Aleksi Ilmonen

Kolmiulotteisen maaston generoinnin metodit peleissä

Tietotekniikan Kandidaatintutkielma

29. huhtikuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Aleksi Ilmonen

Yhteystiedot: aleksi.m.ilmonen@student.jyu.fi

Ohjaaja: Tuomo Rossi

Työn nimi: Kolmiulotteisen maaston generoinnin metodit peleissä

Title in English: Methods of Three-Dimensional Terrain Generation in Games

Työ: Kandidaatintutkielma

Sivumäärä: 25+0

Tiivistelmä: Videopeleissä on viimeisten vuosien varrella sisällön määrä kasvanut runsaasti. Sisältöä luodaan proseduraalisesti generoiden, jotta säästetään aikaa ja resursseja. Tässä tutkielmassa käsitellään erilaisia kolmiuloitteisen maaston proseduraalisen generoinnin menetelmiä, jotka ovat jaettu fraktaali-, agentti ja hakupohjaisiin menetelmiin. Lopuksi pohditaan jokaisten menetelmien vahvuuksia eri tilanteissa.

Avainsanat: Maasto, Generointi, Proseduraalinen, Korkeuskartta, Pelit

Abstract: The amount of content in video games has increased radically in the past few years. Content is being generated procedurally to save time and money. In this paper, different kinds of methods of procedural generation of three-dimensional terrain, which are categorized as fractal-, agent- and search-based methods, are discussed. Finally, the pros of each method in different situations are considered.

Keywords: Terrain, Generation, Procedural, Heightmap, Games

Kuviot

Sisällys

1 Johdanto

Videopeleissä on viimeisten vuosien varrella sisällön määrä kasvanut runsaasti. Samalla sisällöstä on tullut paljon yksityiskohtaisempaa. Tämä on aiheuttanut sen, että sisällön tuottaminen vaatii vuosi vuodelta enemmän aikaa ja resursseja. Tämän tojuamiseksi on kehitelty keinoja automatisoida luomisprosessia hyödyntämällä proseduraalisia menetelmiä.

Proseduraalista generointia hyödyntämällä voidaan luoda monenlaista sisältöä videopeleihin. Tekstuurit, kentät, mallit sekä sisällön sijainti kentässä voidaan generoida proseduraalisesti. Myös maasto voidaan generoida peleihin ja sen toteuttamiseksi onkin kehitelty monenlaisia metodeja. Tässä tutkielmassa keskitytään juuri näihin metodeihin ja algoritmeihin, joita voidaan hyödyntää kolmiulotteisen maaston generoinnissa.

Tutkielman luvussa kaksi selitetään korkeuskartan sekä kohinan käsite ja esitellään yleisimpiä tapoja tuottaa näitä. Kolmannessa luvussa käydään läpi algoritmeja, joilla saadaan luotua fraktaaleja maastoja. Neljännessä luvussa käsitellään ohjelmistoagentteja hyödyntävä metodi luoda korkeuskartta ja maasto. Luvussa viisi tutustutaan hakupohjaisiin generoinnin metodeihin. Kuudennessa luvussa pohdintaa esitettyjen metodien käyttömahdollisuuksia käytännön videopeleissä. Seitsemäs luku on tutkielman yhteenveto.

2 Korkeuskartta ja kohina

Maasto voidaan esittää kaksiulotteisena reaalilukujen ristikkona asettamalla ristikon leveys ja korkeus neliskulmaisen pinnan x- ja y-akseleille. Jos kyseisen ristikon arvot vastaavat maaston korkeutta kyseisessä pisteessä, kyseessä on korkeuskartta (engl. *heightmap*). Korkeuskartta on useimmiten käytetty kolmiulotteisen maaston kuvaamisen tapa, mutta muitakin tapoja on olemassa, kuten kolmiulotteinen vokseliruudukko (engl. *voxel grid*). Vokseliruudukolla voidaan kuvata luolastoja ja ulkonemia, joita ei ole mahdollista esittää korkeuskartalla. Tämä vaatii kuitenkin paljon enemmän muistia (Shaker, Togelius ja Nelson 2016).

Korkeuskarttojen luonnissa hyödynnetään usein kohinan generoinnin metodeja (engl. *noise*) (Smelik ym. 2009). Kohina myös esitetään kaksiulotteisena reaalilukujen ristikkona, joten tämä mahdollistaa kohinan hyödyntämisen korkeuskarttojen luonnissa (Shaker, Togelius ja Nelson 2016). Seuraavaksi käsitellään yleisimpiä kohinoita sekä niiden generointiin hyödynnettäviä algoritmeja.

2.1 Arvokohina

Arvokohina (engl. *value noise*) on itsessään yksinkertainen ymmärtää ja implementoida. Asettamalla ristikkoon satunnaisia arvoja saadaan luotua yksinkertainen arvokohina (Travis 2011). Tämä on primitiivisin arvokohinan muoto, joka tunnetaan myös nimellä valkoinen kohina (engl. *white noise*) (Ebert ym. 2002). Maaston generoinnin kannalta täysin satunnainen kohina ei ole toimiva ratkaisu, koska se luo piikikkäitä maastoja. Tämä johtuu siitä, että vierekkäisten pisteiden arvot eivät ole millään tavalla yhteydessä toisiinsa. Tämä voidaan estää interpoloimalla ristikon arvot.

Interpolointia varten löytyy useita metodeja, mutta yleisimmät ovat bilineaarinen (engl. *bilinear interpolation*) ja bikuutiollinen interpolaatio (engl. *bicubic interpolation*). Interpoloidessa ristikon jokaiseen kohtaan ei aseteta satunnaista arvoa, vaan ainoastaam tiettyihin kohtiin määrätyn välin mukaisesti. Nämä kohdat kuvaavat vuorien huippuja sekä laaksoja maastossa. Tämän jälkeen välissä olevat pisteet interpoloidaan halutulla menetelmällä, jolloin saadaan maastoista yhteneviä (Shaker, Togelius ja Nelson 2016).

2.1.1 Bilineaarinen interpolointi

Bilineaarinen interpolointi hyödyntää painotettua keskiarvoa etäisyyden suhteen laskiessaan puuttuvien pisteiden arvoja. Tämä tarkoittaa sitä, että jos laskemme pisteen [0, 1] korkeutta väliltä [[0, 0], [0, 10]], niin huomaamme pisteen olevan 10% matkasta kyseiseltä väliltä. Tällöin painotettu keskiarvo lasketaan: $korkeus[0,1] = 0.9 \cdot korkeus[0,0] + 0.1 \cdot korkeus[0,10]$. Kun tämä on suoritettu x-suuntaan, niin seuraavaksi suoritetaan sama operaatio y-suuntaan. Bilineaarinen interpolointi on tehokas tapa tuottaa interpoloitu korkeuskartta (Shaker, Togelius ja Nelson 2016). Bilineaarinen interpolointi tuottaa kuitenkin jokaiseen pisteeseen epäjatkuvuuksia (Bourke 1999), mikä näkyy generoidussa maastossa kulmikkaina vuoren huippuina ja laakson pohjina. Myöskin rinteistä muodostuu suoria, mikä ei vastaa oikean maailman maastoja.

2.1.2 Bikuutiollinen interpolointi

Epäluonnollisen kulmikkas ja suora maasto voidaan välttää hyödyntämällä bilineaarisen sijasta bikuutiollista interpolointialgoritmia. Bikuutiollisen interpoloinnin tarkoituksena on luoda pisteiden välille S-käyrää muistuttava käyrä suoran sijasta. Tätä varten käytetään kaltevuusfunktiota $s(x)$, mikä määrää korkeuden rinteessä, kun on kuljettu x osan matkasta. Bilineaarisen interpoloinnin tapauksessa kaltevuusfunktio olisi $s(x) = x$, mutta nyt tarvitsemme funktion, jonka käyrä muistuttaa S-käyrää. Usein tietokonegrafiikassa käytetty funktio on $s(x) = -2x^3 + 3x^2$, sen yksinkertaisuuden ja nopeuden takia (Shaker, Togelius ja Nelson 2016).

2.2 Gradienttikohina

Toinen lähestymistapa korkeuskartan luomiseen on gradienttikohinan (engl. *gradient noise*) hyödyntäminen. Arvokohinassa luotiin korkeudet ja niistä interpoloitiin kaltevuus, mutta gradienttikohina generoi kaltevuudet suoraan ja päättelee niiden perusteella korkeudet. Tällaisellä lähestymistavalla on hyvänä puolena se, että interpoloimalla kaltevuuksia, eli muutoksen nopeuksia, generoidusta maastossa tulee sileämpiä. Myöskin koska huiput ja laaksot eivät synny suoraan ristikon määrittämiin pisteisiin, niiden määrittyessä kaltevuuksien poh-

jalta, järjestäytyy maasto orgaanisemman näköiseksi (Shaker, Togelius ja Nelson 2016).

2.2.1 Perlin-kohina

Perlin-kohina on yleisesti teollisuudessa käytetty kohina, koska se on nopea, tarvitsee vähän muistia sekä on laadultaan hyvä.Sitä voidaan soveltaa myös useampiin ulottuvuuksiin, mutta maaston generoinnin kannalta olemme kiinnostuneita kaksiulotteisesta tapauksesta. Perlin kohinan toteutus vaatii kaksi taulukkoa: kaltevuus- ja permutaatiotaulukon. Kaltevuustaulukko on usein pieni, korkeintaan kahdeksanalkioinen 2D-kohinassa. Jokainen kohta on vektori, joka osoittaa epämääräiseen suuntaan (Travis 2011). Vektoreiden kannattaa olla samanmittaisia ja 2D-kohinalle hyvä tapa valita vektorit, on valita ne yksikköympyrän eri suunnista (Gustavson 2005). Kaltevuustaulukosta valitaan vektori sattumanvaraisesti jokaiseen ruudukon pisteeseen. Permutaatiotaulukko on taas paljon suurempi, usein 256 alkiota 2D-kohinalla. Tätä käytetään satunnaisnumerogeneraattorin (engl. *random number generator*) sijasta.Jotta kyseessä olisi permutaatio, jokainen luku esiintyy taulukossa täsmälleen yhden kerran. Tämän takia n kokoinen taulukko täytetään numeroista 0,...,n - 1, jonka jälkeen taulukko sekoitetaan (Travis 2011).

Kohinaa luodessa ensin etsitään tutkittavan pikselin lähimmät ruudukon pisteet ja niiden kaltevuusvektorit. Kaltevuusvektorit saadaan käyttämällä pisteen koordinaatteja löytääksemme satunnaisen numeron permutaatiotaulukosta ja käyttämällä funktiota: *kaltevuusvektori* = *kaltevuus*[*permutaatio*[x + *permutaatio*[y]]].Tämän jälkeen lasketaan jokaisen kaltevuusvektorin ja ruudukon pisteestä pikseliin menevän vektorin välinen pistetulo. Viimeiseksi interpoloidaan pistetulot (Travis 2011). Alun perin interpolointiin käytettiin bikuutiollisessa interpoloinnissa esiteltyä funktiota: $f(t) = -2^3 + 3t^2$, mutta myöhemmin se vaihdettiin funktioon: $f(t) = 6t^5 - 15t^4 + 10t^3$. Syynä tähän oli se että, kohina funktiolla olisi hyvä olla jatkuva toinen derivaatta (Gustavson 2005).

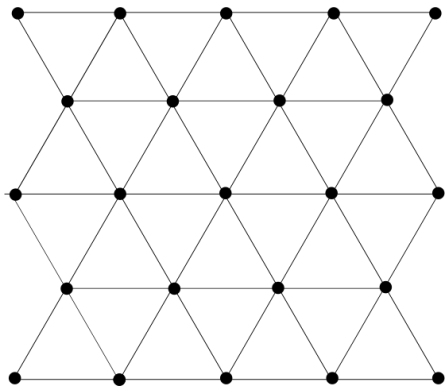
2.2.2 Simplex-kohina

Vuonna 2001 Perlin kohinan kehittäjä Ken Perlin esitteli Simplex kohinan korvaamaan perinteisen Perlin kohinan. Perlin kohinalla oli tiettyjä rajoituksia, joten Simplex kohina ke-

hitettiin voittamaan nämä rajoitukset. Simplex kohinan suurimpia etuja ovatkin: (Gustavson 2005)

- Simplex kohina vaatii vähemmän kertolaskuja.
- Simplex kohina skaalautuu paremmin korkeammilla ulottuvuuksilla ja on suoritusajaltaan O(N^2) N:ssä ulottuvuudessa, kun Perlin kohina on O(2^N).
- Simplex kohinalla ei ole huomattavia suuntaavia artefakteja (engl *directional artifact*).

Simplex kohinan toiminta perustuu simplexien käyttöön hyperkuution sijasta. Simplex on muoto, jolla on vähiten kulmia, kyseisessä ulottuvuudessa. 2D:ssä se olisi kolmio ja 3D:ssä tetraedri. Simplexien hyöty hyperkuutioihin nähden on se, että niissä on kulmia n:ssä ulottuvuudessa vain $n+1$, kun taas hyperkuutioissa niitä on 2^n-1 (Travis 2011).



Kuvio 1. Esimerkki Simplexruudukosta

Toinen Simplex kohinan parannus on se että, siinä ei tarvita interpolointia. Sen sijasta käytetään radiaalista vaimennusfunktiota (engl. *radial attenuation function*) määrättyjen kaltevuuksien painoarvoina. Perlin kohina tarvitsi 2^n-1 interpolointia, kun Simplex kohina tarvitsee $n+1$ radiaalista vaimennusta (Travis 2011). Radiaalinen vaimennusfunktio valitaan tarkkaan siten, että jokaisen kulman vaikutus putoaa nollaan ennen kuin ne ylittävät rajan seuraavan simplexin puolelle. Tämä tarkoittaa sitä, että pisteeseen simplexin sisällä vaikuttaa ainoastaan kyseisen simplexin kulmat (Gustavson 2005).

Simplexien käytössä on myös haittapuolensa. Simplexin, jonka sisällä piste on, määrittäminen ei ole yhtä yksinkertaista kuin hyperkuutioissa. Ensin koordinaatisto täytyy vääristää siten että se vastaa suorakulmaisista kolmioista muodostunutta verkkoa. Tällöin voidaan

päätellä minkä neliön sisällä ollaan, aivan kuten muidenkin kohinoiden kohdalla. Sitten vertaillaan x ja y koordinaattien arvoja, joiden avulla voidaan päätellä kumman neliön sisällä olevien simplexien sisällä piste on. Jos y > x, niin piste on ylemmän simplexin sisällä, ja jos x > y, piste on alemman simplexin sisällä. Kun simplexin kulmapisteet ovat tiedossa, täytyy koordinaatisto vääristää takaisin alkuperäiseen muotoon (Gustavson 2005).

3 Fraktaalipohjainen generointi

Aikaisemmin esitellyt kohinat tuottavat jo varsin orgaanisia tuloksia, mutta niitä katsellessa maastona, niissä on edelleen yksi varsin epäluonnollinen ominaisuus. Tällainen maasto kumpuilee tasaisin väliajoin. Oikeassa maailmassa maastot ovat luonteeltaan fraktaaleja. Tämä tarkoittaa sitä, että samat maaston piirteet ovat havaittavissa eri mittakaavoissa (Shaker, Togelius ja Nelson 2016).

Edellä mainituista kohinoista on mahdollista muodostaa fraktaaleja. Tämän saavuttamiseksi sovelletaan fraktionaalista Brownin liikettä (engl. *fractional Brownian motion*). Fraktionaalisen Brownin liikkeen perimmäinen idea on summata kohinaan useita tasoja eri taajuuksilta. Taajuuksia suurennettaessa pienennetään kohinan vaikutusta. Seuraavaksi käsitellään erilaisia menetelmiä tuottaa tällainen fraktaali maasto. Tällein uudelleen toistettaessa saadaan kohinaan yksityiskohtia eri mittakaavoille, mikä näkyy maastoissa fraktaaleina ominaisuuksina (Travis 2011).

On olemassa myös algoritmeja, jotka luovat suoraan fraktaaleja kohinoita. Seuraavaksi käsittelemme yleisimmät fraktaalin kohinan generointialgoritmit.

3.1 Keskipisteen siirto

Keskipisteen siirto -algoritmi (engl. *midpoint displacement algorithm*) on yksinkertainen menetelmä tuottaa kohina. Janaan sovellettaessa algoritmi aloitetaan asettamalla janan päätepisteiden arvot joko satunnaisluvuilla tai itse määritellyillä arvoilla. Tämän jälkeen jaetaan jana keskipisteen kohdalta kahdeksi janaksi. Tämän pisteen arvo lasketaan päätepisteiden keskiarvona, johon lisätään satunnainen siirto, jonka suuruus määräytyy janan pituuden perusteella. Tämä on kokonaisuudessaan algoritmin ensimmäinen iteraatio. Seuraavissa iteraatioissa toistetaan sama muodostuneille janoille. Janojen lyhentyessä, kun ne jaetaan keskipisteen kohdalta kahdeksi uudeksi janaksi, satunnaisen siirron vaikutus pienenee. Tällä vältetään keskipisteen siirtyminen liian radikaalisti pienemmissä mittakaavoissa (Travis 2011).

Algoritmi skaalautuu myös hyvin korkeammille ulottuvuuksille. Toisessa ulottuvuudessa

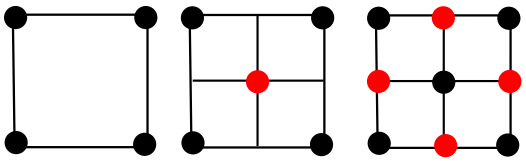
alue jaetaan neliöihin ja neliön jokaisen reunalla olevan janan keskipiste lasketaan ylempänä esitellyllä tavalla. Tässä tilanteessa täytyy laskea myös koko neliön keskipiste, joka saadaan neliön kulmien keskiarvolla. Tähän lisätään myös satunnainen siirto. Tämän jälkeen iteraatiota toistetaan, kunnes haluttu tarkkuus on saavutettu. Tässäkin on muistettava siirron suuruuden määrittäminen neliön sivun pituuden mukaan (Travis 2011).

Keskipisteen siirto -algoritmi on todella nopea, mutta vaatii varsin paljon muistia. Tuloksen laadussa isona ongelmana on säännölliset artefaktit. Tämä tarkoittaa sitä, että vuorten huiput ja laaksot muodostuvat ruudukkomaisesti maastoon (Travis 2011).

3.2 Timantti-neliö

Timantti-neliö-algoritmi (engl. *diamond-square algorithm*) on parannus Keskipisteen siirto -algoritmista. Keskipisteen siirto -algoritmin säännölliset artefaktit johtuvat siitä, että osa pisteistä lasketaan kahden pisteen keskiarvona ja toinen osa pisteistä lasketaan neljän pisteen keskiarvona. Timantti-neliö ratkaisee tämän käyttämällä neljä pistettä jokaisen pisteen laskemiseen. Neliöiden keskipiste lasketaan samalla tavalla kuin keskipisteen siirot -algoritmissä. Tätä kutsutaan algoritmin neliö-vaiheeksi. Neliön sivujen laskemiseksi hyödynnetään ympäröivien neliöiden keskipisteitä. Tällöin saadaan neljä pistettä jokaisen yksittäisen pisteen laskemiseksi. Tämä vaihe on algoritmin timantti-vaihe (Travis 2011).

Timantti-neliö-algoritmissa selkeänä ongelmana on alueen reunapikseleiden laskeminen. Niissä kohdissa timatti-vaiheessa on vain kolme pistettä olemassa. Tämän voi ratkaista muutamalla eri tavalla. Ensimmäinen on jättää neljäs piste huomioimatta ja laskea vain kolmen olemassa olevan avulla. Tätä usein käytetään ja se antaa hyvän tuloksen. Toinen tapa on käyttää neljännen pisteen kohdalla vakioarvoa pisteen laskemiseksi. Tämä tapa aiheuttaa kuitenkin artefakteja reunoille, jotka näyttävät rypyiltä maaston reunoilla (Travis 2011).



Kuvio 2. Timantti-neliö algoritmin ensimmäinen iteraatio

Eräs mielenkiintoinen vaihtoehto on määritellä neljäs piste sen olevan kuvan toisessa laidassa olevan neliön keskipiste. Tämä lisää maaston satunnaisuutta, mutta samalla tekee maastosta saumattoman, jos niitä lisättäisiin peräkkäin useita. Tämä saattaa olla joissain tilanteissa haluttu ominaisuus (Travis 2011).

4 Agentti-pohjainen generointi

Aikaisemmin esitellyllä fraktaalipohjaisilla algoritmeilla saadaan varsin luonnollisen näköisiä maastoja, mutta niitä käytettäessä käyttäjällä ei ole paljoa valtaa määrittää, millaista maastoa ollaan luomassa. Vuoret sekä laaksot muodostuvat satunnanvaraisesti eikä niiden määrään voi vaikuttaa paljoa. Tämän ongelman pohjalta Doran ja Parberry (2010) esitelivät ohjelmistoagentteja (engl. *software agent*) hyödyntävän korkeuskartan generoimismenetelmän.

Agenteille ei ole sovittua universaalia määritelmää (Rudowsky 2004), mutta Doran ja Parberry (2010) käyttävät Russellin ja Norvigin (1995) määritelmää, jonka mukaan agentit voivat olla mitä tahansa, jotka havainnoivat ympäristöä sensoreilla (engl. *sensor*) ja toimivat effektorien (engl. *effector*) kautta. Agentit ovat myös autonimisia, pois lukien muutamat rajoitukset, joihin agentin toiminta perustuu.

Doranin ja Parberryn (2010) agentit toimivat kolmessa vaiheessa, jotka voidaan yleistää rantaviiva-, maasto- ja eroosiovaiheiksi. Ensimmäisessä eli rantaviivavaiheessa monta agenttia toimii luodakseen maamassan ulkolinjan, joka on mahdollisesti veden ympäröimä. Toisessa eli maastovaiheessa suurempi lajitelma toimii luodakseen maastonmuodot, kuten vuoret, laaksot ja rannat. Viimeisessä eli eroosiovaiheessa agentit luovat joet kuluttamalla jo syntynyttä maastoa.

Kun agentit luodaan, niille annetaan tietty määrä poletteja (engl. *token*), joita ne kuluttavat aina kun erillinen toiminta on suoritettu. Näillä rajoitetaan agentin elinikää ja on yksi tapa voida kontrolloida maaston generointia. Maaston makrotason muotoihin voidaan vaikuttaa agenttien ja niiden saamien polettien määrällä (Doran ja Parberry 2010).

Doran ja Parberry (2010) käyttävät esimerkissään viittä erityyppistä agenttia, vaikka on mahdollista luoda useampia agentteja tekemään eri asioita. Nämä viisi agenttityyppiä ovat:

- Rantaviiva-agentti, joka luo maamassan.
- Tasoitusagentti, joka liikkuu satunnaisesti ja tasoittaa korkeuseroja läheisten pisteiden välillä.

- Ranta-agentti, joka luo tasaisen alueen rantaviivan viereen.
- Vuoriagentti, joka nostaa vuoristoja.
- Jokiagentti, joka luo joet, jotka juoksevat vuorilta mereen.

Seuraavaksi tutkitaan tarkemmin näitä agentteja .

4.1 Rantaviiva-agentti

Rantaviiva-agentit luovat maamassan ulkolinjan ennen kuin korkeuskartan dataa lasketaan. Rantaviivan generointi alkaa yhdellä agentilla, joka on vastuussa koko maamassasta. Tämä yksittäinen agnetti jakaa tehtävänsä luomalla useita lapsiagentteja ja antamalla niille oman alueen työskennellä. Nämä lapsiagentit luovat puolestaan omia lapsiagentteja ja prosessi jatkuu kunnes jokaisella agentilla on tarpeeksi pieni alue työskennellä. Jokainen agentti työskentelee autonomisesti riippumatta toisistaan, mutta jokaisella on yhteys luotavaan maastoon riippumatta sen luomisen tilasta (Doran ja Parberry 2010).

Alussa jokaiselle agentille annetaan yksittäinen piste maamassan reunalta, toivottu suunta sekä pisteiden määrä, minkä agentin oletetaan generoivan. Jos annettu piste on maamassan ympäröimä siinä vaiheessa kun agentti alkaa toimimaan, agentti etsii sen toivotusta suunnasta maamassan reunaa. Kun agentti asettuu maamassan reunalle, luo se kaksi satunnaista pistettä. Ensimmäinen on attraktori (engl. *attractor*) ja toinen on repulsori (engl. *repulsor*). Näiden täytyy olla erisuunnissa (Doran ja Parberry 2010).

Agentit laskevat maamassan reunalta viereisten pisteiden arvot, joista se valitsee korkeimman arvon saaneen nostamalla sen merenpinnan yläpuolelle ja liittää sen maamassaan. Pisteiden arvo lasketaan kaavalla: $d_r(p) - d_a(p) + 3d_e(p)$, jossa $d_r(p)$ on etäisyyden neliö repulsoriin, $d_a(p)$ on etäisyyden neliö attraktoriin sekä $d_e(p)$ on etäisyyden neliö lähinpään kartan reunaan. Pisteet, jotka ovat lähempänä attraktoria saavat suuremman arvon kuin pisteet lähempänä repulsoria (Doran ja Parberry 2010).

4.2 Tasoitusagentti

Tasoitusagentit tekevät satunnaisia matkoja ympäri aluetta ja tasoittavat korkeuseroja viereisten pisteiden perusteella. Tasoituksessa käytetään painotettua keskiarvoa laajennetun von Neumannin naapuruston avulla saaduista pisteistä. Tämä tarkoittaa sitä, että halutun pisteen horisontaalit ja vertikaalit naapuripisteet sekä niiden takana olevat pisteet otetaan huomioon keskiarvoa laskettaessa. Keskipisteelle annetaan painoarvoksi kolme ja muille yksi. Tällä vältetään maaston liian nopeaa korkeuden muutosta (Doran ja Parberry 2010).

Jokainen tasoitusagentti palaa takaisin lähtöpisteeseen tietyn ajan välein. Tällä saadaan tasoitusagentit toimimaan läheisellä alueella, mikä voi olla hyödyllistä, jos joillakin alueilla tarvitaan enemmän tasoitusta kuin toisilla (Doran ja Parberry 2010).

4.3 Ranta-agentti

Ranta-agenttien tehtävänä on luoda tasaisia hiekkaisia alueita rantaviivan viereen. Ne asettavat itsensä rantaviivan pisteisiin ja suorittavat satunnaisia matkoja, joilla ne tasoittavat maaston muotoja. Ranta-agentit sallivat kuitenkin satunnaista vaihtelua korkeudessa, jotta rannat eivät olisi täysin tasaisia alueita. Kun agentin suorittama matka tulee päätökseen, palaa se takaisin rantaan ja siirtyy rantaviivaa pitkin seuraavan paikkaan, jossa se suorittaa seuraavan matkansa. Jos ranta-agentti jumittuu matkallaan, kuten törmäämällä vuoristoon, siirtyy se satunnaiseen rantaviivan pisteeseen ja jatkaa matkaansa. Ranta-agentit välttelevät korkeita alueita, joten rantaviivan lähellä olevat vuoristot jätetään rauhaan (Doran ja Parberry 2010).

Ranta-agentin yksi tärkeimmistä parametreistä on sen korkeusraja. Sen yläpuolelta ranta-agentit eivät tasoita maastoa. Jos tämä jätetään matalaksi, niin rantojen keskelle voi muodostua mäkiä, mutta korkealla rajalla ne tasoittuvat (Doran ja Parberry 2010).

Toinen muokattava parametri on ranta-agenttien käyttämä korkeusväli, johon ne tasoittavat rantojen korkeuden. Jos tämä väli jätetään pieneksi, niin rannoista tulee tasaisia. Jos se on korkea, niin rantaan muodostuu pieniä korkeuseroja (Doran ja Parberry 2010).

4.4 Vuoriagentti

Vuoriagenttien tehtävänä on luoda vuoret maastoon. Ne aloittavat satunnaisesta pisteestä keskellä maastoa ja valitsevat toivotun suunnan kulkea. Vuoriagentin liikkuessa nostaa se ylösalaisen V muotoisen kiilan, jonka keskipiste on kuljetun reitin kohdalla. Agentti päättää tietyn väliajoin muuttaa suuntaa alle 90 asteen verran alkuperäisestä suunnasta. Tämä saa aikaan reitissä siksak liikettä, mutta agentti liikkuu suurin piirtein samaan suuntaan. Jos agentti kohtaa ylittämättömän kohteen, kuten merenranta, se muuttaa suuntaansa väistääkseen kohteen (Doran ja Parberry 2010).

V-muotoisen kiilan leveys määrää pitkälti vuoren leveyden ja suurelta osin rinteen jyrkkyyden. Rinteiden jyrkkyydet määritellään satunnaisesti jokaisen kiilan kohdalla suunnittelijan määräämältä väliltä. Tämä luo mielenkiintoisia muotoja vuorien rinteille. Vuoriagentitluovat myös pieniä vuoria vuoriston juureen, jotka kulkevat vuoriston kanssa samaan suuntaan. Tasoitus suoritetaan vuoren kasvattamisen jälkeen (Doran ja Parberry 2010).

Vuoriagenteilla on kaikista eniten muokattavia parametrejä. Suunnittelija voi päättää käytettävien agenttien sekä niille annettavien polettien määrän. Vuoriagenteille annetaan myös tieto vuoren maksimikorkeudesta sekä leveydestä ja jyrkkyydestä. Myöskin voidaan muokata, kuinka usein luodaan pienempiä vuoria vuoriston juureen (Doran ja Parberry 2010).

Vuoriagenteista on myös erikoistapaus nimeltä mäkiagentti. Niiden tehtävänä on tuottaa pienempiä vuoristoja alhaisemmilla huipuilla. Mäkiagentin luoman vuoriston juureen ei luoda kuitenkaan pienempiä vuoria. Mäkiagentille annetaan samat tiedot kuin vuoriagentillekin ja ne toimivat samalla tavalla kuin vuoriagentissa (Doran ja Parberry 2010).

4.5 Jokiagentti

Viimeinen Doranin ja Parberryn (2010) esittelemistä agenteista on jokiagentti. Nimensä mukaan niiden tehtävänä on luoda joet maastoon. Yksittäinen agentti luo yhden joen. Jokiagenttien toiminta alkaa valitsemalla satunnainen piste rantaviivalta sekä vuoristosta. Tämän jälkeen agentti kulkee rantaviivalla olevalta pisteeltä vuoristossa olevaan pisteeseen. Agentti valitsee reittinsä maaston kaltevuuksien mukaan, jotta joista ei muodostuisi täysin suoria.

Vuoristosta valitulle pisteelle päästyään agentti aloittaa matkan takaisin rannikolle kuljettua reittiä pitkin. Tässä vaiheessa agentti luo kiilan muotoisen sisennyksen maastoon, samalla tavalla kuin vuoriagentit nostivat vuoret. Kiilan leveys laajenee mitä alemmaksi agentti kulkee. Tämä vaihe päättyy, kun agentti pääsee takaisin alkupisteeseen tai törmää toiseen jokeen.

Jokiagentit välttävät jokia, jotka ovat liian lyhyitä tai eivät ala vuoristosta. Agentti yrittää useaan otteeseen luoda jokea, mutta jos se ei onnistu, niin se luovuttaa riittävän monen yrityksen jälkeen. Tämän takia on mahdollista, että joki agentti ei luo ollenkaan jokea. Suurin syy tälle on se että agentti kohtaa vuoriston, ennen kuin se on saavuttanut vähimmäispituutensa. Pienemmillä kartoilla tämä osoittautuu suuremmaksi ongelmaksi, koska vuoret ovat keskimäärin lähempänä rantaviivaa kuin laajemmissa kartoissa. Vähimmäispituus onkin jokiagentin tärkein muokattava parametri (Doran ja Parberry 2010).

Jokiagentit luovat joet ylämäkeen ja pysähtyvät, kun ne kohtaavat liian korkean maastonkohdan, vaikka päätepistettä ei olisi saavutettu. Tällä estetään joen kulkeminen vuoriston läpi toiselle vuoristolle. Suunnittelija voi kuitenkin säätää maksimikorkeuden parametriä, ennen kuin agentti luovuttaa. Toiset korkeuden parametrit ovat maksimi sallittu korkeus rantaviivan luona sekä vähimmäiskorkeus vuoristosta valitulle pisteelle. Näillä estetään vesiputousten syntyminen merenrantaan sekä jokien alkaminen vuoriston juuresta (Doran ja Parberry 2010).

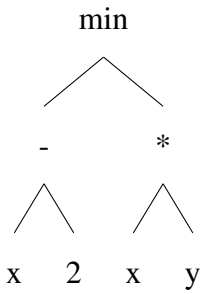
Jokiagentit tarvitsevat myös perääntymisparametrin, mikä aiheuttaa agenttien perääntymisen maksimikorkeuden saavutettua. Agentti perääntyy tietyn määrän pisteitä taaksepäin ja aloittaa vasta silloin joen kaivertamisen. Tällä estetään jokien alkupisteen päätyminen jyrkänteiden päälle (Doran ja Parberry 2010).

5 Hakupohjainen generointi

Aikaisemmin esitellyt maaston generoinnin metodit luovat uskottavan näköisiä maastoja, mutta niistä uupuu suurempi muokattavuus. Rajoitteita tai haluttuja ominaisuuksia on vaikea lisätä maastoihin. Hakupohjaisten (engl. *search-based*) generointimetodien vahvuus on juuri tässä. Ne tarjoavat suuren määrän muokattavuutta generoitavaan maastoon. Seuraavaksi tutkitaan muutamia esimerkkejä hakupohjaisesta generoinnista.

5.1 Geneettinen ohjelmointi

Geneettisessä ohjelmoinnissa ohjelmat kuvataan usein syntaksipuuna (engl. *syntax tree*) eikä koodiriveinä (Poli, Langdon ja McPhee 2008). Esimerkiksi funktiosta min(x-2, x * y) tulee (min (- x 2) (* x y)).



Ohjelman muuttujia ja vakioita (x, 2 ja y) kutsutaan terminaaleiksi (engl. *terminals*). Ohjelman aritmeettisia operaatioita (min, - ja *) kutsutaan puolestaan funktioiksi. Yhdistämällä näitä saadaan geneettisen ohjelmoinnin primitiivinen joukko (Poli, Langdon ja McPhee 2008). Geneettisen ohjelmoinnin joukko muuttuu evolutiivisesti jokaisella ajokerralla lisäämällä ja vaihtamalla funktioita ja terminaaleja sekä yhdistelemällä puun eri osia.

Ong ym. (2005) esittelivät evolutiivisen lähestymistavan maaston generointiin, joka hyödyntää geneettistä algoritmia luodakseen korkeuskartan, jossa on halutunlaisia ominaisuuksia. Heidän metodissa on kaksi vaihetta: maaston siluettivaihe ja korkeuskartan luomisvaihe. Ensimmäisessä vaiheessa ohjelmalle syötetään 2D-kartta, jossa on haluttu maaston muoto. Se voi olla satunnaisesti generoitu tai suunnittelijan tekemä. Ohjelma poistaa luonnottomat muodot maastosta. Tämän jälkeen kartta syötetään toiselle vaiheelle korkeuskartta näyttei-

tä sisältävän tietokannan kanssa. Toisessa vaiheessa ohjelma etsii tietokannasta parhaimman korkeustiedon, mikä sopii ensimmäisessä vaiheessa luotuun karttaan. Tämän metodin ainoat tavat kontrolloida maastoa ovat maaston siluetti sekä tietokannan korkeuskartat.

Frade, Vega ja Cotta (2008) loivat konseptin nimeltä geneettinen maaston ohjelmointi (engl. *Genetic Terrain Programming*) (GTP), jonka tarkoituksena on luoda joka kerta erilainen maasto, mutta samanlaisilla ominaisuuksilla. GTP esitetään geneettisen ohjelmoinnin tavoin syntaksipuuna (genotyyppi), mutta sen tuottama tulos esitetään korkeuskarttana (fenotyyppi). GTP:n funktiojoukko koostuu usein aritmeettisista ja trigonometrisista funktioista sekä exponentti- ja logaritmifunktioista. Terminaalijoukko taas sisältää x- ja y-koordinaatin, kohinafunktion sekä funktiot, jotka ovat riippuvaisia etäisyydestä maaston keskipisteeseen.

Aluksi GTP luo satunnaisesti populaation, josta suunnittelija voi valita yhden tai kaksi yksilöä, joiden perusteella luodaan uudet populaatio ja geneettiset operaatiot. Jos suunnittelija valitsee vain yhden, niin vain mutaatio-operaatio suoritetaan. Suunnittelijan valitessa kaksi yksilöä, yhdistetään ne ja suoritetaan mutaatio-operaatio. Suunnittelija voi päättää kuinka pitkään tätä jatketaan ja voi lopettaa jokaisen kierroksen jälkeen (Frade, Vega ja Cotta 2008).

Koska GTP:ssä genotyypin perusteella tuotetaan fenotyyppi, tarjoaa se todella epäsuoran ja kompaktin korkeuskartan esitystavan. Tämän ansiosta korkeuskarttaa voidaan skaalata loputtomiin, koska se tarkoittaa vain ohjelman ajamisen uusilla koordinaateilla (Shaker, Togelius ja Nelson 2016).

5.2 RTS pelin maaston generointi

Toisen hakupohjaisen generointimetodin esittelivät Togelius, Preuss ja Yannakakis (2010), joka luo kartan RTS (Real-time strategy) pelin vaatimuksien mukaisesti. Se mahdollistaa, että jokainen resurssipiste sekä tukikohta on saavutettavissa. Tämä metodi hyödyntää kahta erilaista esitystapaa.

Epäsuora esitystapa eli genotyyppi on vakion mittainen taulukko. Taulukon mitta määräytyy kartan elementtien määrän ja tyypin mukaan. Neljä erilaista tyyppiä on mahdollista olla:

- Tukikohta: x- ja y-koordinaatit jokaiselle tukikohdalle

- Resurssi 1: x- ja y-koordinaatit jokaiselle resurssi 1:lle
- Resurssi 2: x- ja y-koordinaatit jokaiselle resurssi 2:lle
- Vuori: Jokaiselle vuorelle keskihajonta kolmiulotteisesta normaalijakaumasta, vuoren huipun koordinaatit x ja y sekä painotus parametri, mikä säätää vuorien korkeutta.

Eli, jos kartalle valitaan kaksi tukikohtaa, kolme molempia resurssityyppejä ja 6 vuorta, niin taulukon mitaksi tulee $2*2+3*2+3*2+6*5=46$ (Togelius, Preuss ja Yannakakis 2010).

Metodin suora esitystapa eli fenotyyppi on 100 * 100 ruudukko, jossa jokaisella pisteellä on korkeus väliltä 0 - 99 sekä kolme listaa tukikohtien ja resurssien x- ja y-koordinaateista. Listat saadaan täytettyä genotyypin listan koordinaateista kertomalla ne 100:lla (Togelius, Preuss ja Yannakakis 2010).

6 Käytännön soveltaminen

Jotta tässä tutkimuksessa esiteltyjä metodeja ja algoritmeja voitaisiin hyödyntää optimaalisella tavalla, täytyy ensin miettiä, millaiseen tarkoitukseen maastoa ollaan generoimassa. Kaksi tärkeintä metodin valitsemiseen vaikuttavaa tekijää ovat nopeus ja kontrolli.

Ajonaikaisessa generoimisessa algoritmin nopeus on kaikista tärkein tekijä. Algoritmi ollessa liian hidas, pelaaja saattaa kyllästyä nopeasti, kun hänen täytyy odottaa pelikentän latautumista. Tämä korostuu loputtoman maailman peleissä, jossa maastoa generoidaan koko ajan lisää pelaajan liikkuessa kauemmaksi aloituspisteestä. Tällaisissa tapauksissa kannattaa etsiä sopivaa algoritmia fraktaalipohjaisista metodeista. Niissä esitellyt algoritmit ovat varsin nopeita. Esimerkiksi Minecraft hyödyntää Perlin-kohinaa luodessaan loputonta maastoaan (Shaker, Togelius ja Nelson 2016).

Maastoa voidaan generoida myös pelin tekovaiheessa jo. Siinä tapauksessa maastoa ei generoida pelaajan käyttöön vaan kenttäsuunnittelijoiden avuksi. Maastoa voidaan generoida paljon nopeammin, mitä suunnittelijat voivat käsin tuottaa. Tällöin kontrolli algoritmin tuottamaan maastoon nousee tärkeämmäksi tekijäksi kuin nopeus. Riippuen pelin vaatimuksista, voidaan hyödyntää kaikkia esiteltyjä metodeja. Nopeuttakin kannattaa harkita, jos useampi metodi tuottaa vaatimukset tyydyttävän tuloksen.

Jos peli ei aseta maastolle mitään vaatimuksia, kuten alueiden tavoitettavuus tai vuorien lukumäärä, niin fraktaalipohjaisen generoinnin metodit ovat hyvä valinta, koska ne ovat nopeampia kuin agenttipohjaiset tai hakupohjaiset metodit. Jos taas tarvitaan enemmän kontrollia syntyvien artefaktien määrään, niin agenttipohjainen metodi voi osoittautua hyväksi vaihtoehdoksi. Jos tästä vielä tarvitaan enemmän kontrollia, esimerkiksi tietyt alueet täytyy olla autolla ajettavissa, niin kannattaa kääntyä hakupohjaisten metodien puoleen.

7 Yhteenveto

Tässä tutkielmassa käsiteltiin kolmiulotteisen maaston generoimista proseduraalisesti sekä sen erilaisia metodeja. Monet metodit hyödyntävät korkeuskarttaa maaston esittämiseen tai vähintään esittämisen fenotyyppi on korkeuskartta. Lisäksi pohdittiin näiden metodien hyödyntämisen mahdollisuuksia käytännössä.

Yleisimmin maaston generoinnin menetelmät tukeutuvat kohinoihin. Oikean maaston ollessa fraktaali, hyödynnetään fraktaalipohjaisen generoinnin algoritmeja, joilla voidaan muokata kohinoista fraktaaleja tai luoda suoraan fraktaalikohina. Fraktionaalisella Brownin liikkeellä voidaan luoda kohinoihin fraktaaleja piirteitä. Keskipisteensiirto- ja timanttineliöalgoritmeillä voidaan luoda suoraan fraktaaleja kohinoita. Nämä ovat varsin tehokkaita metodeja, mutta niitä rajoittaa käyttäjän muokattavuus.

Agenttipohjainen generointi tarjoaa suuremman kontrollin maaston luontiin. Suunnittelija voi määrittää monia parametreja, jotka vaikuttavat käytettävien agenttien määrään sekä niiden ominaisuuksiin. Esitellyssä esimerkissä hyödynnettiin viidenlaisia agentteja.

Hakupohjaiset generoinnin metodit tarjoavat vielä asteen enemmän kontrollia maaston luomiseen. Niiden avulla voidaan määrittää tietyn maastonkohdan ominaisuudet. Esimerkiksi tarvitseeko kaksi pistettä olla saavutettavissa toisistaan tai tarvitseeko maastossa olla mahdollista liikkua ajoneuvolla. Monet hakupohjaiset metodit hyödyntävät kahdenlaista esittämistapaa, joista genotyyppi on epäsuora (usein syntaksipuu) ja fenotyyppi korkeuskartta. Tämän ansioista karttaa voidaan skaalata helposti.

Generoinnin algoritmia valittaessa täytyy miettiä pelin asettamia vaatimuksia. Kaksi tärkeintä tekijää ovat nopeus ja kontrolli. Ajonaikaisessa generoinnissa täytyy tehdä valinta nopeuden perusteella, ajon ulkopuolella voidaan taas painottaa kontrollia valintaperusteena.

Lähteet

Bourke, Paul. 1999. “Nearest neighbour weighted interpolation” (joulukuu).

Doran, J., ja I. Parberry. 2010. “Controlled Procedural Terrain Generation Using Software Agents”. *IEEE Transactions on Computational Intelligence and AI in Games* 2 (2): 111–119. https://doi.org/10.1109/TCIAIG.2010.2049020.

Ebert, David S., F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley, William R. Mark ja John C. Hart. 2002. *Texturing and Modeling: A Procedural Approach: Third Edition* [kielellä English (US)]. Elsevier Inc., joulukuu. ISBN: 9781558608481.

Frade, Miguel, F. Fernandez de Vega ja Carlos Cotta. 2008. “Modelling Video Games’ Landscapes by Means of Genetic Terrain Programming - A New Approach for Improving Users’ Experience”. Teoksessa *Applications of Evolutionary Computing,* toimittanut Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni A. Di Caro, Rolf Drechsler, Anikó Ekárt, Anna Isabel Esparcia-Alcázar ym., 485–490. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-78761-7.

Gustavson, Stefan. 2005. “Simplex noise demystified” (tammikuu).

Ong, Teong Joo, Ryan Saunders, John Keyser ja John J. Leggett. 2005. “Terrain Generation Using Genetic Algorithms”. Teoksessa *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation,* 1463–1470. GECCO ’05. Washington DC, USA: Association for Computing Machinery. ISBN: 1595930108. https://doi.org/10.1145/1068009.1068241. https://doi.org/10.1145/1068009.1068241.

Poli, Riccardo, William B. Langdon ja Nicholas Freitag McPhee. 2008. *A field guide to genetic programming.* (With contributions by J. R. Koza). Published via `http://lulu.com` / freely available at `http://www.gp-field-guide.org.uk`. http://dl.acm.org/citation.cfm?id=1796422.

Rudowsky, Ira S. 2004. *Intelligent Agents.*

Shaker, Noor, Julian Togelius ja Mark Nelson. 2016. “Fractals, noise and agents with applications to landscapes”, 57–72. Lokakuu. ISBN: 978-3-319-42714-0. https://doi.org/10.1007/978-3-319-42716-4_4.

Smelik, Ruben, Klaas Jan de Kraker, Saskia Groenewegen, Tim Tutenel ja Rafael Bidarra. 2009. “A Survey of Procedural Methods for Terrain Modelling”. Kesäkuu.

Stuart, Russell, ja Norvig Peter. 1995. *Artificial intelligence : a modern approach* [kielellä English (US)]. Prentice-Hall Inc. ISBN: 0-13-103805-2.

Togelius, Julian, Mike Preuss ja Georgios N. Yannakakis. 2010. “Towards Multiobjective Procedural Map Generation”. Teoksessa *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. PCGames ’10. Monterey, California: Association for Computing Machinery. ISBN: 9781450300230. https://doi.org/10.1145/1814256.1814259. https://doi.org/10.1145/1814256.1814259.

Travis, Archer. 2011. *Procedurally Generating Terrains*. Tekninen raportti.