Tampereen ammattikorkeakoulu

Lineaarisen pelin kenttien luominen proseduraalisella generoinnilla

Jaakko Leskelä

OPINNÄYTETYÖ
Kesäkuu 2023

Tietojenkäsittelyn tutkinto-ohjelma
Game Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Game Production

Leskelä, Jaakko
Lineaarisen pelin kenttien luominen proseduraalisella generoinnilla

Opinnäytetyö 29 sivua
Kesäkuu 2023

Opinnäytetyössä tutkittiin, miten proseduraalista generointia voisi hyödyntää kenttien luomiseen lineaarisessa pelissä. Opinnäytetyössä selvitettiin, miksi proseduraalista generointia harvemmin käytetään lineaarisien pelien kenttien luontiin ja mitkä ovat sen vahvuudet ja heikkoudet. Lisäksi opinnäytetyössä tutkittiin, kuinka helppoa proseduraalinen generointi olisi ottaa käyttöön. Tämä selvitettiin tutkimalla erilaisia proseduraalisen generoinnin työkaluja ja algoritmeja. Algoritmeista valittiin yksi ja lopuksi tehtiin yksinkertainen prototyyppi Unity-pelimoottorissa hyödyntämällä valmiiksi tehtyä työkalua.

Opinnäytetyössä analysoitiin useampia algoritmeja ja niiden toimintaa. Lisäksi tutkittiin algoritmien yleisimpiä käyttötarkoituksia. Algoritmien käyttötarkoituksien ja käyttöönoton helppouden vertailun perusteella valittiin opinnäytetyössä käytettävä algoritmi WaveFunctionCollapse. Opinnäytetyössä käsiteltiin WaveFunctionCollapse- algoritmia hyödyntävän työkalun toimintaa ja tärkeimpiä siihen liittyviä skriptejä. Opinnäytetyössä selvitettiin, miksi proseduraalisen generoinnin työkalujen käyttö voi olla haastavaa.

Opinnäytetyössä huomattiin, kuinka tasojen luomiseen käytettävät 3D-mallit tulee suunnitella proseduraalisen generoinnin työkaluja varten. Opinnäytetyön prototyypin tuloksena oli, että kenttien luonti proseduraalisella generoinnilla vaatii laajaa ymmärrystä työkalujen toiminnasta, varsinkin, jos työkaluja haluaa muokata. Opinnäytetyössä huomattiin, kuinka eri algoritmit erikoistuvat erilaisten kenttien luomiseen. Opinnäytetyössä pohdittiin, kuinka algoritmien toimintaa voisi kehittää erityisesti lineaarisien pelien kenttien luomiseen.

Asiasanat: proseduraalinen generointi, algoritmit, kenttien luonti

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Production

Leskelä, Jaakko:
Creating Levels For Linear Games Using Procedural Generation

Bachelor's thesis 29 pages
June 2023

The objective of this thesis was to gather information on how procedural generation can be used to create game levels. The aim was to find out why procedural generation is rarely used in linear games and discover the strengths and weaknesses of procedural generation in level creation.

The thesis studied focuses on how easy it is to start using procedural generation in a game project. In this thesis, multiple procedural generation tools and algorithms were compared with each other based on their most common use cases and the ease of starting out. In the thesis, algorithm called WaveFunctionCollapse was chosen for a prototype that was created using Unity game engine.

The thesis work used 3D-models for procedural generation. It was crucial for good results that the 3D-models were either specifically designed to be used with procedural generation or at the very least designed to fit in a uniform grid. The result of this thesis is that using procedural generation to create levels requires good knowledge of level design as it uses pre-made rules to create levels and needs clear rules to succeed.

Key words: procedural generation, algorithms, level creation

SISÄLLYS

LYHENTEET JA TERMIT

CA	Cellular Automata-algoritmi
Entropia	Proseduraalisen generoinnin ruudukossa olevan solun vaihtoehtojen määrä eli, kuinka lähellä solu on sen lopputulosta
Lineaarinen	Suoraviivainen – tässä asiayhteydessä tarkoittaa peliä, jonka osat suoritetaan tietyssä järjestyksessä
Skripti	Tietokoneen komentosarja, jonka ohjelma suorittaa.
Solu	Yksittäinen ruutu ruudukossa
Tuloste	Proseduraalisen generoinnin lopputulos
WFC	Wave Function Collapse-algoritmi

1 JOHDANTO

Proseduraalinen generointi on ollut jo vuosikymmeniä osana pelien kehitystä. Yksi ensimmäisistä proseduraalisesti generoiduista pelimaailmoista oli nimittäin jo vuonna 1980 julkaistussa pelissä nimeltään Rogue. Myös nykypäivänä proseduraalinen generointi on olennaisena osana erityisesti open world pelimaailmojen luontia esimerkiksi peleissä Valheim ja No Man’s Sky.

Pelien kehittäminen voi olla uuvuttavaa ja todella aikaa vievää. Niinpä pelien kehittämisessä käytetään työkaluja, jotka vähentävät työmäärää ja tehostavat pelien kehitystä. Tämä opinnäytetyö keskittyy yhteen näistä työkaluista, joka on siis proseduraalinen generointi. Proseduraalinen generointi, englanniksi procedural generation, tarkoittaa prosessia, jossa algoritmien perusteella luodaan automaattisesti sisältöä. Vaikka proseduraalista generointia voi hyödyntää useassa eri käyttötarkoituksessa, tässä opinnäytetyössä keskitytään kenttien luomiseen, mikä on olennainen osa videopelien kehitystä. Tarkemmin sanottuna tässä opinnäytetyössä keskitytään kenttien luomiseen lineaarista peliä varten hyödyntäen proseduraalista generointia. Lineaarinen yksinkertaisimmillaan tarkoittaa viivan kaltaista tai suoraviivaista, mikä pelien tarkoituksessa tarkoittaa peliä, jossa pelin eri osat täytyy suorittaa tietyssä järjestyksessä pelin läpäisemiseksi. Tämän järjestyksen päättää siis yleisimmin pelinkehittäjä.

Opinnäytetyön tehtävänä on tutkia miten proseduraalista generointia voi hyödyntää lineaarisen pelin kenttien luomisessa. Tämän tutkimisessa on myös olennaisena selvittää mitä mahdollisia vahvuuksia tai heikkouksia proseduraalisen generoinnin käytössä on verrattuna manuaaliseen kenttien luomiseen.

Opinnäytetyössä aluksi käsitellään proseduraalisen generoinnin merkitystä pelituotannossa esimerkki pelien avulla. Tämän jälkeen vertaillaan proseduraaliseen generointiin käytettäviä algoritmeja ja valitaan opinnäytetyötä varten sopiva algoritmi ja selitetään algoritmin toimintaa tarkemmin. Tämän jälkeen opinnäytetyössä tehdään yksinkertaistettu toiminnallinen prototyyppi Unity-pelimoottorin avulla. Prototyypin jälkeen tehdään johtopäätöksiä proseduraalisen generoinnin

merkityksestä erityisesti lineaaristen pelien näkökulmasta sekä käydään läpi mahdollisia jatkoehdotuksia.

Proseduraalisella generoinnilla on monenlaisia eri käyttötarkoituksia. Niinpä tässä opinnäytetyössä proseduraalisen generoinnin käyttö rajataan lineaariseen peliin sekä ainoastaan peliympäristön luontiin. Lisäksi peliympäristöjä luodaan ainoastaan valmiista 3D -malleista eikä niiden mallintamista käydä läpi muuta kuin proseduraalisen generoinnin osalta merkitykselliset seikat. Pelin osalta ei käsitellä pelimekaanikoita, sillä opinnäytetyössä käydään läpi vain yksinkertaisen prototyypin avulla kenttien luonti, missä ei oteta huomioon pelimekaanikoiden mahdollista vaikutusta kenttien tyyliin. Opinnäytetyön kannalta tärkeää on myös huomioida, että proseduraaliseen generointiin liittyvää sisältöä on niukasti sekä sisältö on monesti kokeellista.

2 ERILAISET PROSEDURAALISEN GENEROINNIN TYÖKALUT JA NIIDEN KÄYTTÖ

2.1 Proseduraalisuuden määritelmä ja yleiset käyttötarkoitukset

Sana proseduuri tarkoittaa ohjelmoinnin parissa itsenäistä ohjelmointimoduulia, joka toteuttaa jonkin konkreettisen tehtävän osana suurempaa lähdekoodin koodikokonaisuutta (Techopedia 2018). Tästä johdetaan adjektiivi proseduraalinen, joka kuvaa toimintatapaa eli esimerkiksi tässä tapauksessa se tarjoaa lisätietoa generoinnista. Proseduraalinen generointi on tapa luoda dataa tai sisältöä tietyn algoritmin tai logiikan perusteella, manuaalisen ihmisen tekemän työn sijaan (Sarawagi, n.d). Eli manuaalisen ihmisen tekemän työn sijaan, aikaa käytetään työkalun oppimiseen ja kehittämiseen, mikä parhaassa tapauksessa hoitaa sisällön luomisen ihmisen puolesta kokonaan.

Proseduraalisesti yleensä luodaan sisältöä kuten maastoja, 3D malleja, hahmoja, animaatioita esimerkiksi videopeleihin ja animaatioelokuviin. (Brummelen & Chen, n.d). Proseduraalisuus on löytänyt myös tiensä muun muassa tekstuurien luomiseen, jolloin tekstuurit kuvataan sen luomiseen tarvittavan proseduurin avulla. Sen etu verrattuna perinteiseen bittikarttatekstuuriin on sen skaalautuminen loputtomasti yksityiskohtia menettämättä (McCombs, n.d). Proseduraalinen tekstuurien generointi toimii erityisen hyvin proseduraalisesti luotujen maastojen kanssa, sillä molemmat voivat teoriassa skaalautua loputtomiin.

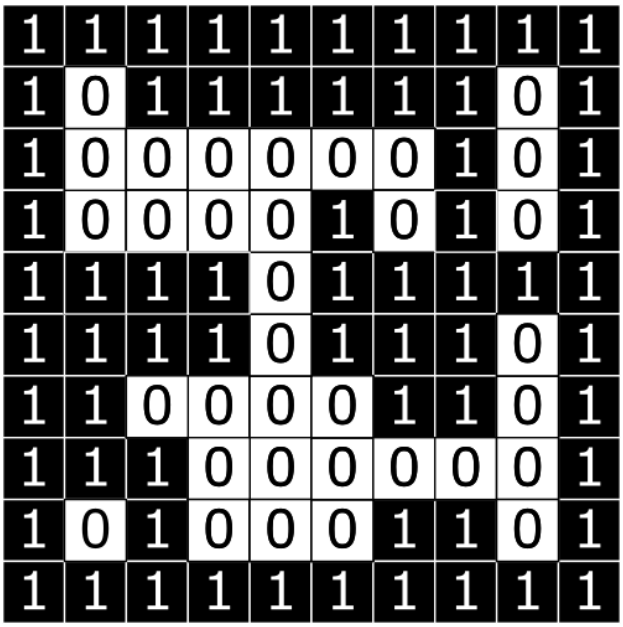
2.2 Työkalujen / algoritmien esittely

Tässä luvussa tutkitaan erilaisia työkaluja sekä algoritmeja mitä voidaan hyödyntää proseduraalisessa generoinnissa. Lisäksi käydään läpi niiden vahvuuksia ja yleisimpiä käyttötarkoituksia.

2.2.1 Cellular automata

Cellular automaton on proseduraalisen generoinnin työkalu, joka perustuu säännölliseen taulukkoon, joka koostuu soluista. Cellular automata on vain tästä

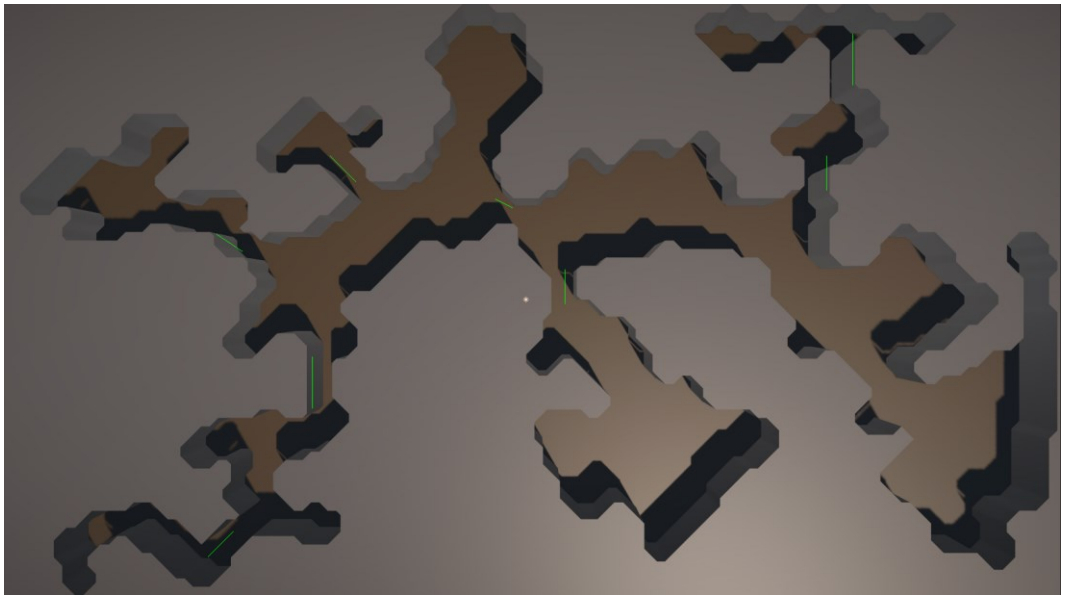
monikko, mitä käytetään tässä opinnäytetyössä. Tässä opinnäytetyössä cellular automata tullaan tästä lähtien lyhentämään käyttäen lyhennettä CA. Wikidot verkkosivun (2018) mukaan CA:ssa on taulukollinen soluja, joista jokaisella on samat säännöt ja niiden tila määritellään yhtä aikaa, vertaillen myös vieressä olevia soluja. Kuvassa 1 näkyy esimerkki CA:lla luodusta luolastosta, jossa solun tilat ovat yleensä yksi tai nolla eli päällä tai pois päältä. Algoritmin ainoaksi säännöksi on asetettu, että reunoilla täytyy olla päällä oleva solu. Myös poikkeustapauksia tai välimalleja voi keksiä tarpeen mukaan, mutta silloinkin joka solulla täytyy olla samat tilat käytettävissä. Lisäksi jotta kyseessä olisi CA täytyy tiloja olla laskettava määrä.



Kuva 1. CA algoritmi visualisoituna.

Cellular automata on itsessään aika yksinkertainen, mutta siitä voi saada monipuolisen erilaisilla muunnoksilla. Esimerkiksi noise-tekstuureja tai algoritmeja voi hyödyntää, jotka antavat enemmän hallintaa ohjelmoijalle ja auttavat visualisoimaan lopputuloksen. Luonnollisten luolastojen luomisessa erityisen hyödyllinen algoritmi on Perlin Noise, josta kerrotaan lisää myöhemmin sen omassa osiossa. Lisää hallintaa lopputulokseen saa, kun taulukkoa siistii useamman kertaa eri sääntöjä käyttäen. CA on usein käytössä luolastojen luomisessa, sillä se luo orgaanisen näköisiä kuvioita. (Wikidot. 2018.)

Lähtötilanteessa Cellular Automatassa kaikki solut ovat joko päällä tai pois. Ensimmäisellä kerralla voi luoda karkean pohjan, minkä seurauksena voi syntyä erillään olevia huoneita tai yksittäisiä palasia erillään muista, tietysti riippuen ensimmäisen kerran säännöistä. Kuvassa 2 nähdään, kuinka toisella kerralla voi esimerkiksi luoda reittejä huoneiden välille. Vihreillä viivoilla on merkitty kohdat, joihin algoritmi on luonut reitit. Tätä prosessia voi jatkaa erilaisia sääntöjä lisäämällä, kunnes lopputulos on tarpeen mukainen. Tässä mainittiin vain hyvin yksinkertainen luolaston siistiminen, mutta mahdollisuuksia on runsaasti.



Kuva 2. CA:lla luotu luolasto.

2.2.2 Wave Function Collapse

Maxim Guminin kehittämä WaveFunctionCollapse on kuvien generointi algoritmi. WaveFunctionCollapsea tullaan käyttämään nimellä WFC tässä opinnäytetyössä, sillä kyseinen nimitys on myös yleisessä käytössä proseduraalisen generoinnin yhteisössä. Isaac Karthin artikkelin (2017) mukaan WFC:ssä uudet kuvat generoidaan sille annettujen syötteenä toimivien esimerkkikuvien mukaiseen tyyliin. Tämä varmistuu, sillä algoritmi vaatii, että uuden kuvan jokainen osa löytyy esimerkkikuvasta.

WFC toimii non-backtracking, greedy search - toimintaperiaatteella. Paul Blackin Dictionary of Algorithms and Data Structures (2005) mukaan greedy algoritm on algoritmi, joka valitsee joko parhaan välittömän tuloksen tai lokaalin ratkaisun,

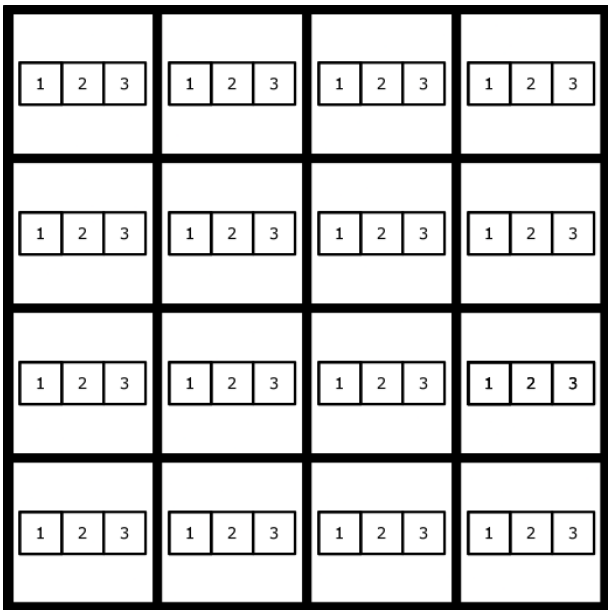
eikä tuota välttämättä optimaalisinta tulosta suuremmassa mittakaavassa. Kyseisen algoritmin vahvuus on kuitenkin sen nopeus ja vaikka lopputulos ei välttämättä ole optimaalisin, se on kuitenkin likimäärin lähellä optimaalisinta lopputulosta. Non-backtracking kuvaa algoritmin toimintaa virheen kohdatessaan. Kun algoritmi törmää tilanteeseen, jossa jokin algoritmin säännöistä ei toteudu, non-backtracking algoritmi lopettaa toimintansa. Backtracking algoritmi taas palautuisi edelliseen tilaan, jossa algoritmin säännöt edelleen pätevät ja yrittäisi uudelleen.

WFC:n toiminta muistuttaa hieman sudokun täyttämistä. Sudokussa on tietyt säännöt, minkä mukaan sudoku täytetään. Jokainen numero mikä sudokuun täytetään, vähentää muiden ruutujen mahdollisia ratkaisuja, kunnes jäljelle jää enää ratkaistu sudoku. Sudokun ratkaisussa on olennaista löytää ruutu, jossa on mahdollisimman vähän mahdollisia vastauksia. Jos ruudulla on vain yksi mahdollinen vastaus, on helppo tietää, että vastaus on oikein. Muuten sudokun täyttäjä voi kokeilla arvaamista, jos mahdollisia vastauksia ruutuun on kaksi tai enemmän ja jatkaa siitä, kunnes sudoku on joko ratkaistu tai sääntörikkomus esiintyy. Sitten sudokun täyttäjä voisi palata tilanteeseen, jossa sääntörikkomusta ei ollut vielä tapahtunut ja kokeilla uudestaan. Tämä olisi back-tracking toteutusta sudokun täyttämisessä. WFC ei käytä backtrackingiä, joka voisi olla hyödyllinen lisä WFC:n toimintaan.

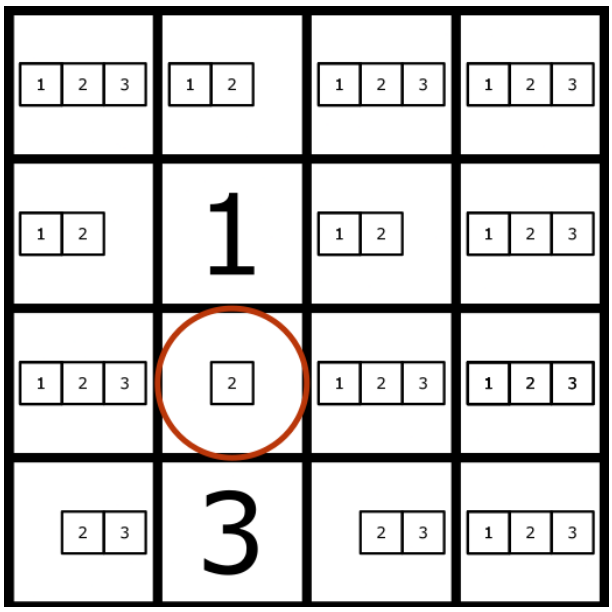
WFC:ssä jokaisella solulla on aluksi samat mahdolliset vastaukset, kuten kuvassa 3 nähdään. Kuvassa 3 soluille on määritelty kolme mahdollista vastausta: yksi, kaksi ja kolme. Näistä numero yksi voi olla numeron yksi ja kaksi vieressä. Numero kaksi voi olla minkä tahansa muun numeron vieressä ja numero kolme taas voi olla numeron kolme tai kaksi vieressä. WFC arpoo aluksi yhdelle satunnaiselle solulle vastauksen sille annetuista vaihtoehdoista eli silloin kyseinen ruutu on sen lopullisessa lopputuloksessa, eikä sitä enää muuteta. Englanniksi tästä käytetään nimitystä collapse, joka tarkoittaa solun päätymistä lopulliseen ratkaisuun. Tämän jälkeen tämän solun tilanne levitetään viereisille soluille, jotka myös päivittävät tilansa, kunnes jokainen solu, johon muutos vaikuttaa on päivittänyt tilanteensa. Mitä enemmän algoritmilla on mahdollisia vastauksia, sitä hitaammin algoritmi suoriutuu.

Seuraavaksi WFC:ssä valitaan solun, jolla on vähiten mahdollisia vaihtoehtoja eli pienin entropia. Mitä pienempi entropia solulla on, sitä vähemmän sillä on mahdollisia vastauksia ja sitä lähempänä solu on sen lopputulosta. Jos useammalla solulla on yhtä pieni entropia, niistä arvotaan yksi, jolle sitten arvotaan lopputulos sille mahdollisista vaihtoehdoista. Kuvassa 4 nähdään tilanne, jossa yhdellä soluista on pienin entropia sen viereisien solujen takia eli tämä solu päätyisi seuraavaksi sen lopputulokseen. Kyseinen solu on merkitty kuvassa 4 punaisella ympyrällä. Tämän jälkeen taas päivitetään kaikkien solujen tila, johon tämä muutos vaikuttaa. Ja tätä prosessia toistetaan, kunnes kaikkien solujen entropia on yksi, eli kaikki solut ovat niiden lopullisessa tilanteessa tai kunnes törmää ristiriitaan, jossa algoritmin säännöt eivät päde.

Maxim Gumin kertoo WFC:n Github sivulla (2022) huomanneensa, että ihmiset piirtävän usein minimaalisen entropian perusteella eli ihmiset piirtävät ensiksi asiat, joissa on vähiten mahdollisuuksia. Eli vaikka WFC:ssä välillä arvotaan soluille niiden lopullinen tila, WFC:llä saadaan aikaan luonnollisen näköisiä lopputuloksia, sillä WFC käyttää minimalistisen entropian heuristiikkaa.



Kuva 3. WFC-algoritmin ruudukon alkutilanne.



Kuva 4. WFC-algoritmin ruudukon esimerkkitilanne.

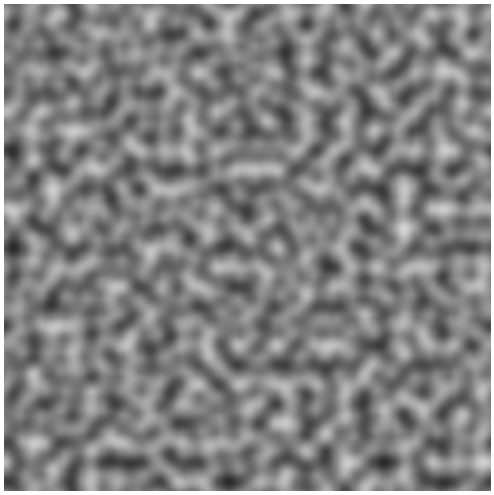
2.2.3 Perlin Noise ja korkeuskartat

Perlin Noise on Ken Perlinin kehittämä proseduraalinen noise-tekstuuri, jonka tarkoituksena on kasvattaa realismin määrää tietokonegrafiikoissa. Perlin kehitti sen vuonna 1983, sillä hänen mielestään CGI, eli computer-generated imagery, näytti liian konemaiselta. (Wikipedia. 2022.)

Perlin Noise on hyödyllinen työkalu pelien tekemisessä. Perlin Noisella voidaan tehdä esimerkiksi animaatioista luonnollisemman näköisiä tai luoda erilaisia proseduraalisia tekstuureja. Tässä opinnäytetyössä kuitenkin tärkein Perlin Noisen mahdollinen käyttötarkoitus on Perlin Noisen hyödyntäminen korkeuskarttana.

Korkeuskartat ovat mustavalkoisia kuvia, joissa jokainen pikseli kuvaa maaston korkeutta. Tummat pikselin arvot tarkoittavat matalaa korkeutta, kun taas vaaleat arvot tarkoittavat korkeita. (3D-Mapper n.d.) Korkeuskarttana voisi teoriassa käyttää mitä tahansa mustavalkoista kuvaa, mutta parhaat tulokset syntyvät, kun korkeuskarttoina hyödynnetään kuvia, joissa esiintyy luonnollisia kuvioita. Korkeuskartan korkeuksien muutokset ovat riippuvaisia mustan ja valkoisen sävyjen määristä. Jos esimerkiksi mustan ja valkoisen sävyjä on 256, voi maastolla olla 256 eri korkeutta.

Kuvasta 5 nähdään kuinka Perlin Noisella luodaan noise-tekstuureja, joissa esiintyy luonnollisia kuvioita. Tämän avulla voidaan luoda luonnollisen näköisiä maastoja. Kun korkeuskarttoja käytetään pelikentän tekemisessä, on olennaista, että korkeusmuutokset eivät ole liian radikaaleja, ja tässä Perlin Noise on erinomainen.



Kuva 5. Esimerkki kuva Perlin Noise – tekstuurista.

2.2.4 Model Synthesis

Paul Merrellin kehittämä Model Synthesis (2023) on esimerkki mallin pohjalta suurien monimutkaisten 2D- sekä 3D-mallien generointiin käytettävä tekniikka. Model Synthesis ottaa vastaan esimerkkimallin ja tämän mallin perusteella luo suuremman mallin, joka muistuttaa esimerkkimallia.

Model Synthesis tekniikan inspiraationa on toiminut texture synthesis, mitä myös edellä mainittu Wave Function Collapse alkuperäisessä muodossaan myös on. WFC ei ole kuitenkaan toiminut Model Synthesis tekniikan inspiraationa, sillä Model Synthesis kehitettiin vuonna 2007 ja WFC vasta vuonna 2016.

Model Synthesis on yleistarkoituksellinen työkalu, joten sen käyttö ei ole rajattua. Model Synthesis ei vaadi erikseen laadittuja sääntöjä, mutta se täytyy toteuttaa säännöllisessä ruudukossa, mikä ei toisaalta ole ongelma, sillä peleissä kentät yleensä luodaan ruudukkoihin muutenkin. Model Synthesis algoritmissa esimerkkimalli jaetaan palasiin jokaisen ruudukossa olevan solun pohjalta. Tämän perusteella mallille luodaan säännöt mallin osien vierekkäisyydelle engl.(adjacency

constraint) .Tämä kertoo, mitkä osat voivat olla yhteydessä muihin osiin. Model Synthesis algoritmin toiminta on hyvin samanlainen WFC:n kanssa, joten siihen ei tässä opinnäytetyössä paneuduta tämän enempää.

2.3 Algoritmien vertailu ja selitys mihin algoritmiin päädyin ja miksi

Olennaista tämän opinnäytetyön kannalta oli valita algoritmi, joka on helppo ottaa käyttöön ja on hyvä valmiiden mallien hyödyntämisessä. Lineaariset pelit ovat usein tarinapohjaisia ja pelintekijä pyrkii luomaan tarkoin suunnitellun pelikokemuksen. Pelin miljööstä halutaan tietynlainen ja pelin teeman pitää säilyä algoritmin käytöstä huolimatta. Valitun algoritmin avulla kentästä pyritään luomaan sulava kokonaisuus isossa skaalassa ilman suurempaa manuaalista työtä. Jokaiselle tässä opinnäytetyössä mainitulle algoritmille tai työkalulle on käyttönsä eikä tässä opinnäytetyössä määritellä yhtä ainoaa oikeaa vaihtoehtoa.

Cellular Automata on helposti käyttöön otettava algoritmi ja sen koodi yksinkertaisimmillaan on myös helppo ymmärtää. Kuten taulukosta 1 nähdään, on CA:n käyttökohteena kuitenkin yleensä luolastot tai maasto. Tätä voisi käyttää osana kenttien luomista, myös lineaarisissa peleissä, mutta tässä opinnäytetyössä haluan keskittyä algoritmiin, jolla on useita käyttötarkoituksia.

Perlin Noise ja korkeuskartat ovat hyviä maastojen luomiseen ja maastot ovatkin isossa osassa montaa peliä. Tässä opinnäytetyössä keskitytään kuitenkin 3D-mallien asetteluun ja yhdistelyyn, joten kyseinen työkaluja ei ollut prototyypissä käytössä.

Paul Merrell on kirjoittanut Comparing Model Synthesis and Wave Function Collapse – artikkelin (2021), jossa hän vertailee Model Synthesiä ja WFC:tä keskenään. Merrellin mukaan molemmat algoritmit toimivat todella samalla tavalla. Model Synthesis ratkaisee suuremmat ruudukot pienemmissä osissa, mikä on Merrellin mukaan tärkeää erityisesti 3D-malleja käyttäessä, sillä ne ovat monimutkaisempia. Pienemmissä osissa ratkaisu vähentää tapauksia, joissa algoritmi ei pysty ratkaisemaan ruudukosta lopputulosta. Ulkopuolinen vertailija voisi mahdollisesti löytää enemmän eroavaisuuksia ja olla objektiivisempi.

WFC ratkaisee ruudukon minimalistisen entropia heuristiikan perusteella, mikä selitettiin WFC:n esittelyosiossa, kun Model Synthesis taas ratkaisee ruudukon järjestyksessä. Model Synthesis olisi ollut potentiaalinen vaihtoehto tähän opinnäytetyöhön, mutta se on ohjelmoitu c++ ohjelmointikielellä, kun taas Unityssä käytetään c# - ohjelmointikieltä. Unityyn on myös tehty valmiiksi WFC työkalu, joten helppouden vuoksi päädyin Wave Function Collapseen.

	Wave Function Collapse	Cellular Automata	Perlin Noise / Korkeuskartat	Model Synthesis
Yleisin käyttökohde kenttien luonnissa	Valmiit 2D tai 3D-mallit ja niistä mallin näköisien suurempien kokonaisuuksien luominen.	Luolastot ja maasto.	Maasto / luolastot 3D- Perlin Noisella	Valmiit 3D-mallit ja niistä mallin näköisien kokonaisuuksien luominen.
Ohjelmointikieli	C# (2D – versio kirjoitettu myös muilla kielillä.)	Modernit ohjelmointikielet. (Java, c++, Python jne.)	Modernit ohjelmointikielet. (Java, c++, Python jne.)	C++ (Mahdollista myös kirjoittaa muilla kielillä.)
Käyttöönotto	Vaatii valmiit esimerkkimallit. Unityyn on yksinkertainen WFC työkalu valmiiksi luotuna.	Helppo, ei vaadi valmiita malleja.	Helppo, ei vaadi valmiita malleja.	Vaatii valmiit esimerkkimallit.
Monimutkaisuus	Kokeilemalla voi saada mielenkiintoisia tuloksia, mutta algoritmin ymmärrys vaatii enemmän aikaa.	Yksinkertaisimmillaan helppo ymmärtää.	Yksinkertaisimmillaan helppo ymmärtää.	Algoritmille löytyy hyviä selitysvideoita, mutta algoritmi on monimutkainen.
Mallit	Vaatii valmiit 2D - tai 3D- mallit, jotka ovat mallinnettu algoritmin toimintaa ajatellen.	Malli luodaan koodilla. Ei vaadi valmiita 3D-malleja.	Malli luodaan koodilla. Ei vaadi valmiita 3D-malleja.	Vaatii valmiit 2D - tai 3D- mallit, jotka ovat mal-

				linnettu algoritmin toimintaa ajatellen.

Taulukko 1. Proseduraalisen generoinnin algoritmien ja työkalujen vertailu.

3 PROSEDURAALISEN GENEROINNIN TYÖKALUN HYÖDYNTÄMINEN KÄYTÄNNÖSSÄ

3.1 Proseduraalisen generoinnin työkalun käyttöönotto Unity-pelimoottorissa

Tässä osiossa käydään läpi WFC-työkalun käyttöönottoa Unity projektissa sekä sen toimintoja. Tässä opinnäytetyössä käytämme unity-wave-function-collapse nimistä Unity-pakettia, jonka on kehittänyt itch.io käyttäjä Selfsame eli oikealta nimeltään Joseph Allen Parker, joka kehitti työkalun alkuperäisen Maxim Guminin WFC:n pohjalta. Kyseinen työkalu hyödyntää WFC-algoritmia, mutta värien sijaan se käyttää 3D-malleja. Tästä työkalusta puhutaan tässä opinnäytetyössä WFC-työkaluna.

WFC-työkalun saa ladattua itch.io sivulta ilmaiseksi ja sen asennus on yksinkertainen. Ensiksi ladataan WFC-työkalusta viimeisin versio tietokoneelle. Tämän jälkeen WFC-työkalu pitää lisätä Unity-projektiin. Tämä onnistuu painamalla Unityn editorissa ylävalikon Window-painiketta ja sen jälkeen painamalla Package Manager. Package Manager osiossa painetaan ikkunan vasemmasta ylälaidasta plusmerkin sisältävää painiketta, josta valitaan “add package from disk”, jonka jälkeen valitaan aiemmin ladattu WFC-työkalu. Tämän jälkeen WFC-työkalu on lisätty osaksi projektia.

3.1.1 Proseduraalisen generoinnin työkalun algoritmimallit

SimpleTiledWFC on skripti, joka on toinen WFC-malleista, joka löytyy työkalusta. Skriptin toiminta perustuu vierekkäisten solujen välisiin sääntöihin. Tämän takia esimerkki syötteeseen pyritään maalaamaan eri 3D-mallien yhteensopivuus, jonka perusteella SimpleTiledWFC pyrkii generoimaan erilaiset yhdistelmät, mitä 3D-mallien vierekkäisyys säännöt sallivat. SimpleTiledWFC skripti on nimensä mukaisesti aika yksinkertainen, mutta monimutkaista sen käytöstä tekee se, kuinka sille syötettyjen 3D-mallien pitää olla lähestulkoon täydellisesti yhteensopivia. Toisaalta lineaarisissa peleissä ympäristön 3D-mallit voivat jo valmiiksi olla yhteensopivia WFC:n näkökulmasta. Prototyyppi mielessä SimpleTiledWFC ei

ole paras vaihtoehto, sillä yhteensopivien mallien tekeminen vie aikaa. Pelien 3D-mallit usein luodaan muutenkin ruudukkoon sopiviksi, jolloin niitä voi käyttää yksinkertaisten prototyyppien luomiseen, mutta ne eivät välttämättä sovi edistyneempään käyttöön.

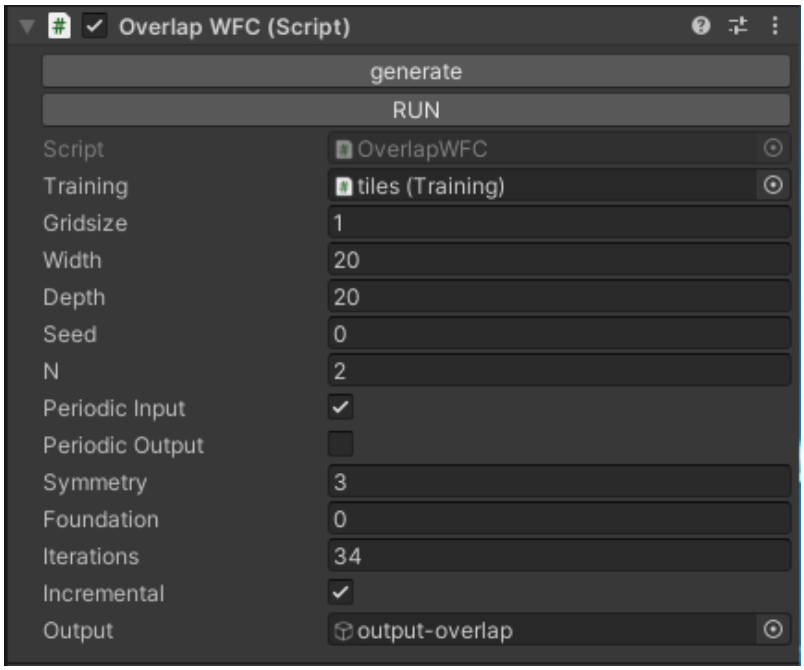
SimpleTiledWFC-skriptin toimintaa voitaisiin myös viedä pidemmälle erilaisina rajoitteina tai sääntöinä, joiden avulla skripti pystyy luomaan monimutkaisia kokonaisuuksia, jotka ovat silti yhtenäisiä. Sääntöjen luominen voi kuitenkin olla hankalaa, koska se vaatii laajaa ymmärrystä algoritmin toiminnasta ja 3D-malleissa täytyy ottaa huomioon useita poikkeuksia. Lisäksi sääntöjen täytyy sopia yhteen tai muuten algoritmi ei toimi.

OverlapWFC on skripti, joka on toinen WFC-malleista. Sen lähestymistapa WFC-algoritmin toteuttamiseen on hyvin erilainen verrattuna SimpleTiledWFC-skriptiin. OverlapWFC muodostaa mielenkiintoisia kokonaisuuksia, jotka noudattavat syötteessä esiintyviä kuvioita. OverlapWFC ei siis toimi vierekkäisyys sääntöjen mukaan. OverlapWFC-skriptin tuloste matkii syötettä niin, että mikä tahansa kohta mikä on syötteessä, täytyy löytyä generoinnin lopputuloksesta. OverlapWFC-skripti antaa huomattavasti enemmän pelivaraa 3D-mallien osalta ja kenttien kehittäjä voi saada yrityksen ja erehdyksen kautta mielenkiintoisia lopputuloksia aikaiseksi, ilman täydellisesti yhteensopivia malleja.

Kuvassa 6 näkyy OverlapWFC-skriptin parametrit, joita käydään läpi seuraavaksi. Gridsize muokkaa, minkä kokoisia 3D-malleja työkalun ruudukkoon voidaan syöttää. Oletuksena on yksi, joka tarkoittaa siis metri kertaa metri kokoisia 3D-malleja tai toisin sanoen 1x1 Unity-yksikön kokoisia 3D-malleja. Tämän jälkeen parametreina ovat Width ja Depth eli tulosteen leveys ja syvyys, mutta käytännössä syvyyden voi ajatella korkeutena, jos lopputulos ajatellaan ylhäältäpäin kuvattuna karttana, joka koostuu 3D-malleista. Seed parametri muokkaa lopputuloksen satunnaisuutta. Jos Seed muuttuja on nolla, algoritmi tekee satunnaisen kuvion. Jos Seed muuttuja mikä tahansa muu numero, algoritmi tekee saman kuvion jokaisella kerralla, jos syötettä ei muokata.

Kuvassa 6 esiintyvä parametri N tarkoittaa algoritmin kuvioiden kokoa eli syötteestä muodostettavien toistuvien kuvioiden kokoa. Tätä ei suositella erityisemmin muutettavan, mutta pieniä lukuja voi kokeilla erilaisten lopputuloksien saavuttamiseksi. Kuvan 6 muuttuja Periodic Input tarkoittaa syötteestä muodostettujen kuvioiden toistamista ja Periodic Output taas tarkoittaa lopputuloksessa toistuvaa kuvioita. Periodic Input parametri on hyvä olla käytössä aina, mutta Periodic Output-muuttujan voi kokeilla laittaa pois päältä, jolloin kuviot eivät toistu lopputuloksessa. Kuvan 6 Symmetry muuttuja tarkoittaa lopputuloksen symmetrisyyttä. Mitä isomman luvun tähän parametriin laittaa, sitä useammin algoritmi pyrkii muuttamaan syötteestä muodostettujen kuvioiden kiertoa. Skriptissä esiintyvälle Foundation parametrille ei löytynyt dokumentaatiota, joten sitä ei tässä opinnäytetyössä muuteta.

Kuvan 6 Iterations parametri muokkaa sitä, kuinka monta kertaa algoritmi pyrkii luomaan lopputuloksen. Tähän suositellaan numeroa 0, jolloin algoritmi pyrkii luomaan lopputuloksen, kunnes onnistuu tai epäonnistuu. Eli toisin sanoen, jos Iterations parametri on 0 ja lopputulosta ei tule näkyviin, on syötteessä jotain vikana. Jos OverlapWFC skriptin viimeinen parametri on Incremental, on päällä, niin OverlapWFC generoi myös silloin, kun pelin käynnistää. Kun asetukset ovat asetettu, niin OverlapWFC mallin voi suorittaa kuvan 6 yläreunassa näkyvästä generate-painikkeesta, jonka jälkeen kentän kuuluisi generoitua.

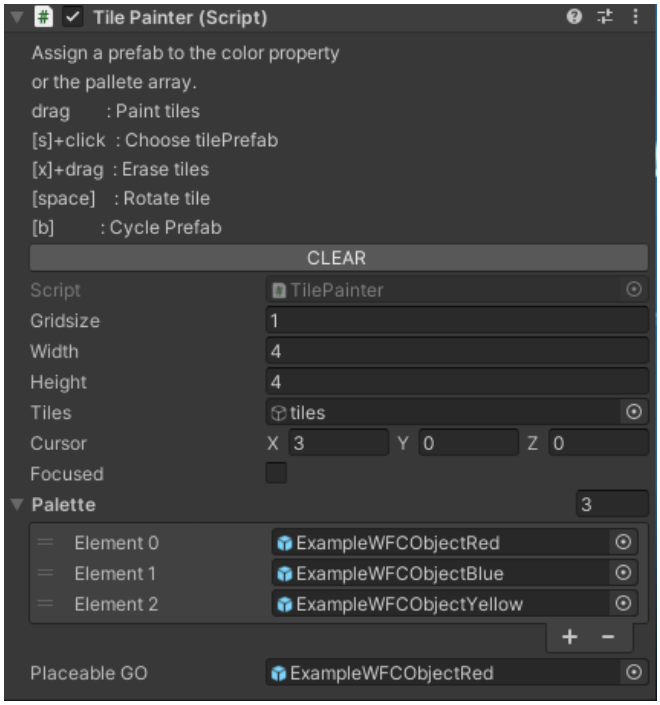


Kuva 6. OverlapWFC-skripti.

3.1.2 Muut olennaiset skriptit

TilePainter.cs on skripti, jolla helpotetaan 3D-malleilla esimerkki syötteen tekoa. TilePainter-skriptin avulla 3D-malleja voidaan “maalata” eli asettaa ruudukkoon pelikehittäjän haluamalla tavalla. TilePainter-skripti toimii hyvin samalla tavalla kuin Unityn oma 2D-tilemap, jossa valmiita 2D-kuvia, jotka ovat suunniteltu peligrafiikoiksi, maalataan ruudukkoon, eli tilemappiin, josta muodostuu pelikenttä. Tässä tapauksessa 3D-malleja ruudukkoon maalaamalla muodostuu esimerkki syöte WFC-algoritmia varten. Esimerkki syötteen voi myös luoda ilman TilePainter-skriptiä, mutta kyseinen skripti nopeuttaa toimintaa huomattavasti.

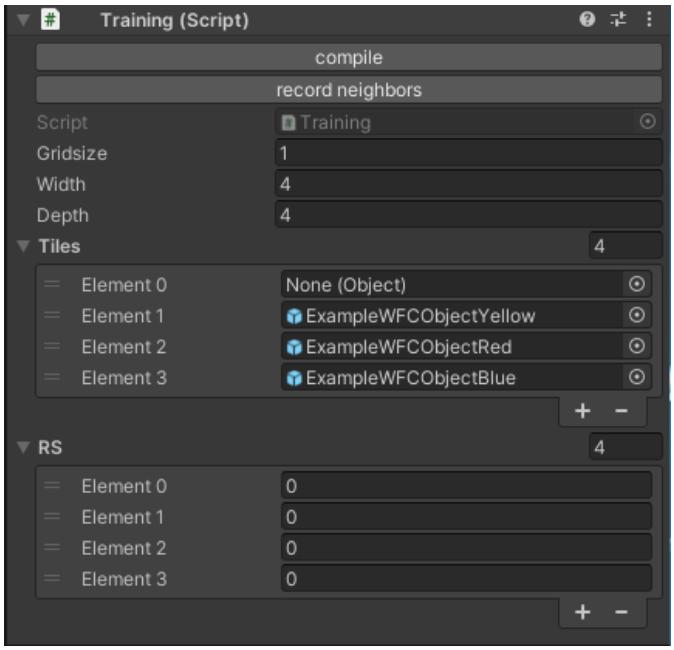
TilePainter-skriptin muuttujat ovat hyvin samanlaiset kuin OverlapWFC skriptissä. Tässä käydään läpi vain olennaiset muuttujat. Gridsize tarkoittaa ruudukkoon maalattavien 3D-mallien kokoa. Width ja Height muuttujat taas muokkaavat ruudukon kokoa. Palette muuttuja on lista, johon laitetaan 3D-mallit, joista halutaan muodostaa esimerkki syöte. Placeable GO tarkoittaa tällä hetkellä valittua 3D-mallia, jota ollaan maalaamassa ruudukkoon. TilePainter-skriptin yläreunassa myös lukee kyseisen skriptin käyttöohjeet ja painikkeet. Näitä voi myös muokata skriptistä, jos ennalta määritetyt painikkeet eivät kelpaa.



Kuva 7. TilePainter-skriptin asetukset.

Training.cs on skripti, joka käsittelee TilePainter-skriptin luoman esimerkki syötteen ja luo sen pohjalta säännöt, joita WFC-algoritmi noudattaa. Training-skriptiä käytetään kummankin WFC-mallin tapauksessa.

Kuvassa 7 nähdään Training skriptin asetukset. Yleensä tämän skriptin muuttujien on hyvä täsmätä TilePainter skriptin muuttujiin. Gridsize on syötteessä esiintyvien 3D-mallien koko ja Width ja Depth ovat syötteen ruudukon koko. Tiles muuttuja on lista, jota ei tarvitse muokata, sillä Training skripti luo tämän sisällön, kun kuvan 7 yläreunassa olevaa Compile-painiketta painetaan. Compile-painike kokoaa syötteeseen maalatuista 3D-malleista tiedot, joita WFC-mallit hyödyntävät. Record Neighbors-painike ja RS eivät ole tämän opinnäytetyön kannalta merkittäviä, sillä niitä käytetään vain, jos SimpleTiledWFC-malli on käytössä.



Kuva 7. Training-skripti.

3.2 Proseduraalisen generoinnin työkalu käytännössä

Tässä osiossa käydään läpi esimerkki WFC-työkalun käytöstä yksinkertaisen prototyypin luomisesta. Oletetaan, että WFC-työkalu on jo lisätty projektiin valmiiksi ja 3D-mallit ovat valmiina. Tavoitteena on luoda kolmesta yksinkertaisesta 3D-mallista kaupunkinäkymä. Tämä on prototyyppinä hyvä, sillä lineaarisissa peleissä säännöllisyys ja johdonmukaisuus on tärkeää. Lisäksi erityisesti lineaarisille peleille olennaista on, että proseduraalisen generoinnin lopputulosta voi

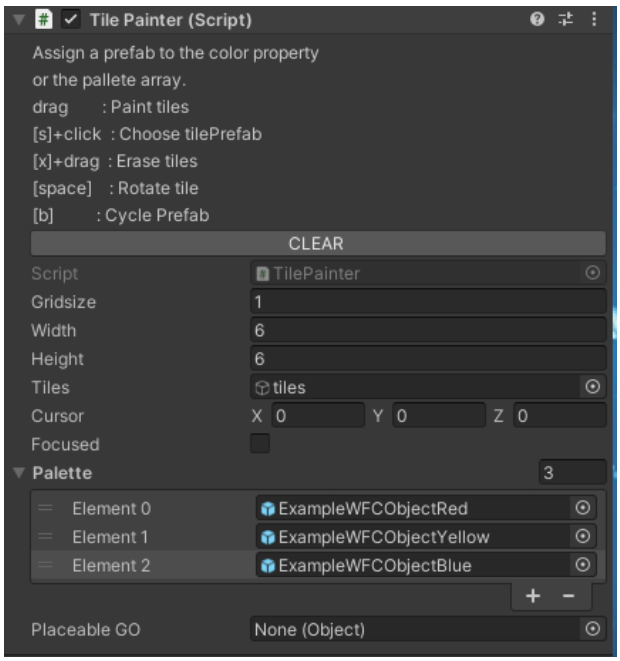
muokata jälkeenpäin, sillä usein halutaan muokata yksityiskohtia, jotka johdattelevat pelaajia oikeisiin kohteisiin.

3.2.1 Algoritmin syötteen valmistelu

WFC-työkalun käyttöönotto alkaa kahden tyhjän peliobjektin luonnilla. Nimetään toinen näistä nimellä “WFC_Input” eli syöte, ja toinen nimellä “WFC_Output” eli ulostulo. Kuvassa 8 näkyy valmiiksi luodut WFC_Input- ja WFC_Output-peliobjektit. WFC-työkalun toiminta rakennetaan näihin kahteen peliobjektiin. Kyseisten peliobjektien nimillä ei ole toiminnan kannalta väliä, mutta ne selkeyttävät toimintaa. Main Camera ja Directional Light ovat valmiiksi luotuina uusissa Unity-sceneissä.

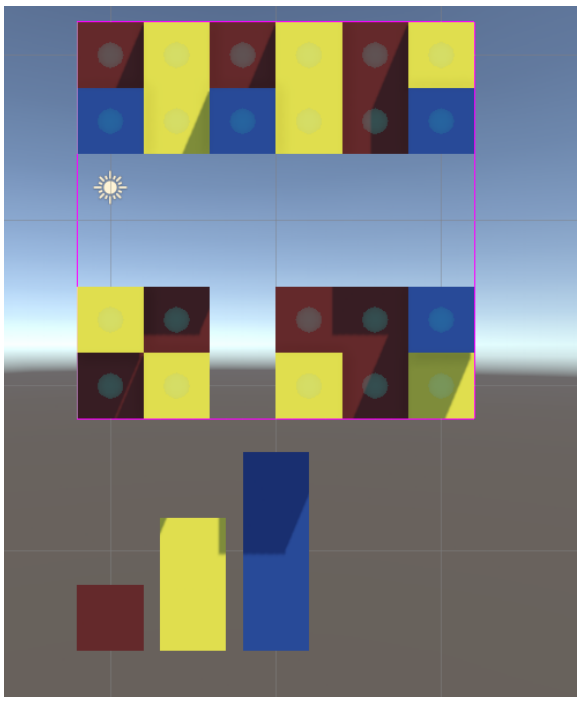
Kuva 8. WFC-työkalun prototyypin ensimmäinen vaihe

Valmistellaan ensiksi työkalun syöte. Ensiksi WFC_Input-peliobjektiin lisätään TilePainter skripti ja painetaan skriptin Clear-painiketta, joka luo Tiles-peliobjektin WFC_Input peliobjektin sisälle. Tiles-peliobjektissa on automaattisesti Training skripti valmiina. Tämän jälkeen muokataan syötteeseen haluttavat muuttujien arvot TilePainter ja Training skripteihin. Kuvasta 9 nähdään kuinka Gridsize muuttujan arvoksi on asetettu yksi eli syötteen 3D-mallien täytyy tässä tapauksessa sopia yhden Unity-unitin sisälle. Width ja Height muuttujiin asetetaan arvo kuusi kuten kuvasta 9 huomataan. Tämän jälkeen lisätään TilePainter skriptin Palette-listaan valmiiksi luodut 3D-mallit, joista halutaan muodostaa esimerkki syöte. Kuvasta 9 nähdään kolme 3D-mallia, jotka on lisätty Palette-listaan. Training skriptiin kopioidaan samat tiedot, jotka laitettiin TilePainter skriptiin.



Kuva 9. TilePainter-skriptin asetukset prototyypissä

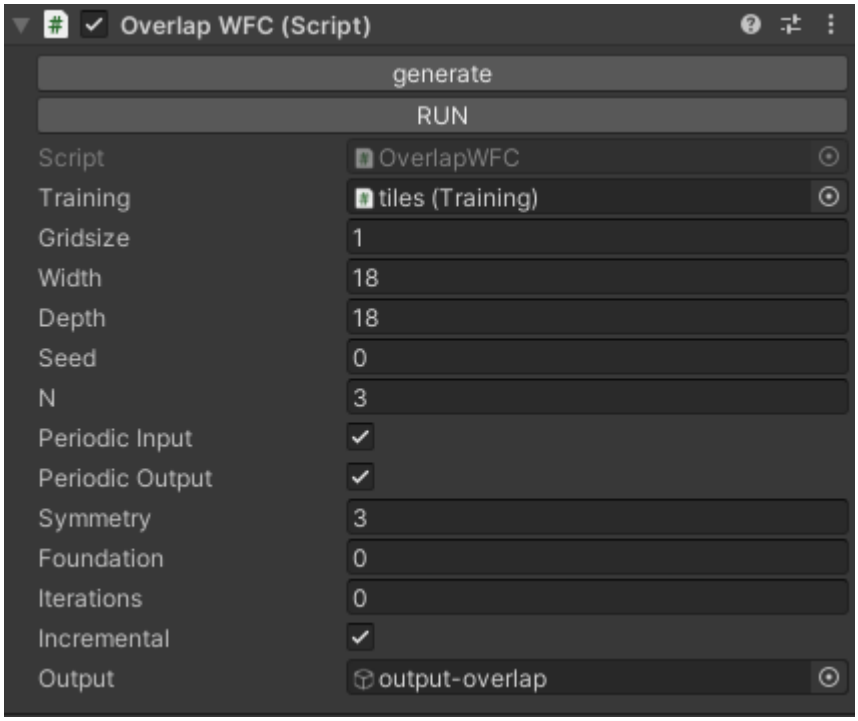
Seuraavana vaiheena prototyypin teossa on esimerkki syötteen maalaaminen. Kuvassa 10 ruudukon alapuolella näkyvät maalattavat 3D-mallit. Punainen kuutio on yhden Unity-unitin kokoinen. Keltainen 3D-malli on yhden Unity-unitin levyinen ja syvyinen, mutta kahden Unity-unitin korkuinen. Sininen 3D-malli taas on kolmen Unity-unitin korkuinen. Kuvasta 10 huomataan, kuinka 3D-mallien korkeuserot eivät näy, sillä ruudukossa 3D-malleja katsotaan ylhäältä päin. Korkeuserot ovat kuitenkin huomattavissa ruudukon alapuolella olevista 3D-malleista. Maalattavista 3D-malleista voi valita haluamansa painamalla S-kirjainta ja klikkaamalla 3D-mallin kohdalta. Tämän jälkeen ruudukkoon voi maalata esimerkki syötteen hiiren vasemmalla painikkeella. Kuvassa 10 näkyy vaaleanpunaisen ruudukon sisällä tässä prototyypissä käytettävä esimerkkisyöte. Tämän jälkeen täytyy painaa Tiles-peliobjektissa olevasta Training-skriptistä Compile-painiketta, joka kokoaa syötteen WFC-mallien käytettäväksi. Tämän jälkeen esimerkki syöte on valmis.



Kuva 10. WFC-työkalun esimerkki syöte.

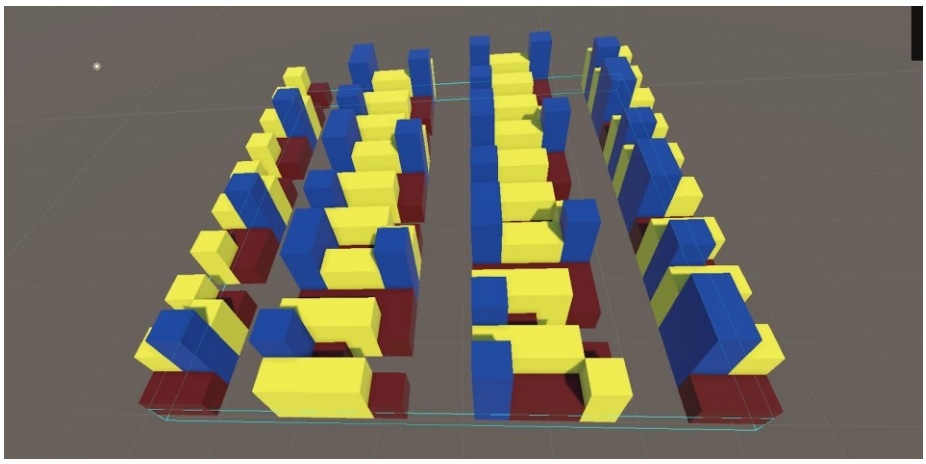
3.2.2 Tulosteen valmistelu ja lopputulos

Seuraavaksi valmistellaan ulostulo eli WFC_Output-peliobjekti valmiiksi. Ensiksi lisätään WFC_Output-peliobjektiin OverlapWFC-skripti. Tämän jälkeen asetetaan OverlapWFC-skriptin asetukset. Training muuttujaan valitaan WFC_Input-peliobjektin sisällä oleva Tiles-peliobjekti. Gridsize täytyy olla sama kuin Training- ja TilePainter-skripteissä. Width ja Depth taas voi olla eri arvoilla kuin esimerkki syötteessä annetut arvot. Asetetaan tässä tapauksessa esimerkki syötteen arvot kolme kertaisena eli arvolla 18. Seed muuttujaan jätetään arvo nolla, joka on myös oletuksena. Muutetaan muuttujan N arvo numeroksi kolme. Periodic Input- ja Periodic Output-muuttujat laitetaan päälle. Symmetry muuttujaan asetetaan arvo 3, sillä 3D-mallit ovat hyvin yksinkertaisia ja niiden kierrolla ei ole väliä, koska ne ovat jokaiselta suunnalta samanlaisia. Tällä saadaan ulostulosta hieman mielenkiintoisempi. Foundation ja Iterations muuttujien arvot jätetään oletusarvoihin ja Incremental muuttuja laitetaan päälle. Kuvasta 11 nähdään OverlapWFC-skriptin lopulliset arvot. Tämän jälkeen asetetaan WFC_Output-peliobjektin kierron X-arvoksi -90, jotta WFC-työkalun lopputulos on kierroltaan oikein 3D-pelimaailmaan nähden. Tämän jälkeen voidaan painaa kuvan 11 yläreunassa näkyvää “generate”-painiketta, jonka jälkeen WFC_Output-peliobjektin alle generoituu lopputulos.

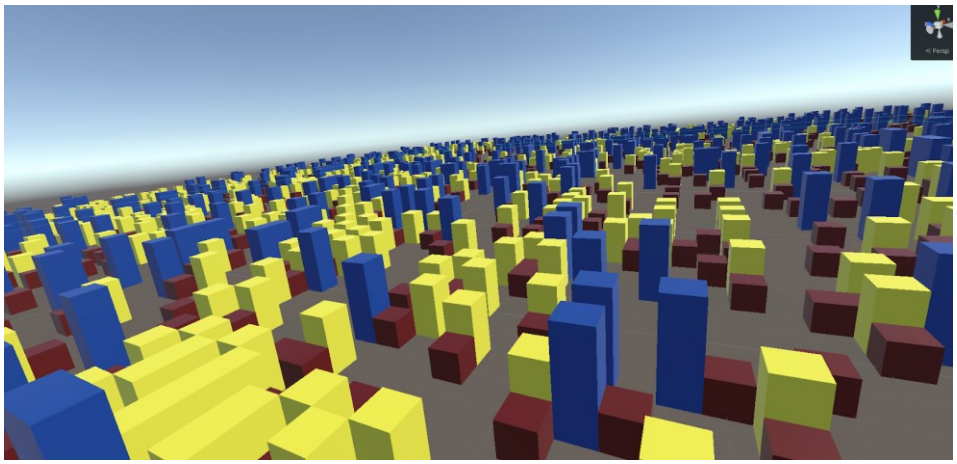


Kuva 11. OverlapWFC-skriptin asetukset prototyypissä.

Kuvasta 12 nähdään WFC-työkalulla esimerkki syötteestä syntynyt lopputulos, joka muistuttaa kaupunkia. Pienessä mittakaavassa algoritmin toiminta ei vaikuta kovin vakuuttavalta, mutta vaikuttavaa on työkalun skaalautuvuus. Kuvassa 13 nähdään esimerkki algoritmin toiminnasta, kun ulostulon koko on lähes kymmenenkertainen aiempaan. Pelikehittäjä voisi luoda pienen prototyypin kymmenessä minuutissa, mutta aikaa menisi huomattavasti pidempään suuremman kaupungin luomisessa käsin. Lopputulos olisi tietenkin parempi, jos 3D-mallit olisivat täydellisesti yhteensopivia ja variaatioita olisi enemmän. WFC-työkalu voi olla kenttien kehittäjälle hyvä prototyyppi työkalu, sillä generoinnin jälkeen jokaista mallia voi vielä muokata tarvittaessa eli generoinnin lopputulos voi toimia vain alkupisteenäkin.



Kuva 12. Prototyypin lopputulos.



Kuva 13. Algoritmin lopputulos isommassa mittakaavassa.

4 YHTEENVETO JA POHDINTA

Tämän opinnäytetyön tarkoituksena oli selvittää, miten proseduraalista generointia voisi hyödyntää lineaaristen pelien kenttien luomisessa. WFC-työkalun lopputulos on lineaaristen pelien kannalta hyvä, sillä se skaalautuu hyvin, ja sen lopputulosta voi muokata generoinnin jälkeen. WFC-työkalun OverlapWFC-malli voi myös hyödyntää valmiita 3D-malleja, jotka ovat mahdollisesti luotu jo valmiiksi peliä varten.

Tässä opinnäytetyössä huomattiin, että WaveFunctionCollapse-algoritmin toteuttava työkalu on helppo ja nopea ottaa käyttöön Unity-projektissa. Työkalun kannalta hankalaksi osuudeksi voisikin osoittautua sellaisten 3D-mallien luominen, jotka sopivat algoritmin toimintaan. Lisäksi algoritmin tai työkalun toimintaa ei tässä opinnäytetyössä muokattu, mikä olisi parhaan lopputuloksen saavuttamiseksi tärkeää, mutta voisi myös olla haastavaa. Tässä opinnäytetyössä tutustuttiin myös muihin proseduraalisen generoinnin työkaluihin. Parhaan kokonaisuuden proseduraalisella generoinnilla saisi yhdistämällä kahta tai useampaa proseduraalisen generoinnin algoritmia, sillä eri algoritmit erikoistuvat eri tarkoituksiin. Myös SimpleTiledWFC mallin käyttäminen ja sitä varten 3D-mallien luominen voisi auttaa kenttien kehittäjää.

Heikkouksena proseduraalisen generoinnin käyttämisessä voi olla toistuvuus, jolloin kenttä voi näyttää yksitoikkoiselta. Tässä tapauksessa WFC onkin hyvä, sillä pelin kenttää voi myös muokata generoinnin jälkeen manuaalisesti. Kuitenkin proseduraalisen generoinnin tarkoitus on välttää manuaalinen työ, jolloin sen hyöty kärsii hieman.

LÄHTEET

Black, P. 2005. Dictionary of Algoritms and Data Structures: Hakusana greedy algorithm. Sanakirja. Viitattu 16.01.2023. https://www.nist.gov/dads/HTML/greedyalgo.html

Brummelen, J & Chen, B. n.d. Procedural generation: creating 3D worlds with deep learning. Verkkosivu. Viitattu 19.12.2022 https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html

Gumin, M. 2022. WaveFunctionCollapse. Verkkosivu. Viitattu 16.01.2023. https://github.com/mxgmn/WaveFunctionCollapse

Karth, I. 2017. WaveFunctionCollapse is Contraint Solving in the Wild. Artikkeli. Viitattu 16.01.2023. https://isaackarth.com/papers/wfc_is_constraint_solving_in_the_wild.pdf

KhanAcademy. n.d. https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise

Merrell, P. 2021. Comparing Model Synthesis and Wave Function Collapse. Artikkeli. Viitattu 01.02.2023. https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf

Merrell, P. 2023. Model Synthesis. Verkkosivu. Viitattu 01.02.2023. https://paulmerrell.org/model-synthesis/

McCombs, S. n.d. Intro To Procedural Textures. Blogi. Viitattu 21.12.2022. http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

Perlin Noise. Wikipedia. Viitattu 24.01.2023. https://en.wikipedia.org/wiki/Perlin_noise

Sarawagi, S. n.d. Procedural Generation – A Comprehensive Guide Put in Simple Words. Blogi .Viitattu 19.12.2022. https://scaleyourapp.com/procedural-generation-a-comprehensive-guide-in-simple-words/

Techopedia. 2018. Procedure. Verkkosivu. Viitattu 19.12.2022. https://www.techopedia.com/definition/3727/procedure

Wikidot. 2018. Cellular Automata. Verkkosivu. Viitattu 10.01.2022. http://pcg.wikidot.com/pcg-algorithm:cellular-automata

3D-Mapper. n.d. Verkkosivu. Viitattu 01.02.2023. https://3d-mapper.com/height-maps-and-textures/