Tampereen yliopisto

Jesse Liljenmaa

AUTOMAATTINEN SISÄLLÖNTUOTANTO VIDEOPELEISSÄ

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Toukokuu 2021

TIIVISTELMÄ

Jesse Liljenmaa: Automaattinen sisällöntuotanto videopeleissä
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2021

Pelien suunnittelemiseen ja tuotantoon kuluu huomattava määrä resursseja, joskus niin paljon, etteivät pienet yritykset pienine budjetteineen pysty työllistämään pelikehittäjistä erillisiä pelisuunnittelijoita. Tämän ongelman tuomia hankaluuksia helpottaa *automaattinen sisällöntuotanto* (*Procedural Content Generation*, PCG). PCG:n avulla pystytään algoritmisesti luomaan kenttiä, esineitä, tapahtumia, hahmoja, pulmia, tehtäviä ja muuta pelisisältöä, joko auttaen pelisuunnittelijaa suunnittelemaan sisältöä pelikehityksessä tai luoden sisältöä pelaajan pelatessa peliä. Tunnetuinpana esimerkkinä automaattisesta sisällöntuotannosta kaupallisten videopelien kontekstissa voidaan pitää maailman myydyintä videopeliä Minecraftia (Mojang Studios, 2011), joka automaattisia sisällöntuotantotekniikoita käyttäen luo jokaiselle pelaajalle uniikin maailman ennen varsinaisen pelaamisen aloittamista.

Tässä tutkielmassa käsitellään automaattisen sisällöntuotannon suosituimpia tekniikoita, joista annetaan teorian lisäksi toteutusesimerkkejä. Tutkielmassa käydään pohjustus- ja analysointitarkoituksessa läpi myös automaattisen sisällöntuotannon motiiveja sekä vertaillaan sen ja pelisuunnittelijan luoman sisällön välisiä eroja sekä niiden hyviä ja huonoja puolia.

Automaattisesta sisällöntuotannosta on tehty runsaasti tutkimuksia. Laajin teos on Nelson ynnä muiden vuonna 2016 kirjoittama kirja, joka sisältää aikalaisteknologian kuvausten lisäksi automaattisen sisällöntuotannon taksonomian, jota käytetään tämän tutkielman rakenteen muodostajana. Tuoreimpien tekniikoiden käsittelyn jälkeen vastataan työn tutkimuskysymykseen: onko taksonomia pätevä lähtökohtana kuvaamaan nykyaikaisia tekniikoita? Kysymyksen suhteen tutkimus päätyy positiiviseen vastaukseen.

Tutkielma toteaa automaattisen sisällöntuotannon olevan vielä hyvin teoreettinen tutkimusala: poislukien tavanomaiset tekniikat suurinta osaa tekniikoista ei ole käytetty kaupallisten pelien kontekstissa pääosin tekniikoiden käyttöönottoon liittyvien vaikeuksien takia. Viimeaikaisten pelitrendien nojalla on kuitenkin olemassa mahdollisuus, että videopelikehittäjät tarttuvat uusiin tekniikoihin ja alkavat hyödyntää niitä.

Avainsanat: procedural content generation, video games, automaattinen sisällöntuotanto, videopelit

Sisällysluettelo

1 Johdanto

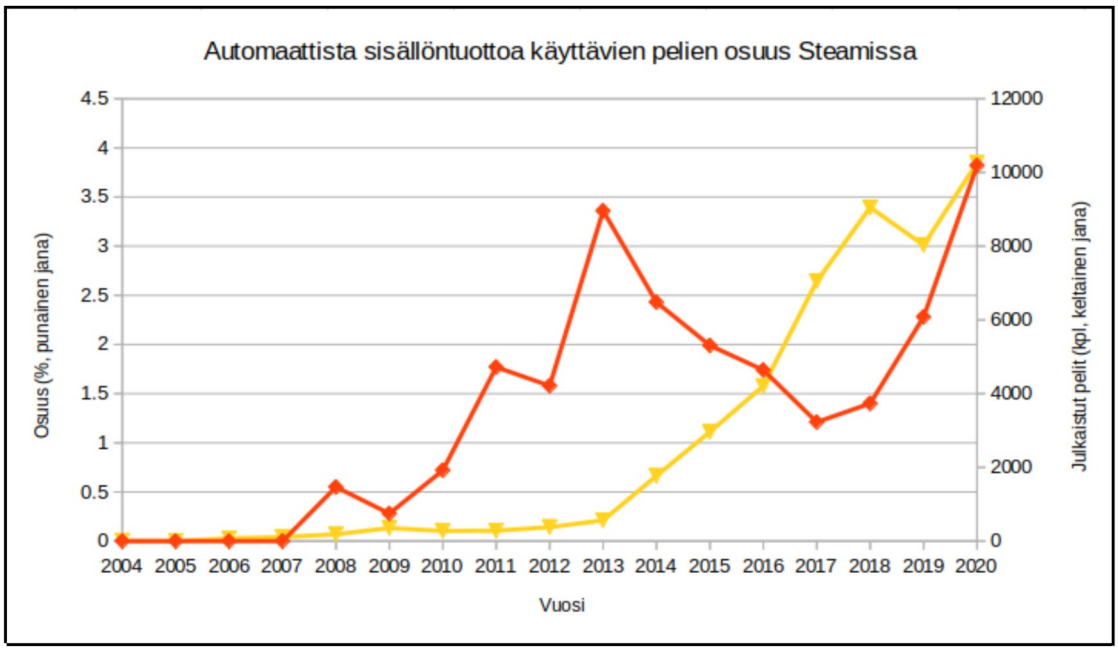
Perinteisesti videopelien sisällön suunnittelee pelisuunnittelija. Suuremmissa peliyrityksissä on palkkalistoilla suunnittelijoita, joiden tehtävänä on suunnitella ja rakentaa pelin pelimaailma, grafiikat, kaksi- tai kolmiulotteiset mallit, tarina ja muu luovuutta vaativa sisältö niin, että pelikokemus kohtaa pelaajan miellyttävällä tavalla. Tämä ei kuitenkaan ole läheskään aina taloudellisesti mahdollista pienille peliyrityksille tai yksittäisille henkilöille, sillä mainitut elementit vaativat työtuntien käyttämistä niiden suunnitteluun varsinaisen pelin ohjelmoinnin yhteydessä (Bevilacqua et al., 2011, Hendrikx et al., 2013). Yhtenä tätä työtä helpottavana mahdollisuutena on käyttää hyväksi *automaattista sisällöntuotantoa* (*Procedural Content Generation, PCG*), joka on vaihtoehtoinen tapa tuottaa sisältöä videopeleihin: pelisisältö pystytään luomaan algoritmisesti, mahdollisesti ohittaen pelisuunnittelijan tarpeen. Tarkemmin kuvailtuna automaattisen sisällöntuotannon avulla pystytään luomaan tietokoneavusteisesti kenttiä, esineitä, tapahtumia, hahmoja, pulmia, tehtäviä, tekstuureja ja muuta pelisisältöä (Browne et al., 2010).

Automaattisen sisällöntuotannon historia on pitkä: sitä on käytetty ainakin vuodesta 1978 alkaen, kun Atari 2600 -konsolille julkaistu sokkelopeli Maze Craze loi jokaiselle pelikerralle uuden sokkelon pelaajan ratkaistavaksi. Tähän aikaan automaattisen sisällöntuotannon painopiste pohjautuikin satunnaisten numeroiden avulla luotuun pelisisältöön, eikä siten sisältänyt mitään erityisempää teknologiaa (Antonova, 2015). Pelillisten aspektiensa lisäksi PCG:tä alettiin 1980-luvulta alkaen käyttämään suurta tilaa vaativien pelielementtien (kuten pelimaailmojen) tiivistämiseen. Ajan kuluessa ja tietokoneiden muistikapasiteetin kasvaessa tämä tarve poistui (Browne et al., 2010).

Nykypäivän tyypilliseen PCG:n käyttöön kuuluu vieläkin erityisesti satunnaisten pelikenttien luonti (Arnaboldi & Loiacono, 2019). Tämä mahdollistaa erittäin suuren uudelleenpelattavuuden videopelille, sillä tämänkaltaista ei-determinististä pelisisältöä pystytään luomaan loputon määrä (Browne et al., 2010). Yhä enenevässä määrin PCG:tä on alettu käyttämään myös pelikehityksen aikana pelimaailman luonnin avustajana (Arnaboldi & Loiacono, 2019; Bevilacqua et al., 2011). Erityinen PCG-tekniikkojen hyöty verrattuna pelisuunnittelijaan on sen mahdollisuus luoda pelisisältöä, jota pelisuunnittelija ei välttämättä koskaan ajattelisi tehdä (Antonova, 2015; Browne et al., 2010). PCG:ssä on keskeisinä ongelmina se, miten luodaan mielekästä, luovaa ja aitoa pelisisältöä ja miten tällaista pelisisältöä luovia algoritmeja luodaan (Mateas & Smith, 2011; Nelson et al., 2016).

Seuraavaksi havainnoillistetaan peliensisäisen PCG:n suosiota kaupallisessa kontekstissa tätä tutkielmaa varten luodun suppean pelimarkkinatutkimuksen avulla. Kaavioon 1 on koottu maailman suurimman PC-pelien myynti- ja jakamisalusta Steamin (Valve Corporation) tarjoamia tilastoja PCG-tekniikan käytöstä Steamissa julkaistujen pelien osalta palvelun koko olemassaolon ajalta. Kaavio on muodostettu Steam Searchin avulla suodattamalla pelejä tunnisteella "Procedural Generation" ja lajittelemalla löytyvät videopelit julkaisuvuoden perusteella. Keltainen kuvaaja kuvaa vuosittaista Steamissa julkaistujen pelien lukumäärää. Punainen kuvaaja kuvaa sitä, kuinka suuri osa julkaistuista peleistä käyttää PCG-tekniikoita.

Kaaviosta on havaittavissa kolme trendiä. Ensinnäkin vuodesta 2008 vuoteen 2013 PCG-tekniikoiden käyttö on ollut kasvussa suhteessa vuosittain julkaistujen pelien määrään, huipentuen Minecraftin (Mojang Studio, 2011) julkaisuun, jolla on ollut mitä todennäköisimmin vaikutusta PCG-tekniikoiden suosioon maailman myydyimpänä videopelinä. Minecraft käyttää kohinakarttoja luodessaan maailmansa annetun siemenluvun perusteella. Toiseksi vuodesta 2013 vuoteen 2018 PCG-tekniikoita käyttävät pelit ovat musertuneet valtaisan pelijulkaisuvirran alle, joten aikaisemmin saavutettua pelituotantotahtia ei olla pystytty pitämään yllä. Viimeisenä huomiona viime vuonna PCG:tä käyttävien pelien osuus on noussut korkeammalle kuin koskaan aikaisemmin.



Kaavio 1. Steamin PCG:tä käyttävien pelien osuus kaikista tiettynä vuonna julkaistuista videopeleistä. Todellinen prosenttimäärä on suurempi, sillä tunnisteita lisätään pääosin käyttäjäkunnan voimin. Tutkimusta varten muodostettu kaavio.

Tässä tutkielmassa analysoidaan automaattisen sisällöntuotannon tutkimusaihepiiriä kirjallisuuskatsauksen muodossa. Tarkemmin kuvailtuna tavoitteena on selvittää, *minkälaista sisältöä* automaattisella sisällöntuotannolla pystytään luomaan ja *millä tavalla* tämä pystytään toteuttamaan. Kysymyksiin vastataan esittelemällä sekä PCG:n avulla luotua sisältöä että sitä luovaa tekniikkaa. Nykyisten tekniikoiden pohjustamista varten käydään tapauskohtaisesti läpi myös vanhempia tekniikoita ja niiden historiallisia tavoitteita. Tutkielmassa pohditaan myös, miksi tiettyjä tekniikoita suositaan toisten sijasta. Tekniikoiden lukumäärän vuoksi kaikenkattavaa analyysiä aiheesta ei pystytä tekemään: tämän sijaan ainoastaan kaikkein suosituimmat tekniikat käydään lävitse.

Tässä tutkielmassa hyödynnetään viitteellisesti Togeliuksen tuottamaa taksonomiaa (Nelson et al., 2016), jossa on luokiteltu erinäisiä automaattisen sisällöntuotannon tekniikoiden tunnusmerkkejä. On kuitenkin huomattava, ettei taksonomiaa ole päivitetty viimeiseen viiteen vuoteen lainkaan. Tämä tutkielma toteaa, että viime vuoden uudet tekniikat ovat kuitenkin hyvin mahdutettavissa taksonomian raameihin, eikä taksonomia siksi tarvitse uusia luokkia.

Tämän tutkielman tekoon on haettu materiaalia alan tietokannoista hakulauseella “Procedural Content Generation” AND “Video Games”, sillä on erittäin todennäköistä, että aihetta käsitellyt tieteellinen julkaisu sisällyttäisi avainsanoinaan vähintään nämä kaksi. Pääasiallisena tietokantana on käytetty Association for Computing Machineryn tietokantaa, joka sisältää suurimman osan automaattisesta sisällöntuotannosta kertovista tieteellisistä julkaisuista. Lisäksi Springerin, IEEE:n, ja erinäisten yliopistojen tietokannoista on löytynyt tieteellisiä julkaisuja samoilla käsitteillä. Tähän tutkielmaan on sisällytetty julkaisuista ne, jotka joko kuuluvat alan perustavanlaatuisiin teoksiin tai kuvailevat tiettyä PCG-arkkitehtuuria tutkielmaan sopivalla tavalla.

Suomessa automaattiseen sisällöntuotantoon liittyen on tehty yksittäisiä tutkimuksia, mutta tässä tutkielmassa esitettyä aiheen läpikäyntiä ei ole suoritettu aiemmin. Haavisto (2015) ja Reunanen (2017) käsittelevät pelikenttien automaattista luomista. Hannula (2020) on luonut simuloidun eläinmaailman, jonka sisältämät eläimet kehittyvät geneettisesti. Liikkanen (2015) tutkii erilaisia suosittuja proseduraalisen sisällöntuotannon keinoja, ja tuottaa tutkimuksen yhteydessä kartanluontityökalun.

Seuraavaksi esitellään tutkimuksen rakenne. Luvussa 2 pohjustetaan PCG-tekniikoita esittämällä niiden tavanomaisia käyttökohteita. Tämä tapahtuu tulkiten Bevilacqua ynnä muiden ja Hendrikx ynnä muiden tuottamia tutkimuksia aiheesta. Luvussa 3 esitellään lyhyesti Togeliuksen taksonominen luokittelu PCG-tekniikoista, sekä tieteellisiä julkaisuja, joita käsitellään taksomian avulla. Luvussa 4 esitellään Togeliuksen taksonomian jälkeen julkaistuja tieteellisiä papereita ja näihin liittyviä PCG-tekniikoita ja videopelejä. Lopuksi luvussa 5 esitetään yhteenveto ja

johtopäätöksiä, selvennetään, miksi taksonomia ei tarvitse päivitystä, sekä pohditaan tulevaisuuden kehitystä.

2 Tyypillinen oikean elämän PCG:n käyttö

Tässä luvussa esitellään tyypillisiä, oikeassa elämässä käytettyjen PCG-tekniikoiden käyttökohteita, jotka on jaettu kahteen erikseen käsiteltävään osaan. Ensimmäinen osa käsittelee pelikehityksen aikaista PCG:tä, kun taas toinen videopelien aikana tapahtuvaa PCG:tä. Kummassakin osassa on esitelty yksi aihetta käsitellyt tutkimus. Kyseisten tutkimusten listaamat tekniikat käydään lävitse omissa luvuissaan.

2.1 Pelikehityksen aikainen PCG:n käyttö

Miten automaattista sisällöntuotantoa hyödynnetään pelikehityksessä? Voi olla haastavaa määrittää, minkälainen PCG:n käyttö on ollut suosituinta viime vuosina, sillä aiheesta ei olla tehty tutkimuksia moneen vuoteen. Vuodelta 2011 löytyy kuitenkin yksi tutkimus, joka käsittelee aihetta (Bevilacqua et al., 2011). Tutkimuksen metatekstistä pystytään tulkitsemaan, kuinka PCG oli vasta tuloillaan oleva pelikehityksen aikainen sisällöntuotantometodi. Tästä huolimatta tutkimuksen esittelemät keinot ovat vieläkin erittäin suuressa käytössä.

Bevilacqua ynnä muut ovat tutkineet erilaisia pelikehityksessä käytettyjä metodeja luoda luonnollisen näköistä oheisympäristöä eli ympäristöä, jota ei ole tarkoitettu tutkittavaksi. Tähän kuuluu esimerkiksi kaukaisuudessa näkyvät vuoret, joet, metsät ja asutus. Tutkimuksen mainitsemia tekniikoita luonnollisen oheisympäristön luomiseen ovat kohinakartat (ks. Luku 3.2.4), hakupohjaiset algoritmit (ks. Luku 3.2.2), ja parametrisoidut algoritmit (esim. *Answer Set Programming*; ks. Luku 3.2.3). Teiden ja kaupunkien automaattiseen tuotantoon käytetään graafiteoriaa ja L-systeemejä, joista L-systeemejä käytetään myös kasvillisuuden luonnissa (ks. Luku 3.2.1). Kaikissa edellämainituissa tekniikoissa käytettiin myös näennäissatunnaisnumeroita, jotta luomisprosessi on probabilistinen eikä siten luo samaa sisältöä uudestaan toisella kerralla (ks. Luku 3.2.5). Jokaista tutkimuksen mainitsemaa tekniikkaa käytetään vielä näinä päivinä: tekniikkoihin perehdytään tarkemmin, kun yksittäisiä tekniikoita käsitellään omissa luvuissaan.

2.2 Suorituksenaikainen PCG:n käyttö

Miten tavanomainen julkaistu peli käyttää PCG:tä? Kuten edellisessäkin luvussa, aiheesta ei olla tehty tutkimusta vuosiin. Hendrikx ynnä muut ovat kuitenkin tehneet tutkimuksen, jossa paljastuu tyypillinen PCG:n käyttö vuodelta 2013. Taulukossa 1 on kuvattuna tutkimuksen tulokset.

Tutkijat ovat jaotelleet peleissä PCG-tekniikaa hyödyntävät osat kuuteen kategoriaan:

1. *Game Bits* (yksittäisiä osia, jotka eivät esiinny itsenään, kuten tekstuureja);
2. *Game Space* (kokonaisuuksia Game Biteistä, kuten pelikenttiä);
3. *Game Systems* (kokonaisuuksia Game Spaceista, kuten virtuaalista asutusta);
4. *Game Scenarios* (pelin esittämiä tilanteita, kuten tarinoita tai pulmia);
5. *Game Designs* (pelin sääntöjä); ja
6. *Derived Content* (pelaajien tekojen takia kumpuavia tarinoita, kuten pelinsisäisiä sanomalehtiartikkeleita).

Table I. Use of PCG-G Techniques in Games

Games (with year of release)	Game Bits	Game space	Game Systems	Game Scenarios
Borderlands (2009)	x			
Diablo I (2000)		x		
Diablo II (2008)		x		x
Dwarf Fortress (2006)		x	x	x
Elder Scrolls IV: Oblivion (2007)	x			
Elder Scrolls V: Skyrim (2011)				x
Elite (1984)		x	x	x
EVE Online (2003)	x	x		x
Facade (2005)				x
FreeCiv and Civilization IV (2004)		x		
Fuel (2009)		x		
Gears of War 2 (2008)	x			
Left4Dead (2008)				x
.kkrieger (2004)	x			
Minecraft (2009)		x	x	
Noctis (2002)		x		
RoboBlitz (2006)	x			
Realm of the Mad God (2010)	x			
Rogue (1980)		x		x
Spelunky (2008)	x	x		x
Spore (2008)	x	x		
Torchlight (2009)		x		
X-Com: UFO Defense (1994)		x		

We did not find commercial games that generate procedurally Game Designs or Derived Content.

Taulukko 1. Suorituksenaikaisia PCG-tekniikoita käyttäviä pelejä kategorisoituna kuuteen eri luokkaan (Hendrikx et al., 2013).

Kuten taulukosta käy ilmi, pelin loogisia sääntöjä tai kumpuvia tarinoita generoivia pelejä ei löytynyt: varsinkin loogisten sääntöjen osalta on vaikea tehdä mielekästä peliä. Suurin osa automaattisesti luoduista osista keskittyy pääosin pieniskaalaisiin osasiin ja näistä koostuviin kokonaisuuksiin, sekä tarinoihin ja pulmiin.

Jotta PCG ja sen käyttö konkretisoituisi lukijalle, selvennetään auki muutaman listassa esiintyvän pelin PCG-tekniikoiden käyttö. Dwarf Fortress (Bay 12 Games, 2004) luo kokonaisen fantasiamaailman (em. kategorian kohdat 2. ja 3.) ja sen sisällä tapahtuvia valtataisteluita ja hahmojen henkilökohtaisia suhteita ja saavutuksia (4.). Elite (Acornsoft, 1984) generoi tähtijärjestelmiä (2. ja 3.) sekä niiden sisäistä ekonomiaa. Facade (Procedural Arts, 2005) on pelitutkijoiden kehittämä interaktiivinen tarinapeli, jossa pelaaja yrittää pelastaa (tai pilata) pelihahmopariskunnan avioliiton

puhumalla heidän kanssaan (4.). Usein tieteellisissä julkaisussa esimerkkinä käytetty tasohyppelypeli Spelunky (Derek Yu, 2008) luo käsintehdyistä luolapalasista (1.) luolaston (2.) yhdistämällä näitä palasia loogisesti keskenään. Räiskintäpelisarja Borderlands (Gearbox Software, 2009) luo pelaajien käyttämät aseet satunnaistamalla niiden ominaisuuksia (1.).

3 Togeliuksen taksonomia sekä vastaavat tieteelliset paperit

Tässä luvussa esitellään lyhyesti Togeliuksen vuonna 2010 luoman ja vuonna 2016 päivittämän taksonomian automaattisen sisällöntuotannon tekniikoista (Browne et al., 2010; Nelson et al., 2016). Alan taustatiedon esittelyn lisäksi taksonomiaa käytetään tulevissa luvuissa hyväksi erilaisten automaattisen sisällöntuotannon tekniikoiden esittelyssä.

3.1 Togeliuksen taksonomia

Tässä aliluvussa esitellään Togeliuksen taksonomia. Alkuperäinen taksonomia on käännetty heuristisiksi kysymyksiksi, sillä tämä tiivistää sanoman mahdollisimman hyvin.

1. Onko sisältö luotu pelikehityksen vai pelin pelaamisen aikana? Pelisisällön luonti voidaan suorittaa ennen pelin käynnistämistä tai pelin aikana, kuten aikaisemmassa luvussa on kuvailtu. Pelikehityksen aikana käytettyjä PCG-tekniikkoja käsitellään luvussa 3.2.1.

2. Onko luotu sisältö tarpeellista vai vapaaehtoista sisältöä? Automaattisesti luotu pelisisältö pystytään sijoittamaan pelissä niin, ettei se ole pelin läpäisyyn vaadittavaa. Tämä mahdollistaa sen, että automaattisesti luotu pelisisältö voi sisältää laadullisesti kehnoja artefakteja, jotka pelaaja voi jättää huomioimatta. Tässä tutkielmassa ei palata tähän kriteeriin, sillä vastauspiiri ei kuulu tutkielman rajojen sisälle.

3. Kuinka tarkasti sisällöntuotantoa voidaan hallita? Automaattisen pelisisällön luomisprosessiin pystytään joissain tapauksissa vaikuttamaan: käyttäjä tai kehittäjä pystyy tietyissä tekniikoissa vaikuttamaan luotuun sisältöön, kun taas toisissa tietokone suorittaa luonnin autonomisesti. Tätä aspektia käsitellään tekniikoiden ohessa.

4. Huomioiko sisällöntuotanto pelaajan käyttäytymistä? Sisällöntuotanto voi ottaa huomioon pelaajan käyttäytymisen pelisession aikana joko helpottaakseen, vaikeuttaakseen tai personoidakseen pelikokemusta. Tämä mahdollisuus mainitaan, jos tekniikka käyttää sitä.

5. Käyttääkö sisällöntuotanto satunnaisuutta? Sisällöntuotannon voi tehdä ilman satunnaisuuden läsnäoloa. Tällöin motiivina voi olla tilansäästäminen tai kustannustehokkuus pelikehityksen aikana. Satunnaisuutta PCG-tekniikoissa käsitellään luvussa 3.2.5.

6. Valmistuuko sisältö kerralla vai hiljalleen? Sisällöntuotanto pystytään tekemään osissa, jolloin sisällön päivityksien välissä sisältöä pisteytetään. Tätä kerrataan, kunnes sisältö saa riittävästi pisteitä. Tähän periaatteeseen sopivia tekniikoita käsitellään luvussa 3.2.2.

7. Onko ihmisellä sananvaltaa sisällönluonnissa? Automaattisen sisällöntuotannon ei tarvitse vallata koko luomisprosessia itselleen: monissa tekniikoissa automaattista sisällöntuotantoa käytetään ainoastaan kehittäjän työkaluna, esimerkiksi täyttämään osittain suunnitellun pelikentän puuttuvat kohdat automaattisesti. Näitä käsitellään luvussa 3.2.1.

3.2 Taksonomian aikaisia toteutustekniikoita

Tässä luvussa esitellään Togeliuksen taksonomian aikaisia eli ennen vuotta 2017 käytettyjä automaattisen sisällöntuotannon tekniikoita. Luokittelu perustuu pääosin taksonomian kysymyksiin, mutta joistain tekniikoista on luotu omat aliluvunsa aiheiden käsittelyn helpottamisen vuoksi.

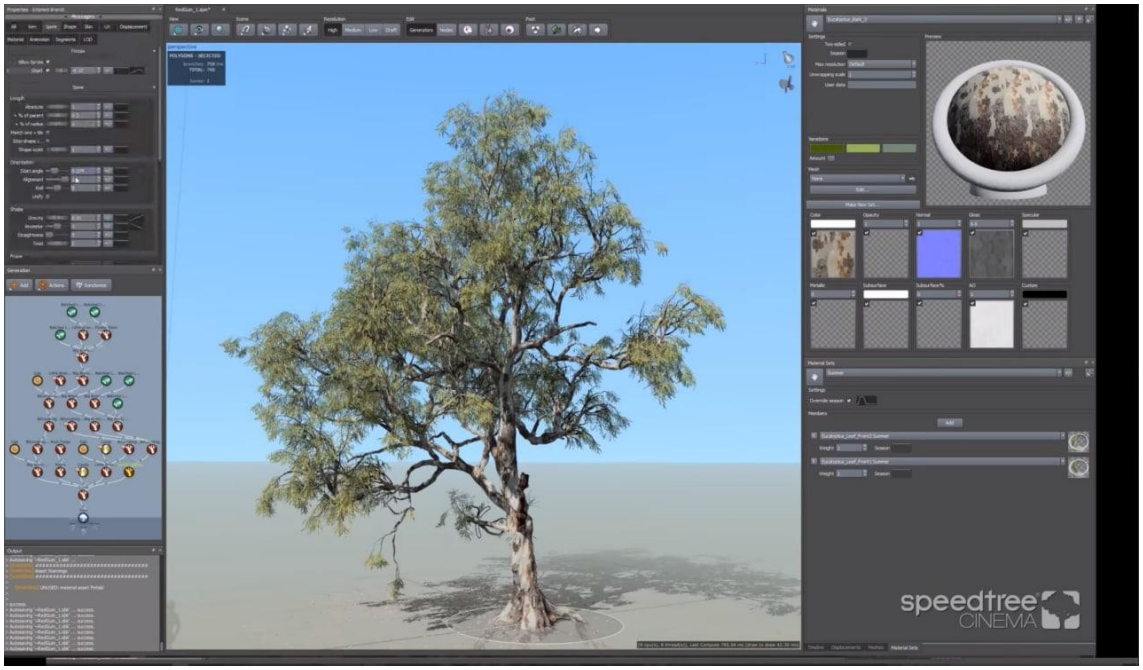
3.2.1 Pelikehityksen aikana käytettyjä PCG-tekniikoita

Edellä esitetyn taksonomian ensimmäinen kysymys erottelee pelikehityksen aikana tapahtuvan sisällöntuotannon pelaamisen aikana tapahtuvasta sisällöntuotannosta. Tässä aliluvussa käsitellään PCG-tekniikoita, joita käytetään pääosin pelikehityksen aikana.

Yleensä pelin juonenkehityksessä käytetään graafipohjaista käyttöliittymää, joka helpottaa pelisuunnittelijan työtä varmentamalla juonen integriteetin esimerkiksi niin, ettei pelaaja joudu alueelle, josta ei pääse ulos (Nelson et al., 2016). Yksi tuoreimmista aiheeseen liittyvistä tutkimuksista on Grabska-Gradzińska ynnä muiden vuonna 2021 tuottama modernien pelistudioiden juonitekniikkojen katsaus, joka samalla esittelee automaattisen juonentäyttökonseptin. Tämän sovelluksen tehtävänä olisi täyttää tai lisätä pelisuunnittelijoiden aloittama juoni joko osittain tai täydellisesti. Vaikka konseptia on käsitelty teoreettisesta näkökulmasta jo aiemmin (ks. Nelson et al., 2016), ei tutkimuksen mainitsemaa juonenluontityökalua (pl. teoreettiset luonnokset) ole vielä luotu tosiasialliseen käyttöön (Grabska-Gradzińska et al., 2021). Työkalun luomisen

vaikeuksiin kuuluu eritoten kriteeri juonen elävyydestä, sillä tähän mennessä luodut työkalut osaavat luoda vain hyvin yksinkertaisia juonia, jotka keskittyvät pelisuunnittelijan luomien esineiden ja pelikenttien interaktioon (Nelson et al., 2016; ks. kuitenkin luku 4.1 tästä tutkielmasta). Tutkijat kuitenkin mainitsevat, että pätevän juonenluontityökalun kehittämistä tutkitaan, mahdollisesti evolutiivisen algoritmin voimaannuttamana (Grabska-Gradzińska et al. 2021).

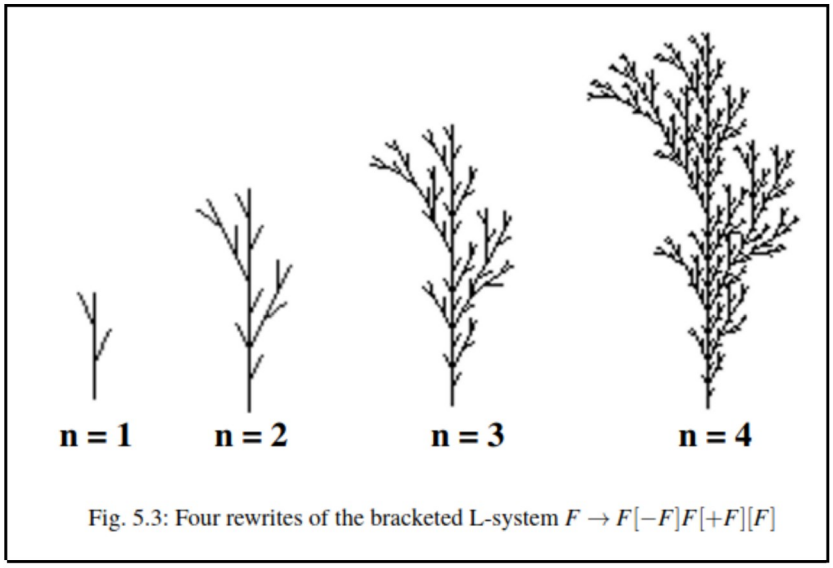
Nykyaikana yksi tunnetuimmista ja käytetyimmistä työkaluista on kasvillisuudenluontisovellussarja SpeedTree (Interactive Data Visualization, 2015). Sovelluksen avulla pelikehittäjä kykenee luomaan puita, puskia, ruohoa ja muuta halutunlaista kasvillisuutta nopeasti työtä varten tehdyllä algoritmilla. Sovellus on suuressa käytössä AAA-tason eli suurten pelistudioiden peleissä (Nelson et al., 2016; ks. Kuva 1).



Kuva 1. Kuvakaappaus SpeedTreen käyttöliittymästä.

Teknisesti käsiteltynä kasviston luontiin käytetään formaaleja kieliä. Jokaiselle kasvillisuuden alalajille on määriteltynä kielioppi, jota kasvisto noudattaa: jokaista kasvillisuuden atomaalista osaa pystyy seuraamaan jokin toinen atomaalinen osa. Yhdistämällä osia toisiinsa, seuraten aina esimerkiksi satunnaisesti valittua sopivaa kieliopin sääntöä päädytään luomaan kasvillisuutta, joka näyttää homogeeniseltä, mutta koostuu strukturaalisesti aidoista kasveista. SpeedTreen tapauksessa käytetään Lindenmayer-systeemejä (*Lindenmayer systems*, *L-systems*). L-systeemit ovat alaluokka formaaleja kieliä, joiden kielioppikäskyt suoritetaan kaikkiin aakkosiin samanaikaisesti, muistuttaen hyvin paljon todellisen kasvin kasvua (Nelson et al., 2016). Kuvassa 2 on havainnoillistettu L-systeemeiden käyttöä kasvillisuuden luomiseen. Kuvan kasvi

rakentuu toistamalla yksittäistä formaalia lausetta rekursiivisesti. Formaalissa lauseessa esiintyvät hakasulkeet kuuluvat *haarukoitujen L-systeemien* (*Bracketed L-systems*) syntaksiin ja symbolisoivat "tallennuspisteitä", joihin pystytään palaamaan myöhemmin kasvin piirtämisen yhteydessä, helpottaen säännön kirjoittamista.



Kuva 2. Heinäkasvin luominen haarukoidulla L-systeemillä (Nelson et al., 2016).

Pelikehityksen aikaiseen automaattiseen sisällöntuotantoon kuuluu myös muita PCG-tekniikoita, mutta suurinta osaa näistä pystytään myös soveltamaan suorituksenaikaisesti. Tämän vuoksi kriteerin täyttävät PCG-tekniikat on sijoitettu omien alilukujen alle, joissa näitä käsitellään tarkemmin.

3.2.2 Hakupohjaiset algoritmit

Taksonomian kuudes kysymys jakaa PCG-tekniikat samantien tapahtuviin ja vaiheittain tapahtuviin. *Hakupohjaiset PCG-tekniikat* (*Search-Based Procedural Content Generation*, SB-PCG) ovat vaiheittain tapahtuvia, evolutiivisia periaatteita noudattavia tekniikoita: ensin pelisisältöartefakteja luodaan hyvin useita kappaleita esimerkiksi satunnaisesti, jonka jälkeen spesialisoitunut algoritmi (*hyvyysfunktio*, *fitness function*) pisteyttää sisällön: algoritmi käy lävitse artefakteja ja pisteyttää ne ohjelmoijan antamien laadullisten kriteerien perusteella. Näitä parhaita instansseja mutatoidaan eli muokataan lievästi mutaatioalgoritmilla. Tätä prosessia jatketaan, kunnes joku instansseista saavuttaa riittävän suuren laatupistemäärän. Huomioitavaa on, että pisteytyksessä käytetyt kriteerit eivät ole konkreettisia vaan esimerkiksi reaaliarvoja

asteikolla. Luvussa 3.2.4 käsitellään konkreettisia kriteerejä *Answer Set Programmingin* muodossa. (Arnaboldi & Loiacono, 2019; Nelson et al., 2016)

Teknisesti selvennettynä pelisisältö on "koodattuna" kahdessa erilaisessa muodossa: *genotyyppinä* eli evolutiivisen algoritmin käyttämänä datana, sekä *fenotyyppinä* eli varsinaisena pelisisältönä (vrt. biologian vastaavat käsitteet). Fenotyyppiä eli pelisisältöä testataan ja pisteytetään hyvyysfunktion avulla, jonka perusteella genotyyppiä eli evolutiivista mallia kehitetään (Browne et al., 2010; havainnoillistettu myöhemmin esimerkin avulla). Evolutiivisia algoritmeja on sovellettu eri videopeligenreihin, mutta pääasiallisena painopisteenä tutkimukset ovat keskittyneet tasohyppelypelien kenttien ja rallipelien ratojen luomiseen (Arnaboldi & Loiacono, 2019).

Havainnoillistavana esimerkkinä hakupohjaisista algoritmeista tässä tutkielmassa esitellään Arnaboldi & Loiaconon vuonna 2019 tuottama tutkimus, jossa luodaan kenttägeneraattori ensimmäisen persoonan räiskintäpeli Cube 2: Sauerbratenia (Dot3 labs LLC, 2004) varten. Tutkimus perustuu kahteen aikaisempaan tutkimukseen, jotka käsittelevät samaa toteutusaihetta samasta pelistä, joten tutkimuksen PCG-tekniikka on tasoltaan kehittyneempää muihin vastaaviin tutkimuksiin verrattuna. Cube 2: Sauerbraten on myös erityisen monipuolinen evolutiivista algoritmiä varten, sillä peli sisältää eri kantaman aseita ja muita pelillisiä elementtejä, jotka mahdollistavat sekä hyökkääviä että puolustavia taktiikoita. Tutkimus myös kuvailee yleisesti automaattisen sisällöntuotannon haasteita, joita sisällön tulisi ottaa huomioon ja eliminoida: näistä kerrotaan, kun puhutaan pisteytyskriteereistä myöhemmin tässä luvussa.

Tutkimuksessa pelikentän genotyyppi eli mutatoitava data luodaan neliön muotoisten avoimien alueiden ja suorakulmion muotoisten käytävien kokojen ja koordinaattien avulla (ks. Kuva 3). Huomioitavaa on, että tämänkaltaisilla genotyypeillä ei pystytä luomaan esimerkiksi fenotyyppejä, jotka sisältävät pyöreitä huoneita. Muodostuvaan pelikenttään eli fenotyyppiin sijoitetaan poimittavia aseita ja muita vastaavia peliteknisiä elementtejä, jonka jälkeen karttaa testataan simuloimalla taistelua tietokonevastustajien (bottien) avulla. Simuloinnin jälkeen genotyyppiä eli huoneiden ja käytävien kokoelmaa muokataan evolutiivisesti: tämä tarkoittaa tässä tapauksessa sitä, että mutatointiprosessi muokkaa genotyyppiä niin, että huoneet ja käytävät voivat kasvaa tai pienentyä ko'oiltaan. Tämän lisäksi huoneet pystyvät myös siirtymään.

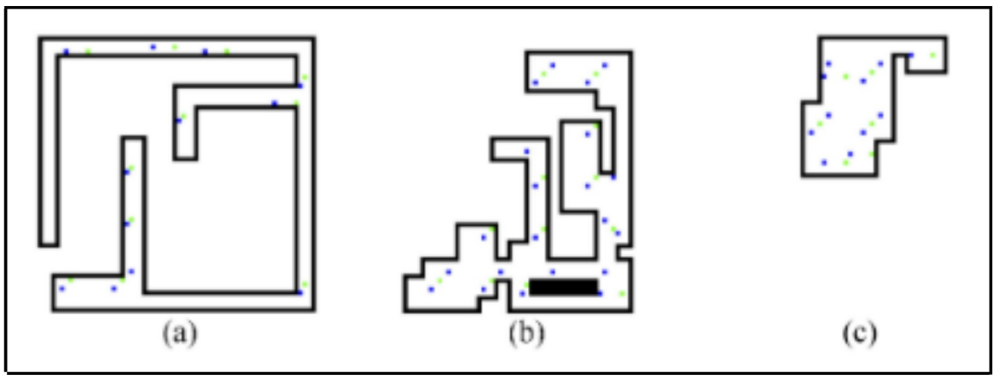
Pääasiallisina fenotyypin eli pelikentän pisteytyskriteereinä tutkimuksessa pidetään kartan tasapainoa, intensiivisyyttä ja elossapysymistä, joita mitataan suureina. Ensiksi, jos yksi boteista oli voitolla koko pelin ajan, kartta ei ole peliteknisessä mielessä tasapainossa. Toiseksi, jos kartalla käytävien taisteluiden välissä on liian paljon tai liian vähän "valmistautumisaikaa", voi pelaaja kokea kartan tylsäksi hitauden takia tai stressaavaksi nopeuden takia. Viimeiseksi, kartan tulisi mahdollistaa pelaajan

pärjäämisen: jos pelaaja ei kykene olemaan kartalla elossa suurimman osan peliaikaa, voi hän turhautua. Edellä kuvattua evolutiivista prosessia kyetään myös säätämään asettamalla joku edellä mainituista kriteereistä toisia tärkeämmäksi; tällöin prosessi tuottaa eri kriteereitä suosivia karttoja.

Kuvassa 3 esitellään kolme algoritmin luomaa karttaa. Kartoista A suosii pitkän kantaman aseita, kun taas C lyhyen kantaman aseita. Kartta B on tasapainotettu niin, että kumpaakin asekategoriaa pystytään käyttäämään tehokkaasti. Kartoista parhaimpana voidaan pitää karttaa B, sillä se mahdollistaa erilaisia pelityylejä enemmän kuin muut kartat.

Tutkimuksen tulos on kannustava: ihmisen ja botin pelatessa vastakkain luoduilla kartoilla evolutiivisen prosessin arvioimat tulokset tasapainosta ja muista kriteereistä vastasivat suurimmalta osin ihmisten mielipiteitä. Tämä tarkoittaa sitä, että tekniikkaa kyetään käyttämään mielekkäiden pelikenttien tekemiseen.

Hakupohjaisissa algoritmeissa on kuitenkin huonoja puolia. Ensinnäkin evoluutioprosessi on erityisen laskentaraskas, sillä jo prosessin alkuvaiheissa joudutaan tuottamaan monia kymmeniä ellei satoja pelisisältöinstansseja, joista suurin osa arvioidaan, todetaan huonolaatuisiksi ja poistetaan. Toiseksi, jos halutun pelisisällön kriteerit vaihtuvat, joudutaan jokainen algoritmeista (sisällön luonti, mutatointi ja pisteytys) muuttamaan erilaiseksi. Viimeiseksi, evolutiivinen näkökulma voidaan nähdä kokonaisuudessaan aivan liian monimutkaisena tapana luoda pelisisältöä, varsinkin silloin jos edellisten algoritmien koodipohja alkaa paisua liian suureksi ja epäselkeäksi. (Mateas & Smith, 2011)



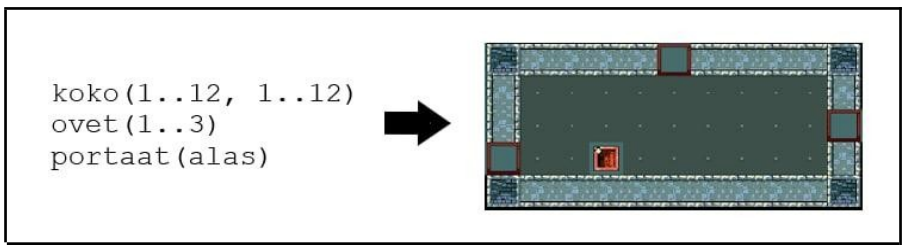
Kuva 3. Erilaisia SB-PCG-teknologialla Cube 2: Sauerbraten -peliin luotuja pelikenttiä pohjapiirroksina (Arnaboldi & Loiacono, 2019).

3.2.3 Answer Set Programming

Taksonomian kolmas ja seitsemäs kysymys kysyvät, kuinka hallitusti suunnittelija kykenee ohjaamaan pelisisällön luontia. *Answer Set Programming*, *ASP*, on looginen ohjelmointiparadigma, joka mahdollistaa hyvin suuren kontrollin luotavista

artefakteista. Pelikontekstissa ASP:n avulla ohjelmoija pystyy määrittelemään, mitä loogisia sääntöjä luotavan pelisisällön täytyy täyttää. Määrityksen jälkeen operaatioon erikoistunut algoritmi löytää kaikki mahdolliset pelisisällöt, jotka noudattavat ohjelmoijan asettamia sääntöjä (Antonova, 2015). Ohjelmoija pystyisi esimerkiksi määräämään, että muodostettavan pelikentän koko on pieni, ja se sisältää tietyntyyppisiä esineitä. On huomioitava, ettei ASP itsessään tarjoa algoritmia, joka luo sisällön, vaan vain ulkoiset raamit, joita ohjelmoijan luoma sisäinen algoritmi noudattaa (Mateas & Smith, 2011). ASP:n filosofia poikkeaa jyrkästi evolutiivisten tekniikoiden filosofiasta, sillä pelisisällön sisältämät elementit määrää suunnittelija evolutiivisen algoritmin sijasta, eikä evolutiivista tai vastaavaa prosessia ole (Hartzen, 2012). ASP:n hyötyinä nähdään erityisen yksinkertainen pelisisällön määrittely ja sen virheettömyys: ASP:n avulla luodut peliartifaktit eivät voi rikkoa pelisuunnittelijan määräämiä kriteereitä (Mateas & Smith, 2011).

Kuvassa 4 on esitetty yksinkertainen esimerkki ASP:stä. Kuvan vasemmalla puolella on suunnittelijan määräämät kriteerit yksittäiselle huoneelle: haettavan huoneen koko tulisi olla yhdestä kahteentoista ruutua pysty- ja vaakasuuntaan (maksimikoko 12x12), ovia tulisi olla yhdestä kolmeen, ja huoneessa on oltava portaat alas. Kuvan oikealla puolella on algoritmin löytämä sääntöjä noudattava huone: huone on kooltaan 3x9 ruutua, ovia on kolme, ja huoneessa on portaat alas.



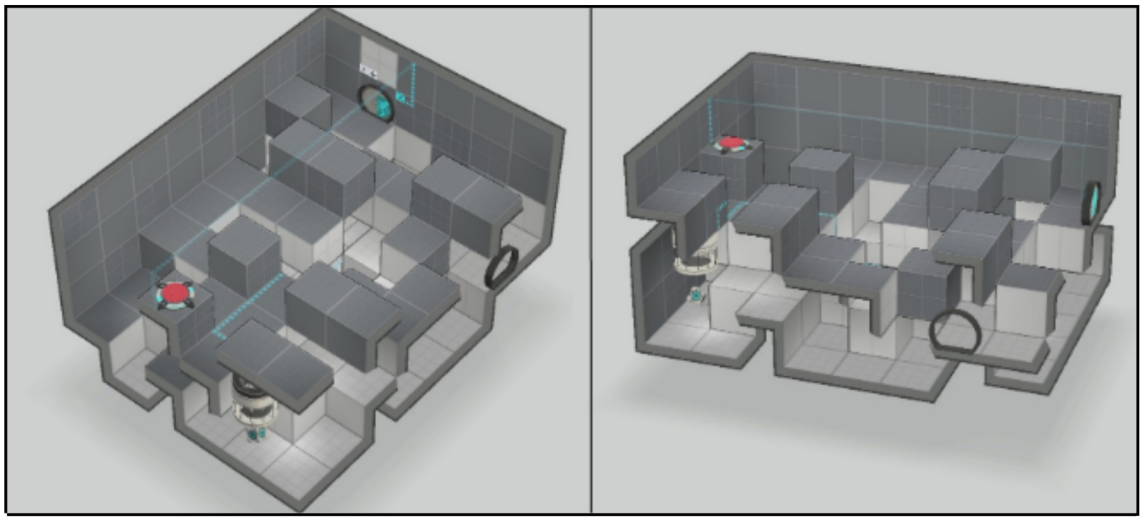
Kuva 4. Karkea esimerkki ASP:n avulla luoduista säännöistä ja sääntöjen avulla löydetystä huoneesta. Tutkielmaa varten tehty kuva käyttäen Nevanda-tilesettiä.

Havainnoillistavana esimerkkinä ASP:n käytöstä tässä tutkielmassa käytetään Antonovan vuonna 2015 suorittamaa diplomityötä. Tutkielmassa tutkitaan ASP-paradigman käytettävyyttä luomalla tasohyppely- ja pulmapeli Portalille (Valve Corporation, 2007) uusia tasoja. Portal-pelissä pelaajan tulee saavuttaa yksittäisten tasojen poistumisalueet asettelemalla pelikentän pinnoille pelaajan kaukosiirtämiseen kykeneviä portaaleja. ASP:tä käytettiin tutkimuksessa pelitason olemuksen määrittämiseksi. Ensinnäkin ASP:n avulla piti luoda tasoja, jotka ovat läpäistävissä, eli pelitason poistumisalueen tuli olla pelaajan saavutettavissa. Lisäksi pelitason elementtejä pystyttiin säätämään: kuinka suuri pelitaso on, kuinka monta portaalia

pelaajan tulisi vähimmäismääräisesti käyttää, kuinka paljon esteitä ja muita elementtejä on pelaajan ja poistumisalueen välissä ja niin edelleen. (Antonova, 2015; ks. Kuva 5)

Antonova toteaa, että ASP-tekniikalla tuotettua valikoimaa pelitasoja pitää karsia ihmisvoimin, sillä moni luoduista tasoista ei näytä esteettisesti mukavalta tai on liian triviaali todellisen pelin pulmaksi. Huomattavasti pienikokoisen mutta säännöiltään kompleksin pelitason luomiseen käytettävä aika normaalilla kuluttajan koneella kestää yli minuutin, mutta astettakin suurempi kestää tunteja, ellei kone kerkeä kaatumaan muistin loppuessa. ASP-teknologiaa ei siksi kyetä käyttämään kovinkaan monipuolisesti pelin pelaamisen aikana. Pelikehityksen aikana tämänkaltainen nopeus on kuitenkin hyväksyttävissä.

Edellä mainittuja sekä hakupohjaisten algoritmien ongelmia on yritetty korjata Hartzen ynnä muiden vuonna 2012 luomassa toteutuksessa yhdistämällä tekniikat yhteen niin, että evolutiivinen algoritmi käsittelee genotyyppinä ASP:n loogisia sääntöjä. Hartzenin ynnä muiden kaltaisia toteutuksia ei kuitenkaan ole lainkaan enempää, joten ideaa ei olla nähtävästi lähdetty kehittämään pidemmälle.



Kuva 5. ASP:n avulla luotu Portal-kenttä kuvattuna kahdesta eri kulmasta. Kenttää kokeilleet pelaajat kuvailivat kenttää "epäinhimillisen näköiseksi" ja "haastavaksi", mutta kuitenkin "miellyttäväksi" ratkaista (Antonova, 2015).

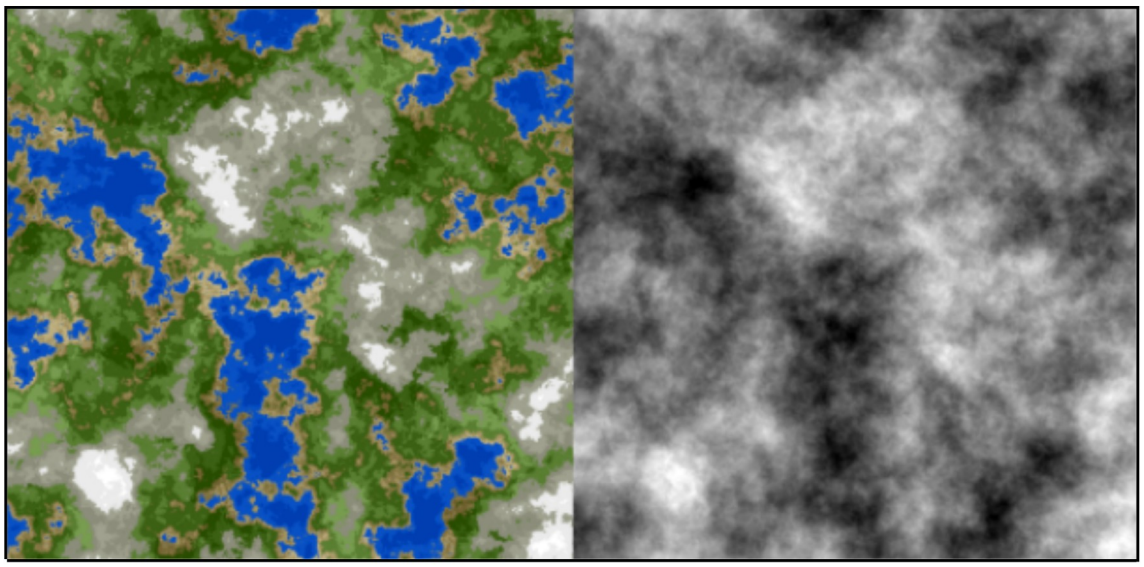
3.2.4 Kohina ja sen käyttökohteet

Kohina (*noise*) on satunnaisesti tuotettua signaalia. Hyvin tavanomainen esimerkki on vanhan television tuottama valkoinen kohina, joka esiintyy sekä äänenä että kuvana. Sähkötekniikan parissa kohina on signaalia haittaava elementti, mutta tietojenkäsittelytieteessä kohinan tuomia satunnaiselementtejä voidaan käyttää hyväksi luodessa pelisisältöä. Kenties käytetyin tekniikka 3D-pelimaailmojen automaattiseen luontiin on *kohina-algoritmit*, joiden suurin etu on niiden miltei täydellinen autonomisuus: niiden avulla luotu maasto ja erityisesti sen vaihteleva korkeus näyttää

realistiselta, vaikka pelikehittäjä ei sen huomattavammin parametrisoi maailmaansa (Bevilaqcua et al., 2011).

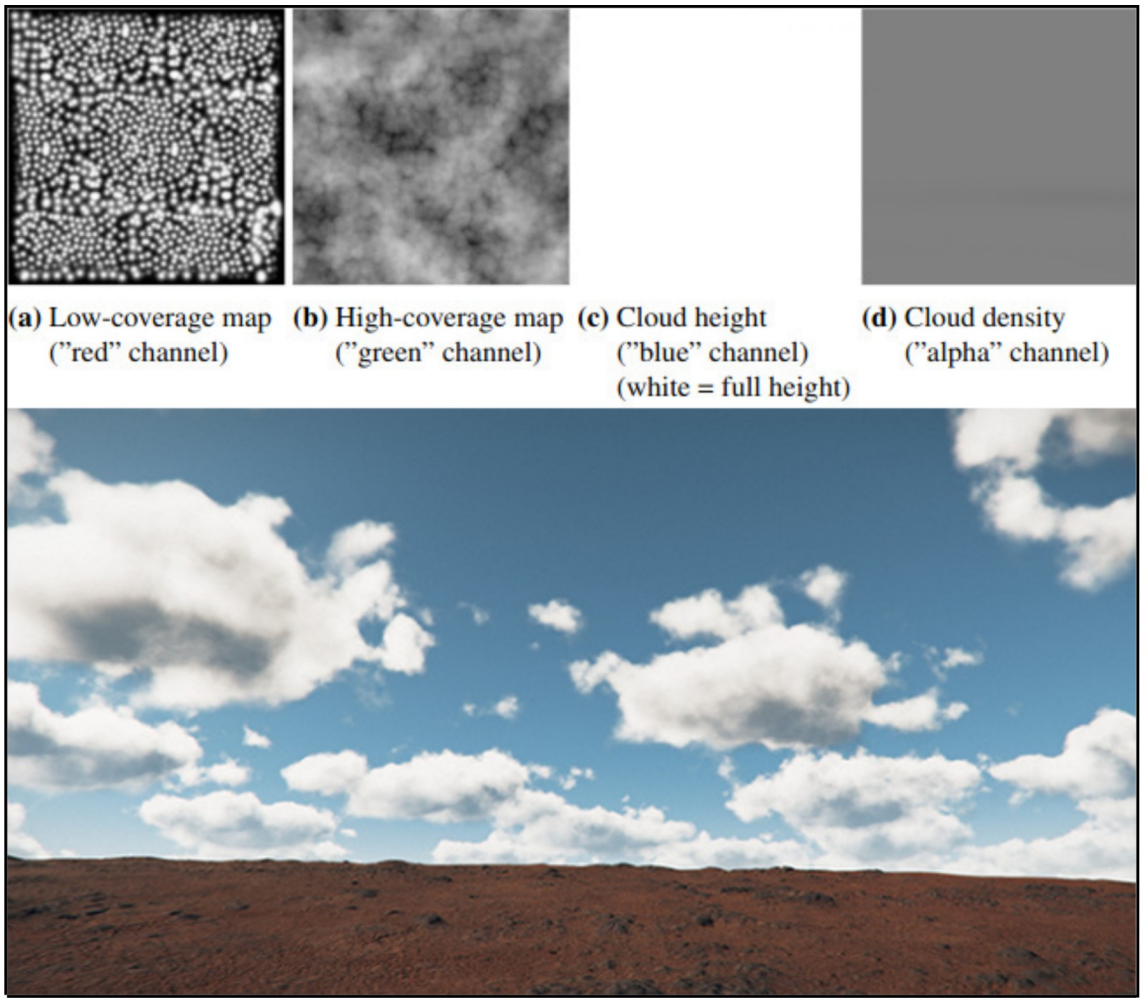
Tarkemmin kuvailtuna kohina-algoritmit ovat algoritmeja, jotka ottavat argumenttinaan koordinaatiston pisteen ja muuntavat sen esimerkiksi tiivistealgoritmisiä piirteitä hyväksikäyttäen reaaliarvoksi. Videopelikontekstissa tämä tarkoittaa sitä, että kohina-algoritmiin pystytään syöttämään esimerkiksi pelimaailman koordinaattipisteet, jotka algoritmi muuntaa maailman sen hetkiseksi korkeudeksi. Täysin puhdasta eli satunnaista kohinaa ei pystytä hyväksikäyttämään saman tien, sillä sen avulla luodun maailman korkeuskartta vaihtelisi jyrkästi ja täten olisi epärealistisen näköinen. Tämän vuoksi pelikehityksessä käytetään kohina-algoritmeja, jotka pyöristävät ja tasaavat tuottamansa kohinan ennen sen käyttöönottoa. Esimerkiksi hyvin tuottaneet videopelit Minecraft ja No Man’s Sky käyttävät kohina-algoritmeja maailmojensa luomisessa. (Bevilaqcua, 2011; Hyttinen, 2017).

Kuvassa 6 on esimerkki tavanomaisesta kohina-algoritmillisesti tuotetusta maailmasta, joka on kuvattuna kartankaltaisesti ylhäältä päin. Kohinakartan vaaleimmat pisteet kuvautuvat vuoriksi, kun taas tietyn tummuuskynnyksen ylittävät pisteet täyttyvät järvillä. Väliin jäävät korkeudet koostuvat rannoista ja kasvillisuuden peittämästä sisämaasta.



Kuva 6. Simplex-kohinan avulla luotu ylhäältä kuvattu maailma vasemmalla ja vastaava kohinakartta oikealla (Travall, 2018).

Kohinakarttoja käytetään myös volumetristen eli kolmiulotteisten objektien automaattiseen luomiseen. Kuvassa 7 on esillä automaattisesti luotuja pilviä ja niitä vastaavia kohinakarttoja, joita käsitellään *kanavina* (*channels*). Nämä erillään olevat kanavat määrittävät yksitellen pilvien muodon, koon, korkeuden ja tiheyden. Luomalla eriparametrisia kohinakarttoja pystytään luomaan eri pilvityyppejä. (Häggström, 2018)



Kuva 7. Kohinakarttoja ja niiden avulla luotuja kumpupilviä (Häggström, 2018).

Häggström mainitsee, että suurin ongelma pilviä muodostavan teknologian käyttämisessä oikean elämän sovelluksissa on renderöinnin hitaus, varsinkin silloin, jos pilvikerroksia on useita. Hän toteaa myös, että pilvien läpi matkustaminen esimerkiksi lentosimulaattorikontekstissa tuhoaa tekniikassa käytettyjä illusorisia keinoja, jotka vakuuttavat pilvien aitouden ainoastaan maan tasolta katsottaessa. Tämä oleellisesti rajoittaa teknologian käyttöä. Lisäksi Häggström epäilee, ettei tekniikkaa pystytä nopeuttamaan muuta kuin päivittämällä laitteistoa tehokkaammaksi.

3.2.5 Deterministisyydestä

Taksonomian viides kysymys kysyy, käyttääkö tekniikka satunnaisuutta pelisisällön luonnin aikana. Hyvin moni ellei suurin osa nykyajan PCG-tekniikoista hyödyntää satunnaisuutta sisällöntuotannon aikana, joka mahdollistaa ei-deterministisiä artefakteja (Mateas & Smith, 2011). Ei ole kuitenkaan vaadittua käyttää satunnaisuutta automaattisen luonnin yhteydessä: sisältö voidaan luoda myös deterministisesti eli niin,

että sisältö on täsmälleen sama sisällöntuotantoalgoritmin suorittamisten jälkeen. Tässä luvussa käsitellään deterministisyyttä PCG:ssä ja sen käyttötarkoituksia menemättä kuitenkaan yksittäisiin algoritmisiin toteutustapoihin, joita käsitellään tutkielmassa omissa kappaleissaan.

Deterministiset algoritmit ovat olleet suuressa suosiossa etenkin ROM-moduulien ja levykkeiden aikakaudella, sillä tilansäästäminen on ollut tärkeää näiden pienen tilan vuoksi. Esimerkiksi avaruusseikkailupeli Elite (Acornsoft, 1984) tiivistää pelinsä sisältämät tähtijärjestelmät käyttämällä tiiviisti säilöttyä dataa ja algoritmia datan purkamiseen. Tilansäästössä käytetyillä PCG-algoritmeillä on kuitenkin nykyaikanakin vielä käyttöä *tietokonedemoissa* eli tietokoneilla suoritetuissa yleensä visuaalisesti näyttävissä esityksissä: eräs demogenreistä edellyttää teoksen mahtuvan tietyn kokoiseen määrään tavuja. Pelikontekstissa erityisen tunnettu demo on *.kkrieger* (.theprodukkt, 2004), 96 kilotavun pelikategorian voittanut peli, joka tallentaa suurimman osan pelidatastansa PCG-tekniikoita käyttämällä (Hendrikx et al., 2013). Teknisemmin kuvailtuna peli ei tallenna resurssejaan kuten karttoja suoraan muistiin, kuten tavallinen videopeli tekee, vaan laskee ne pelin suorituksen aikana lähdekoodiin kirjoitetun algoritmin perusteella. Tämä tarkoittaa sitä, että muodostuvat tekstuurit ja muu data on suorituksen aikana tilapäisesti talletettuna tietokoneen RAM-muistiin.

Deterministisiä PCG-tekniikoita on kuitenkin käytetty myös nykyaikana pelaajille yhteisten loputtomien maailmojen luontiin. Tunnettu esimerkki on No Man’s Sky (Hello Games, 2016), joka luo saman suuren maailman jokaiselle pelaajalle käyttäen PCG-tekniikoita. Teknisesti peli käyttää samankaltaista lähestymistapaa maailmanluomiseen kuin tilansäästämisalgoritmit, mutta kykenee pelkkien annettujen koordinaattien avulla renderöimään koordinaateissa sijaitsevan ympäristön ruudulle. Tätä aihetta käsitellään tarkemmin luvussa 3.2.4.

4 Togeliuksen taksonomian jälkeisiä PCG-tekniikoita

Tässä luvussa käsitellään Togeliuksen taksonomian jälkeen kehitettyjä PCG-tekniikoita. Kummatkin tutkimuksen aikana löydetyistä uusista PCG-tekniikoista käyttävät hyväkseen *koneoppimista* (*Machine Learning*). Koneoppimisessa opetetaan tuotantoalgoritmi luomaan uutta sisältöä vanhan perusteella. Teknisesti tämä tapahtuu antamalla algoritmin löytää vanhasta sisällöstä tiettyjä säännöllisiä kaavoja, joita se opetuksen jälkeen kykenee käyttämään hyväkseen luodessaan uutta sisältöä. Hyvin tunnettu esimerkki ja koneoppimisen tutkimusala on syntetisoitujen kuvien ja videoiden luonti ihmisistä (*syväväärennökset*, *deepfake*). Esimerkiksi hakupohjaisista tekniikoista koneoppiminen eroaa niin, että taustalla on todellinen, koulutettavana oleva ja oppiva tekoäly eikä vain lista kriteerejä, joiden perusteella tulos pisteytetään. Koneoppiminen

ei itsessään ole uusi konsepti PCG:n piirissä, mutta seuraavaksi esitettyihin tekniikoihin ei viitata Nelsonin ynnä muiden alan kirjassa.

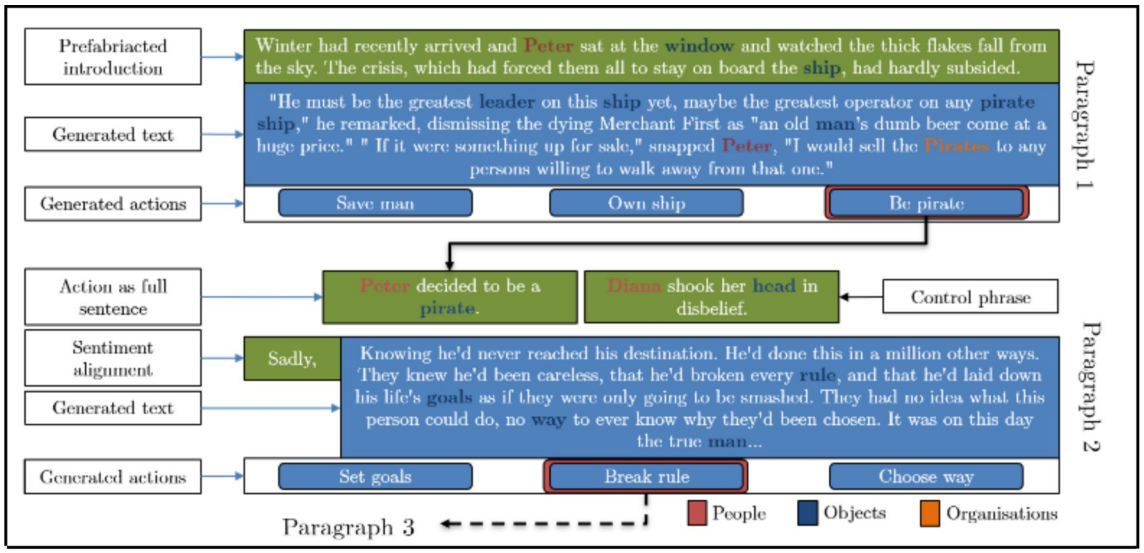
4.1 Luonnollisen kielen prosessoinnin pelillistäminen

Ensimmäinen koneoppimistekniikoista on *luonnollisen kielen prosessointiin* (*Natural Language Processing, NLP*) liittyvä pelillistäminen. Luonnollisen kielen prosessoinnin kenties parhain julkiselle yleisölle saatava malli on Generative Pre-trained Transformer 2 (GPT-2, OpenAI, 2019), joka jatkaa käyttäjän kirjoittamaa lyhyttä kehotetta aidosti uudella tekstillä. Pian julkaisunsa jälkeen GPT-2:n pelillisti AI Dungeon (Latitude, 2019), joka teki myös aiheen tunnetuksi suurelle yleisölle. Pelin ideana on *pelikirjan* (*Choose Your Own Adventure*) tavoin luoda GPT-2-mallin avulla skenaarioita, joihin pelaaja vastaa sillä, mitä aikoo tehdä. GPT-2:n avulla toteutettu algoritmi kykenee luomaan pätevän vastauksen pelaajan toimiin (ks. taksonomian neljäs kysymys). AI Dungeonin innoittamana aiheesta on kirjoitettu kohtalaisen runsaasti tieteellisiä julkaisuja, joista tähän tutkimukseen on valittu Effelsberg ynnä muiden vuonna 2020 luoma tarina-algoritmi.

Jotta Effelsbergin ynnä muiden toteutusta pystytään vertaamaan AI Dungeoniin, on käytävä läpi jälkeenmainitun heikkouksia. AI Dungeon kärsii erityisesti konseptien ja esineiden hahmottamisesta, sillä se ei usein esimerkiksi muista, mitä tavaroita pelaaja kantaa mukanaan tai mitä erilaisia funktioita erilaisilla tavaroilla on niiden valtaisan lukumäärän takia. Effelsberg ynnä muiden toteutus huomioi nämä heikkoudet ja helpottaa tarinan koherenttiuden säilymistä identifioimalla substantiiveja luokkiin, kuten eläimiin, artefakteihin, ruokaan ynnä muihin. Lisäksi tietyillä kovakoodatuilla verbeillä kyetään manipuloimaan maailmaa hallitusti: esimerkiksi kirjoittamalla “ota kivi maasta” muokataan pelaajan tavaraluetteloa lisäämällä sinne kivi ja poistetaan se maasta.

Kuvassa 8 on esitetty Effelsbergin ynnä muiden toteutuksen pääasiallinen toimintaperiaate. Edellä mainitun sanojen luokittamisen ohella toteutus myös analysoi tarinassa jo esiintyneitä substantiiveja, joiden perusteella toteutus luo pelaajalle tarinaan sopivia jatkovaihtoehtoja. Lukijaa kuitenkin kannustetaan arvioimaan kuvassa esitetyn tarinan kaunokirjallinen laatu.

Effelsbergin ynnä muiden toteutuksessa on kuitenkin vielä juonellisia ongelmia: toteutuksen tarinassa aiemmin luomat hahmot eivät ilmaannu tarinaan takaisin, ellei pelaaja referoi niihin. Ilman referointia yksittäinen luotu hahmo pysyy mallin muistissa vain noin pari kappaleen ajan, sillä malli huomioi vain viimeiseksi luotuja lauseita aktiivisena juonenkehityskontekstina. Tämän vuoksi mahdollisten hahmojen määrä yksittäisessä tarinassa on suhteellisen rajallinen. Kovin laajoja juonirakennelmia toteutus ei siis kykene luomaan.



Kuva 8. Effelsbergin ynnä muiden kontekstin huomioiva tarinanjatkaja.

4.2 Generatiiviset kilpailevat verkot

Neuroverkot (*Neural Networks*) ovat koneoppimisesta tunnettu tekoälyn kouluttamisen malli, joka muotoillaan ihmisen aivojen struktuurin kaltaisesti. Neuroverkkojen avulla tekoäly pystyy oppimaan tunnistamaan säännönmukaisuuksia lähdedatasta. *Generatiiviset kilpailevat verkot* (*Generative Adversial Networks, GAN*) ovat vuonna 2014 kehitetty koneoppimistekniikka, jossa kaksi neuroverkkoa luo sisältöä kilpailemalla keskenään. Yksi neuroverkoista opiskelee jo luotua sisältöä ja yrittää luoda sen perusteella uutta sisältöä, kun taas toinen verkoista yrittää määritellä, onko annettu sisältö todellista vai ensimmäisen verkon luomaa sisältöä. Analysoiva neuroverkko antaa palautetta luovalle neuroverkolle, joka tämän oppimisprosessin aikana oppii luomaan vakuuttavampaa sisältöä samalla kun toinen neuroverkko oppii paremmin tunnistamaan aidot kappaleet vastikään luoduista kappaleista. Generatiivisia kilpailevia verkkoja käytetään erityisesti aidontuntuisten valokuvien, kuten ihmiskasvojen, syntetisoinnissa. (Liu et al., 2018)

Videopelien kontekstissa GAN-tekniikalla on pyritty luomaan videopelien kenttiä. Aiheesta löytyy kaksi tieteellistä julkaisua, joista ensimmäisenä ilmestynyt luo Super Mario Bros. -peliin (Nintendo Co., Ltd., 1985) uusia kenttiä, kun taas toinen luo DOOM-pelin (id Software, 1993) pelikenttiä (Liu et al., 2018; Giacomello et al., 2018). Kummassakin tutkimuksessa neuroverkkoihin syötetään pelin alkuperäisten, pelisuunnittelijoiden suunnittelemien pelikenttien data, jonka neuroverkot käyvät edelläesitetyn prosessin avulla läpi. Kummankin tutkimuksen generoimat pelikentät ovat kuitenkin kovin alkeellisia, eivätkä Liun ynnä muiden tapauksessa ole edes usein voitettavissa.

5 Yhteenveto ja pohdintaa

Tässä tutkielmassa on tutkittu automaattisen sisällöntuotannon historiaa, motiiveja, suosiota, yleisiä sekä tuoreita teknisiä toteutustapoja sekä niiden hyötyjä ja haittoja. Tutkielma on vastannut johdannossa esitettyihin tutkimuskysymyksiin – minkälaista sisältöä PCG:llä luodaan ja millä tavoin PCG toteutetaan – esittämällä ja analysoimalla valikoituja PCG:tä tutkivia ja toteuttavia tutkimuksia. Taksonomiaan liittyvään tutkimuskysymykseen ja sen vastaukseen perehdytään tulevassa kappaleessa.

Kaupallisissa videopeleissä on käytetty hyvin kapeakatseisesti PCG-tekniikoita (Hendrikx et al., 2013; ks. Kaavio 1). Ensiksi tätä vaikeuttaa teknologioiden hankala käyttöönotto: kokonaisen automaattisen sisällöntuotantoalgoritmin luonti yhtä pientä projektia varten on usein liian paljon työtä suhteutettuna siitä saatuihin etuihin. Tällaisen algoritmin luonti useamman projektin käyttöön on myös haasteellista, sillä on erittäin hankalaa ellei jopa mahdotonta luoda kaikenkattavaa automaattista sisällöntuotantoalgoritmia, joka kykenisi luomaan haluttua sisältöä moneen yleensä hyvin erilaiseen peliprojektiin (Hendrikx et al., 2013). Muutenkin automaattisen algoritmin luonti on luonteeltaan investoivaa: jos työkalua tullaan käyttämään uudestaan, sen tuottamisen mahdollisuus todennäköisesti kasvaa.

Toiseksi tämän hetken teknologiaa tarkastellen voidaan huomata, etteivät automaattisen sisällöntuotannon tuottamat artefaktit ole aina miellyttävän näköisiä tai taiteellisesti oikeanlaisia suunnittelijan tai pelaajan mielestä, vaikka muuten olisivatkin peliteknisesti mielenkiintoisia (Antonova, 2015; Bevilaqcua et al., 2011). Kolmanneksi monet tekniikoista käyttävät liian paljon prosessointivoimaa, eivätkä täten ole soveltuvia suoritettavaksi kuluttajan tietokoneella (mm. Antonova, 2015; Häggström, 2018). Viimeiseksi kaikissa tekniikoissa ei myöskään pystytä poissulkemaan virheellisten artifaktien luomisen mahdollisuutta: esimerkiksi satunnaista pulmaa luodessa on mahdollisuus, ettei luotua pulmaa pystytä ratkomaan, joka on omiaan aiheuttamaan suurta harmia pelin integriteetille (Mateas & Smith, 2011; Liu et al., 2018).

Edellä mainitut huonot puolet ottaen huomioon pystytään kuitenkin havaitsemaan, että suurin osa nykypäivän automaattisen sisällöntuotannon tutkimuksesta keskittyy valmiiksi koulutetun tekoälyn integroimiseen niin, että pelisisältö vaikuttaisi mahdollisimman paljon pelisuunnittelijan tekemältä. Tämä tarkoittaa sitä, että painopisteenä on vanhan kopioiminen eikä todellisesti uuden tekeminen, joka voidaan nähdä tekniikoissa haittana. Tästä huolimatta koneoppimisen tehokkuuden vuoksi pelisisällön luominen on kovin vaivatonta, jos edeltäviä artefakteja on vain olemassa (Effelsberg et al., 2020). Ei ole vaikea uskoa, että koneoppimisesta löydettäisiin vielä muitakin sovelluksia PCG:n piiriin.

Johdannossa esitetyn pelimarkkinakatsauksen perusteella PCG-tekniikoita käyttäviä pelejä on vain muutama prosentti pelikannasta (ks. Kaavio 1). Tästä kannasta suurimman osan kattaa satunnaisten pelikenttien luonti. Syy juuri tämänkaltaiseen automaattiseen sisällöntuotantoon lienee sen pitkä historia: esimerkiksi vuosikymmeniä vanha “roguelike”-peligenre ja sen edustajat sisältävät miltei poikkeuksetta proseduraalisesti luotuja pelikenttiä (Browne et al., 2010; Arnaboldi & Loiacono, 2019). Pienen prosentuaalisen osuuden ohella on kuitenkin huomioitava PCG-tekniikoiden suosion kasvu viime vuosina. On mahdollista, että pelitrendien suosiessa PCG-tekniikoita yhä enemmän resursseja varataan sen tutkimiseen ja mahdollistamiseen videopeleissä.

Yhtenä tutkimuksen tavoitteena oli määrittää, onko toisessa luvussa esitelty Togeliuksen taksonomia vielä pätevä luokittelemaan PCG-tekniikoita. Nelson ynnä muiden vuonna 2016 julkaisema kirja automaattisesta sisällöntuotannosta videopeleissä esittelee Togeliuksen taksonomian ohella koneoppimisen yhtenä pelisisällön tuotantomahdollisuutena, mutta ei mainitse lainkaan luonnollisen kielen prosessointia eikä GAN-tekniikkaa. Togeliuksen taksonomia on tehty vuonna 2010 ja uudistettu vuonna 2016, josta spekuloitava teoksen seuraava päivitys voisi osua lähivuosille. Kuitenkin näyttää siltä, että tässä tutkielmassa esitellyt koneoppimista käyttävät tekniikat sopivat Togeliuksen taksonomiaan: luonnollisen kielen prosessoinnin pelillistäminen kategorisoituu järkevästi erityisesti neljännen kysymyksen käsittämään piiriin, ja GAN-tekniikka on vain yksi muoto aiemmin käytetyistä koneoppimisen muodoista, johon esimerkiksi Nelson ynnä muut viittaavat.

Automaattisen sisällöntuotannon aiheesta enemmän kiinnostuneille suositellaan Nelsonin ynnä muiden vuonna 2016 kirjoittamaa kirjaa aiheesta, jossa kuvaillaan tarkemmin tässä tutkielmassa esiteltyjä tekniikoita sekä tekniikoita, joita käytetään harvemmin tai jotka ovat liian teoreettisia oikean elämän käytäntöihin soveltuviksi.

Lähdeluettelo

Antonova, E. (2015). Applying Answer Set Programming in Game Level Design. Diplomityö. Aalto-yliopisto.

Arnaboldi, L., Loiacono, D. (2019). Multiobjective Evolutionary Map Design for Cube 2: Sauerbraten. *IEEE Transactions on Games*. IEEE. https://ieeexplore.ieee.org/document/8350029

Bevilacqua, F., de Carli, D. M., d’Ornellas, M. C. & Pozzer, C. T. (2011). A Survey of Procedural Content Generation Techniques Suitable to Game Development. *Brazilian Symposium on Games and Digital Entertainment.* IEEE. https://ieeexplore.ieee.org/document/6363215.

Browne, C., Stanley, K., Togelius, J. & Yannakakis, G. (2010). Search-Based Procedural Content Generation. *Proceedings of the 2010 international conference on Applications of Evolutionary Computation.* Springer.

https://link.springer.com/chapter/10.1007/978-3-642-12239-2_15

Effelsberg, W & Freiknecht, J. (2020). Procedural Generation of Interactive Stories using Language Models. *FDG '20: International Conference on the Foundations of Digital Games.* ACM. https://dl.acm.org/doi/fullHtml/10.1145/3402942.3409599

Giacomello, E., Lanzi, P. L. & Loiacono, D. (2018). DOOM Level Generation Using Generative Adversarial Networks. *2018 IEEE Games, Entertainment, Media Conference (GEM), Galway, Ireland.* IEEE. https://doi.org/10.1109/GEM.2018.8516539.

Grabska-Gradzińska, I., Grabska, E., Nowak L. & Palacz W. (2021). Application of Graphs for Story Generation in Video Games. *2021 Australasian Computer Science Week Multiconference (ACSW '21)*. ACM. https://dl.acm.org/doi/10.1145/3437378.3442693.

Haavisto, S. (2015). Proseduraalinen sisällöntuotanto – tarkastelussa pelikentät. Turun yliopisto. https://www.utupub.fi/handle/10024/113791

Hannula, R. (2020). Elinympäristöön sopeutuvien eläinlajien generoiminen videopeleissä. Tampereen yliopisto. https://trepo.tuni.fi/handle/10024/121658

Hartzen, A., Justinussen, T. & Togelius, J. (2012). Compositional procedural content generation. *PCG'12: Proceedings of the The third workshop on Procedural Content Generation in Games*. ACM. https://dl.acm.org/doi/10.1145/2538528.2538541.

Hendrikx, M., Iosup, A., Meijer, S. & van der Velden, J. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*. ACM. https://dl.acm.org/doi/10.1145/2422956.2422957

Hyttinen, T. (2017). Terrain synthesis using noise. Pro gradu -tutkielma. Tampereen yliopisto. http://urn.fi/URN:NBN:fi:uta-201705081539.

Häggström, F. (2018). Real-time rendering of volumetric clouds. Master's thesis. Umeå University. http://www.diva-portal.org/smash/get/diva2:1223894/FULLTEXT01.pdf

Liikkanen, J. (2015). Proseduraalinen pelisisällön luominen. Tampereen ammattikorkeakoulu. https://www.theseus.fi/handle/10024/99946)

Liu, J., Lucas, S. M., Risi, S., Schrum, J., Smith, A. & Volz, V. (2018). Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM. https://doi.org/10.1145/3205455.3205517.

Mateas, M. & Smith, A. M. (2011). Answer Set Programming for Procedural Content Generation: A Design Space Approach. *IEEE Transactions on Computational Intelligence and AI in Games.* IEEE. https://doi.org/10.1109/TCIAIG.2011.2158545.

Nelson, M. J., Shaker, N., Togelius, J. (2016). Procedural Content Generation in Games. Springer.

Nethack Wiki. (2021). Nevanda tileset for NetHack. https://nethackwiki.com/mediawiki/images/2/26/Nevanda.png

Reunanen, H. (2017). Proseduraalinen kyberpunk-pelimaailma. Tampereen Ammattikorkeakoulu. http://urn.fi/URN:NBN:fi:amk-2017121220718)

Travall, X. (2018). Procedural 2D Island Generation – Noise Functions. https://medium.com/@travall/procedural-2d-island-generation-noise-functions-13976bddeaf9

Valve Corporation. (2021). Steam Search. https://store.steampowered.com/search/