin kymmenen sukupolven eroavaisuudet, sekä viimeisenä sumennettu karttapohja. Mitä pidemmälle sukupolvia jatketaan, sitä tasaisemmaksi alue muuttuu. Kartan yläosasta alaosaan jatkuva musta alue luo maastoon pelialueen halkaisevan joen.

KUVA 15. Soluautomaatilla toteutettu kartan pohja. Mustat alueet ovat vesistöä.

Seuraavaksi kuvan jokaiselle valkoiselle pikselille arvotaan korkeus. Aiemmin esitellyn Perlinin kohinan mukaisesti korkeusarvojen välillä interpoloidaan käyttämällä s-mutkan toteuttavaa funktiota f(x) = -2x^3 +3x^2 (KUVA 7), jolloin mäkien huipuille ei tule teräviä kulmia. Mikäli tekstuuria on sumennettu, saadaan maasto tasaisemmin laskeutumaan vesialueeksi. Jokaisesta tekstuurin pikselistä otetaan korkeuskertoimeksi arvo väliltä 0–1 pikselin värin mukaan. Valkoinen pikseli on arvoltaan 1, musta 0 ja harmaan sävyt jotain siltä väliltä. Maa-alueen ja veden välille sumentuu harmaa pikseli, jonka arvo voi olla esimerkiksi 0,5. Kun tällä korkeuskertoimella kerrotaan maaston korkeusarvo, saadaan selville lopullinen maaston korkeus kyseisellä kohdalla. Lopullinen korkeus on sitä lähempänä nollaa, mitä tummempi pikseli on. Vesistöalue on tasainen, sillä sen korkeus on aina 0.

Pelin kartassa pitää olla tasainen alue, josta peli alkaa. Aloitusalueen koko arvotaan haluttujen rajojen välille, se ei esimerkiksi voi ikinä olla alle kymmenen metrin levyinen, eikä yli neljääkymmentä metriä pitkä. Tämä alue on aina halutulla korkeudella ja sen koordinaatit tallennetaan muistiin myöhempää käyttöä varten, kun sinne tullaan luomaan objekteja. Näiden tietojen pohjalta voidaan asettaa terrainin korkeudet (KUVA 16).



KUVA 16. Aikaisemman tekstuurin pohjalta luotu maastopohja (Unity 5.1.1f1 2015, kuvankaappaus)

5.3 Tekstuurien lisääminen

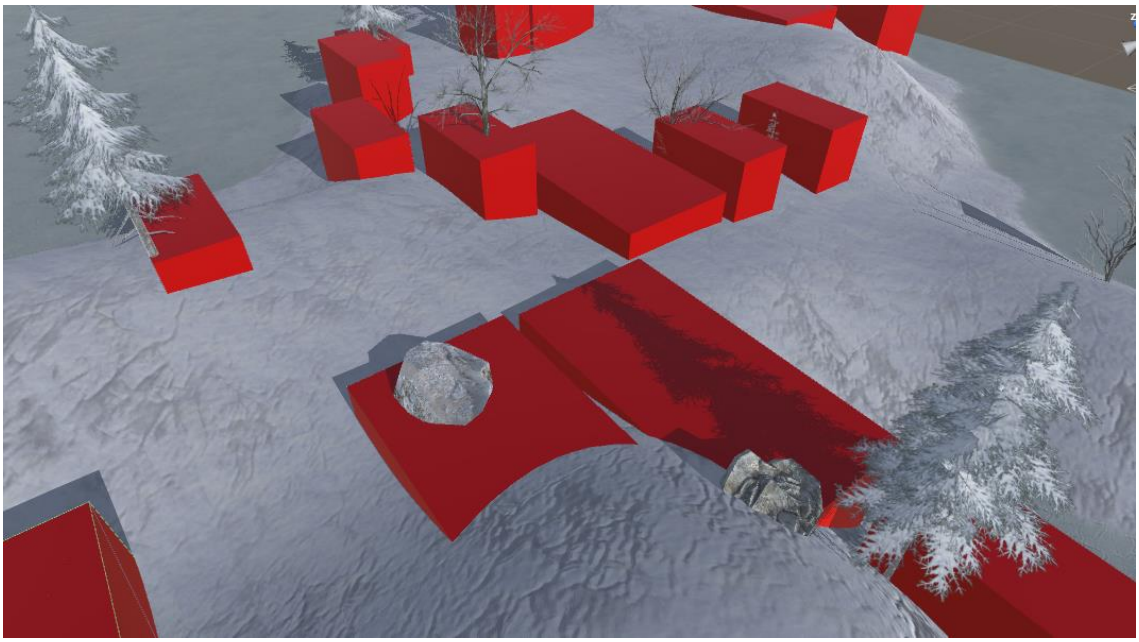
Kun terrainin korkeuskartta on saatu toteutettua, siirrytään lisäämään maastoon tekstuurit. Oletuksena maastogeneraattorini käyttää kolmea tekstuuria: lumi, jää sekä vaikeakulkuinen alue. Terrainiin on helppo tarvittaessa lisätä myös muita tekstuureita, kuten yksityiskohtia lumeen ja jäähän, sekä poluille omansa. Samassa kohtaa maastoa voi näyttää useaa tekstuuria päällekkäin esimerkiksi niin, että toinen näkyy selkeämmin ja toinen vain tuo vaihtelua ja yksityiskohtia. Myös normaalikarttojen lisääminen onnistuu helposti ja generaattori osaa lisätä ne terrainiin automaattisesti.

Helpointa on määritellä, mihin kohtiin maastoa piirretään jäätekstuuri katsomalla maaston korkeus: mikäli se on lähellä nollaa, siinä on jäätä. Lumisen maaston ja vaikeakulkuisen alueen kohdalla katsotaan maaston jyrkkyyttä, jonka perusteella arvioidaan, missä suhteessa näitä kahta tekstuuria sekoitetaan. Jyrkän maaston kohdalla painotetaan enemmän vaikeakulkuista tekstuuria, kun taas tasaisella maalla käytetään vain lumitekstuuria. Vaikeakulkuista tekstuuria voi lisätä myös muuallekin, esimerkiksi kivikkoihin.

5.4 Rakennelmat ja muut objektit

Maaston muotojen ja tekstuurien luomisen jälkeen lisätään kentälle erilaisia rakennuksia. Eri objektien päällekkäisyyksien välttäminen on tässä vaiheessa suurin ongelma. Uusia esineitä luodessa pitää tietää, missä muut esineet sijaitsevat, jotta puita ei luoda kivien tai talojen sisään. Yksi vaihtoehto on ottaa satunnaisesti piste ja tarkistaa, onko kyseisellä paikalla esteitä. Tämä kuitenkin tuo liikaa satunnaisuutta kentän luomiseen esimerkiksi, jos kaikki arvotut pisteet painottuvat kartan yhteen nurkkaan ja muu kartta jää autioksi.

Käyttämällä binääristä tilanjakajaa pystytään varmistamaan, että objekteja ei luoda päällekkäin. Objektia luodessa tilanjakajalta pyydetään halutun kokoinen vapaa alue, joka ei ole vielä varattu. Tälle alueelle luodaan objekti, jonka jälkeen se merkitään varatuksi, jottei sinne myöhemmin luoda uusia objekteja (KUVA 17). Binäärisen tilanjakajan puumaisen rakenteen vuoksi on helppo varmistaa, että objektit saadaan levittymään tasaisesti ympäri karttaa. Puurakenteessa jokainen solu haarautuu kahdeksi pienemmäksi soluksi, joten esimerkiksi ensimmäinen haara määrittää, käsitelläänkö kartan oikeaa vai vasenta puolta. Säännöllisellä polkujen vaihtelulla objektit levittyvät tasaisesti ympäri karttaa.



KUVA 17. Binäärisen tilanjakajan varatut alueet merkitään punaisilla laatikoilla (Unity 5.1.1f1 2015, kuvankaappaus)

Kenttägeneraattori aloittaa objektien lisäämisen aloitusalueelle. Sinne lisätään nuotio sekä sen ympärille muutama penkki. Mikäli alue on pieni, voi sinne asettaa esimerkiksi

talon. Suuremmalle alueelle voidaan luoda useampia taloja sekä mahdollisesti leikkimökki tai vaja. Tämän jälkeen järvien rannoille on hyvä luoda laitureita sekä soutuveneitä. Rantaviivan etsiminen onnistuu ottamalla kaksi satunnaista pistettä, toinen maalta ja toinen vedestä. Näiden pisteiden välistä haetaan kohta, jossa maaston korkeus muuttuu nollasta isommaksi. Tälle paikalle voidaan asettaa haluttu objekti, ja jälleen tilanjakajalle ilmoitetaan varatusta alueesta päällekkäisyyksien välttämiseksi. Metsäalueelle voidaan myös lisätä laavuja sekä niiden yhteyteen kuuluvia objekteja kuten nuotiopaikkoja ja halkopinoja.

5.5 Puut, kivet ja pensaat

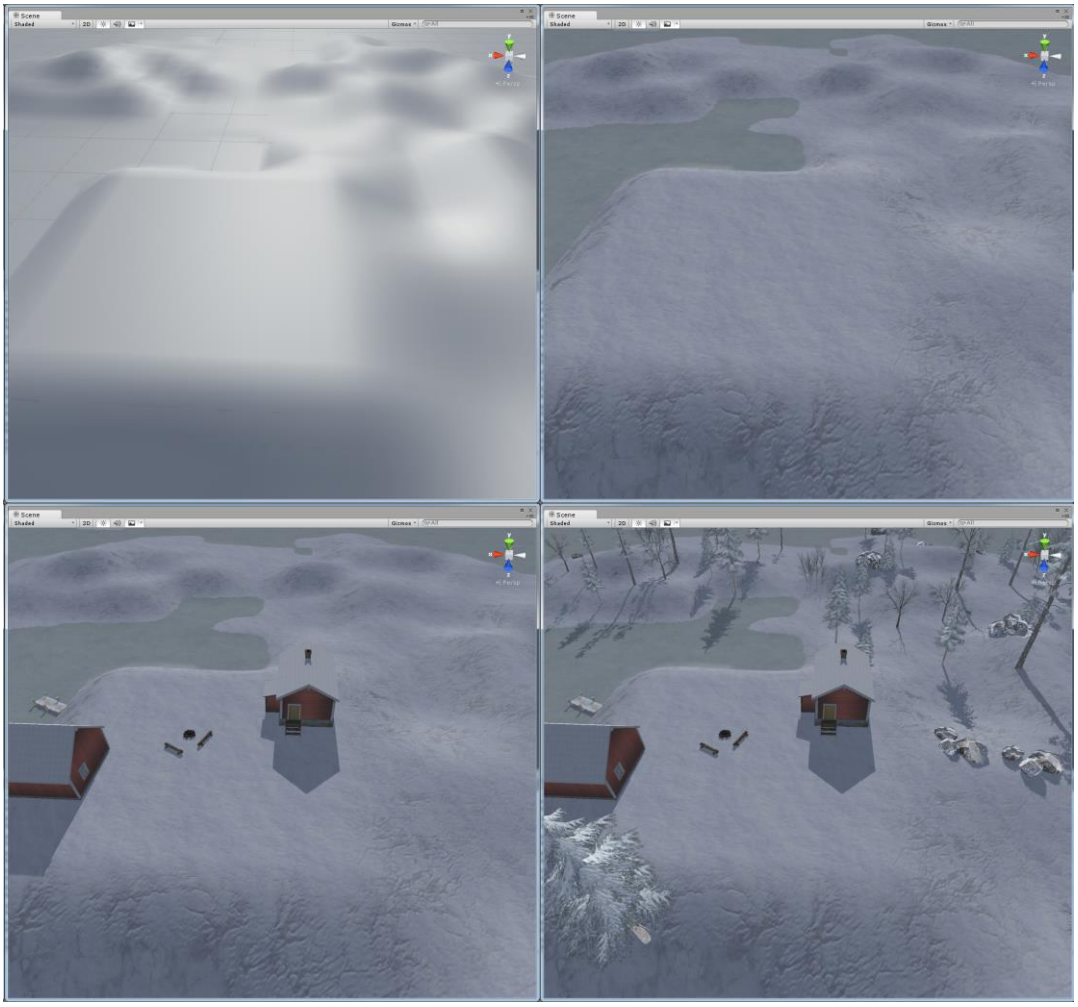
Jotta metsä näyttäisi metsältä, pitää se täyttää luonnonmukaisilla objekteilla. Työssäni minulla on käytössä toimeksiantajani 3D-malleja erilaisista puu- ja pensastyypeistä, sekä erimuotoisia ja -kokoisia kiviä. Generaattorissa ei ole muita säätömahdollisuuksia kuin säätää puiden, pensaiden ja kivien esiintymistiheyttä. Tarpeen mukaan maastosta on mahdollista tehdä esimerkiksi todella tiheä metsä täyttämällä se puilla tai avoin kivikko.

Myös metsän objektit hyödyntävät samaa binääristä tilanjakajaa etsiessään itselleen vapaata paikkaa terrainilta. Näin voidaan välttyä objektien päällekkäin luomiselta. Metsän täyttämiseen olisi mahdollista myös lisätä paljon muita asioita, mitä luonnosta voisi löytyä, kuten kukkia, kantoja tai kaatuneiden puiden runkoja, mutta työn tekemisen hetkellä niitä ei ollut käytettävissä. Generaattoriin voisi myös lisätä useita eri säätömahdollisuuksia, jotka muokkaisivat metsän ulkoasua kuten, onko se lehti- tai kuusimetsää, onko siellä tiheikköjä, mitä puulajeja metsässä esiintyy ja niin edelleen.

5.6 Lopputulos

Generaattorin eri vaiheet ovat siis maaston pohjan luominen, tekstuurien lisääminen, aloitus-alueen muodostaminen sekä viimeisenä objektien lisääminen maastoon. Kuvassa 18 näkyy selkeästi jokainen eri vaihe. Kun generaattori on tehnyt tehtävänsä, voi pelintekijä vielä tarkistaa, että kenttä on tarpeeksi mielenkiintoinen ja eheä kokonaisuus pelattavaksi. Karttaan pystyy tässä vaiheessa tekemään halutessaan muutoksia. Maastoa voi tasoittaa Unityn omilla terrain työkaluilla, ja tekstuureita voi parannella.

Myös kaikkien objektien lisääminen, siirtäminen tai poistaminen onnistuu vaivattomasti Unity-editorissa. Maastoon voi esimerkiksi suoraan assettikansiosta raahata mitä vain objekteja. Koska generaattorin ei ollut tarkoituskaan luoda lopullista karttaa, ei sen tuottaman lopputuloksen tarvitse olla täydellinen. Kartan tekijällä on aina mahdollisuus tehdä lopulliset hienosäädöt.



KUVA 18. Kartanluontityökalun eri vaiheet (Unity 5.1.1f1 2015, kuvankaappaus)

6 POHDINTA

Toimeksiantajan tehtävä osoittautui luultua vaikeammaksi, mutta siitä huolimatta, tai kenties siitä syystä, pystyin oppimaan valtavan määrän uutta proseduraalisesta sisällönkehityksestä. Lopputulos onnistui hyvin, ja toimeksiantaja sai siitä hyvän työkalun useiden karttojen luomiseen Ice Lakes -peliinsä. Pelisisällön luominen nopeutuu, kun pelisuunnittelijan ei tarvitse tehdä karttoja aivan alusta asti itse.

Luomaani generaattoria olisi vielä voinut jatkaa pidemmällekin, mahdollisesti jopa niin, että se loisi jokaiselle pelikerralle uuden pelialueen. Tämä kuitenkin olisi vaatinut valtavan määrän lisää suunnittelua sekä testaamista, jotta voitaisiin varmistaa, että kartoista tulisi hyvän näköisiä ja toimivia. Nyt kun pelisuunnittelija viimeistelee pelialueet, ne voidaan hioa juuri halutun laisiksi, sekä niihin voidaan lisätä omia, persoonallisia yksityiskohtia.

Opinnäytetyötä varten tutkittujen erilaisten proseduraalisten menetelmien monipuolisuus ja monikäyttöisyys yllätti. Ennen projektia minulla oli jonkinlainen käsitys siitä, mitä proseduraalisuus käytännössä tarkoittaa, mutta lopussa huomasin, että näitä työkaluja voi käyttää apuna lähes missä vain. Näiden työkalujen kehittäminen on kuitenkin todella työlästä ja hankalaa. Aluksi ajattelin, että tehtävä olisi paljon helpompi kuin minkälaiseksi se lopulta paljastui. Kuvittelin esimerkiksi, että maaston muodot pystyttäisiin melko vaivattomasti arpomaan sopiviksi, mutta niin tehty maasto näytti täysin epärealistiselta ja kulmikkaalta. Myös pelialueen objektien, kuten talojen ja puiden, asettelu tuotti aluksi ongelmia. Objektit ilmestyivät usein päällekkäin tai paikkoihin, joissa niiden ei kuulunut olla.

Opin, että hyvän proseduraalisen sisältögeneraattorin toteuttaminen vaatii halutun laisen lopputuloksen täydellistä tuntemista. Jotta pystyy ohjelmoimaan työkalun, joka rakentaa asioita, pitää sille osata asettaa täsmälliset säännöt, minkä perusteella objektien rakentaminen suoritetaan. Mikäli ei osaa rakentaa tarpeeksi selviä algoritmeja, ei voi olla varma lopputuloksen laadusta.

LÄHTEET

Burgess, J. 2013. Skyrim's Modular Approach to Level Design. Luettu 3.6.2015. http://www.gamasutra.com/blogs/JoelBurgess/20130501/191514/Skyrims_Modular_Approach_to_Level_Design.php

Fessler, D. 2014. Thoughts on modular animation. Luettu 3.6.2015. http://danfessler.com/blog/thoughts-on-modular-animation

Johnson, M. 2015. Before Spelunky and FTL, There was Only ASCII. Luettu 15.8.2015. http://www.pastemagazine.com/articles/2015/07/before-spelunky-and-ftl-there-was-only-ascii.html

Jones, B. 2012. More GPU Tile map demos (Zelda). Luettu 15.8.2015. http://blog.tojicode.com/2012/08/more-gpu-tile-map-demos-zelda.html

Kokkonen, V-P. 2D Best Practices In Unity. Luento. Youtube 2014. Katsottu 15.8.2015. https://www.youtube.com/watch?v=HM17mAmLd7k

Rosén, C. 2006. Cell Noise and Processing. Linköping University. Tutkielma.

Shaker, N., Togelius, J. & Nelson, M. 2015. Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer.

Smith, G. & Togelius, J. 2015 The Power and Peril of PCG. http://www.gdcvault.com/play/1022134/Making-Things-Up-The-Power

SpeedTree. Kuvakaappaus ohjelmiston käyttöliittymästä. 2015. http://www.speedtree.com/gui-interface1.html

Unity 5.1.1f1. 2015 Unity Technologies.