Tampereen yliopisto

Rama Hannula

ELINYMPÄRISTÖÖN SOPEUTUVIEN ELÄINLAJIEN GENEROIMINEN VIDEOPELEISSÄ

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Toukokuu 2020

TIIVISTELMÄ

Rama Hannula: Elinympäristöön sopeutuvien eläinlajien generoiminen videopeleissä
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2020

Videopelit ovat taloudellisesti erittäin merkittävä viihteenmuoto, jonka tuotannon kustannukset ovat huomattavan suuria. Kehityskustannuksia voidaan kuitenkin säästää tuottamalla osa pelien sisällöstä proseduraalisesti eli automaattisesti algoritmien avulla, jolloin tarvitaan vähemmän palkattua ihmistyövoimaa. Paitsi että näin voidaan säästää kehityskustannuksissa, voidaan samalla saavuttaa myös uudenlaisia pelikokemuksia, joita ei muuten pystyttäisi tuottamaan.

Tässä tutkielmassa tarkastelen erilaisten fiktiivisten eläinlajien tuottamista videopeleissä. No Man's Sky- ja Dwarf Fortress -peleissä tuotetaan eläinlajeja proseduraalisesti, mutta lajien sopimista ympäristöön ei näissä peleissä huomioida erityisen tarkasti. Tuotetut lajit voidaan kuitenkin saada sopimaan paremmin ympäristöönsä simuloimalla evoluutiota ja luonnonvalintaa geneettisellä algoritmilla. Koska monimutkaisten elinympäristöjen ja ekosysteemien vaikutusta lajin selviytymiseen on vaikea laskea suoralla sopivuusfunktiolla, voidaan kromosomien sopivuus laskea simulaatiopohjaisella sopivuusfunktiolla, jolloin ekosysteemin lajien välinen interaktio pystytään huomioimaan paremmin.

Esittelen toteuttamani Simulaatiopohjainen lajievoluutio -järjestelmän, joka käyttää geneettistä algoritmia ympäristöön sopeutuvien eläinlajien tuottamiseen. Toteutuksessa sopivuus lasketaan lajien selviytymisajan mukaan videopelisääntöihin perustuvassa simulaatioympäristössä. Koska eläinlajia kuvaavan kromosomin ominaisuudet vaikuttavat suoraan lajin kykyyn selviytyä ympäristössä, tuottaa järjestelmä erilaisiin ympäristöihin erilaisia eläinlajeja.

Avainsanat: proseduraalinen sisällöntuotanto, geneettiset algoritmit, videopelit

1 Johdanto

Viime vuosikymmeninä videopelit, tai lyhyesti pelit, ovat jatkuvasti kasvattaneet suosiotaan ja nousseet vannoutuneiden harrastajien kellareista valtavirtaan. Kaikenikäiset pelaajat kuluttavat videopelejä yhä enemmän, eri muodoissa ja eri alustoilla. Vuonna 2017 tehdyn tutkimuksen [Suomen virallinen tilasto (SVT) 2017] mukaan ainakin kerran vuodessa digitaalisia pelejä pelanneiden määrä Suomessa oli kasvanut 90-luvun alusta nelinkertaiseksi ja samalla mediaani-ikä oli noussut 19 vuodesta 35 vuoteen. Vuonna 2017 vähintään kerran kuussa videopelejä pelaavia yli kymmenvuotiaita oli 41 prosenttia Suomen väestöstä. Maailmanlaajuisesti videopelien liikevaihdon arvioitiin vuonna 2019 olevan 150 miljardia Yhdysvaltain dollaria, mikä on yli kaksi kertaa enemmän kuin maailman elokuvateollisuuden ja musiikkiteollisuuden liikevaihto yhteensä [Protocol 2020].

Videopelien suosion ja taloudellisen merkittävyyden kasvaessa myös niiden kehittämiseen käytettyjen resurssien määrä on kasvanut. Ensimmäiset videopelit kehitettiin tyypillisesti yksin tai muutaman hengen ryhmissä. Tekniikan kehittyminen on kuitenkin mahdollistanut aina vain monimutkaisempien ja visuaalisesti näyttävämpien pelien kehittämisen, minkä myötä videopelien kehittämiseen on alettu tarvita yhä enemmän eri osa-alueisiin erikoistuneita osaajia. Vuonna 2018 Suomessa oli 220 aktiivista pelialan yritystä, jotka yhdessä työllistivät yli 3200 työntekijää [Neogames 2018]. Yhdessä pelialan yrityksessä työskenteli siis keskimäärin 14,5 työntekijää.

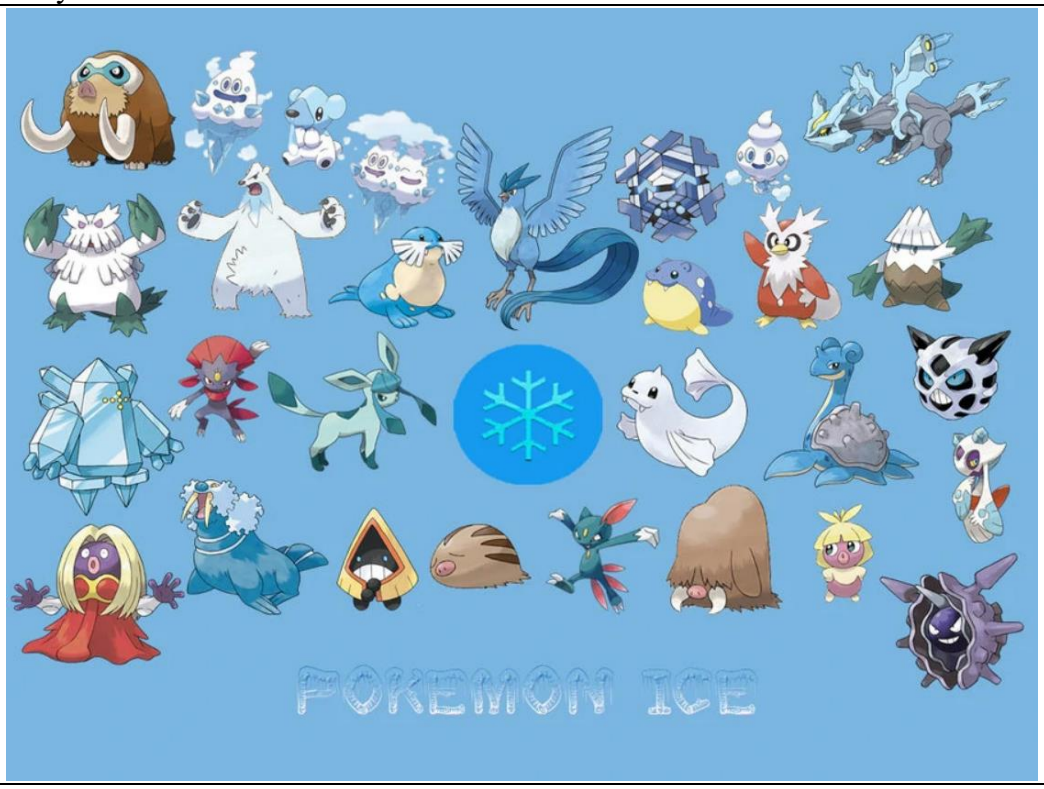
Videopelien kehittämiseen käytetyt resurssit ja kustannukset ovatkin nousseet aiempaan verrattuna huimasti. Erityisesti suuret AAA-luokan pelistudiot saattavat työllistää jopa useita satoja työntekijöitä eri osastoilla, ja pelien budjetit liikkuvat sadoissa miljoonissa euroissa.

Eräs suurimmista kulueristä itse kehitysvaiheessa on sisällön tuotanto [Hendrikx *et al.* 2013]. Perinteisesti sisältö on toteutettu käsin erilaisten suunnittelijoiden ja artistien toimesta. Pelisisältöä on kuitenkin mahdollista tuottaa myös proseduraalisesti eli automaattisesti ilman ihmiskehittäjän työpanosta erilaisten algoritmien avulla, jolloin voidaan vähentää sisällön tuottamiseen vaadittavaa työtä ja näin vähentää pelinkehityksen kuluja. Vaikka erilaisten sisältötyyppien kehittämiseen onkin olemassa lukuisia eri tekniikoita, on niiden käyttöönotto ollut pelialalla hidasta [Hendrikx *et al.* 2013].

Tuottamalla sisältöä proseduraalisesti voidaan paitsi säästää kustannuksissa, myös tarjota kokonaan uusia pelikokemuksia. Monet pelit hyödyntävät proseduraalisesti tuotetun sisällön monipuolisuutta ja vaihtelevuutta olennaisissa pelimekaniikoissa. Joissakin peligenreissä, kuten rogue-like -peleissä, koko peli rakentuu proseduraalisen sisällöntuotannon ympärille.

Erilaiset eläimet ja hirviöt ovat yleinen sisältötyyppi videopeleissä. Osa peleissä esiintyvistä eläinlajeista perustuu todellisiin lajeihin, mutta osa eläinlajeista voi olla myös täysin fiktiivisiä. Useissa fantasiapeleissä todellisten eläinlajien lisäksi esiintyviä fantasiaolentoja ovat esimerkiksi lohikäärmeet, peikot ja yksisarviset. Myös tieteispeleissä ja postapokalyptisissä peleissä esiintyy fiktiivisiä lajeja ja olentoja, kuten erilaisia avaruusolentoja ja mutantteja.

Eläinlajit esiintyvät peleissä niille tyypillisessä elinympäristössä. Havumetsässä pelaaja olettaa törmäävänsä sellaisiin lajeihin, jotka muistuttavat enemmän karhua, hirveä tai jänistä kuin mursua, pingviiniä tai kamelia. Ilmiö on havaittavissa esimerkiksi Pokémon-pelisarjassa, jossa ei esiinny lainkaan todellisia eläinlajeja. Monet fiktiivisistä Pokémon-hirviöistä muistuttavat ulkonäöltään todellisia eläinlajeja. Esimerkiksi useista kuvan 1 Pokémon-hirviöistä on mahdollista päätellä pelkän ulkonäön perusteella, että ne elävät kylmissä olosuhteissa.



Kuva 1. Monet jäätyypin Pokémon-hirviöt muistuttavat todellisia kylmillä alueilla eläviä eläinlajeja [PokéFanon Wiki 2020].

Eläinlajien tuottamista proseduraalisesti ei ole juuri hyödynnetty videopeleissä, eikä sitä ole käsitelty tieteellisessä keskustelussa. Tieteellinen keskustelu proseduraalisen sisällöntuotannon ympärillä on toistaiseksi keskittynyt pitkälti videopeliympäristöjen ja -kenttien tuottamiseen, vaikka samoja menetelmiä voitaisiin hyödyntää myös eläinlajien generoimisessa. Koska videopeliympäristöissä esiintyy usein erilaisia fiktiivisiä eläimiä ja hirviöitä, olisi hyödyllistä pystyä tuottamaan proseduraalisesti juuri kuhunkin ympäristöön sopivia eläinlajeja.

Tässä tutkielmassa tarkastelen proseduraalisten sisällöntuotannon menetelmien hyödyntämistä fiktiivisten eläinlajien generoimisessa videopeliympäristöihin. Kuvailen myös minkälaisia eläinlajeja voidaan tuottaa kehittämälläni sisällöntuotantojärjestelmällä, joka tuottaa tiettyyn elinympäristöön sopeutuvia eläinlajeja geneettisen algoritmin avulla. Järjestelmä tuottaa lajien abstrakteja esityksiä simuloimalla yksinkertaista ekosysteemiä videopeliympäristössä. Tällaista järjestelmää voisi hyödyntää esimerkiksi proseduraalisesti tuotettujen peliympäristöjen asuttamiseen niihin sopivilla eläinlajeilla.

Luvussa 2 esittelen proseduraalisen sisällöntuotannon käsitteitä ja ominaisuuksia, sekä esimerkkejä peleistä, joissa proseduraalista sisällöntuotantoa on hyödynnetty. Luvussa 3 tarkastelen lähemmin geneettisiä algoritmeja ja niiden käyttöä proseduraaliseen sisällöntuotantoon. Luvussa 4 hahmottelen geneettisen algoritmin hyödyntämistä ympäristöön sopeutuvien eläinlajien generoimisessa kahden esimerkkipelin avulla. Luvussa 5 esitän joitakin kehittämäni *Simulaatiopohjainen lajievoluutio* -järjestelmän (SPLE) toteutusyksityiskohtia ja luvussa 6 esittelen kyseisen järjestelmän eri ympäristöihin tuottamia eläinlajeja.

2 Proseduraalinen sisällöntuotanto videopeleissä

Tässä luvussa selitän, mitä on videopelien *sisältö* (content), ja miten sitä voidaan tuottaa proseduraalisesti. Tarkastelen myös syitä, miksi proseduraalista sisällöntuotantoa voi olla mielekästä käyttää apuna videopelien kehityksessä, ja mitä eri ominaisuuksia erilaisilta proseduraalisen sisällöntuotannon menetelmiltä voidaan tarvita eri tilanteissa. Luvun lopussa esittelen, miten proseduraalista sisällöntuotantoa on aiemmin hyödynnetty eri videopeleissä.

2.1 Pelin sisältö

Videopelit koostuvat *sisällöstä* (content), mutta tälle on vaikea antaa tarkkaa määritelmää. Sisällön voidaan ajatella käsittävän lähes kaiken, mitä peli pitää sisällään. Ainoastaan itse pelimoottori, jonka sisällä peli toimii, ja ei-pelaaja-hahmojen tekoäly eivät ole pelin sisältöä. [Togelius & Yannakakis 2011].

Hendrikx ja muut [2013] jakavat proseduraalisen sisällön eri luokkiin sen koon perusteella. *Pelin palaset* (game bits) ovat pelin pienimpiä perusosia, jotka eivät itsessään kiinnitä pelaajan huomiota. Pelin palasia ovat tekstuurit, äänet, kappaleiden käyttäytyminen ja keskinäinen interaktio sekä erilaiset luonto- ja karttaelementit, kuten kasvillisuus, rakennukset, vesi ja pilvet. [Hendrikx *et al*. 2013].

Pelitila (game space), eli toisin sanoen *pelimaailma* tai *peliympäristö*, on yksi-, kaksi- tai kolmiulotteinen ympäristö, jossa pelaaja toimii ja johon pelin tapahtumat ja palaset sijoittuvat. Pelitila voi olla joko *sisätilakartta* (indoor map), *ulkoilmakartta* (outdoor map) tai vesistö. Nämä eri tyyppiset pelitilat generoidaan yleensä eri menetelmillä. [Hendrikx

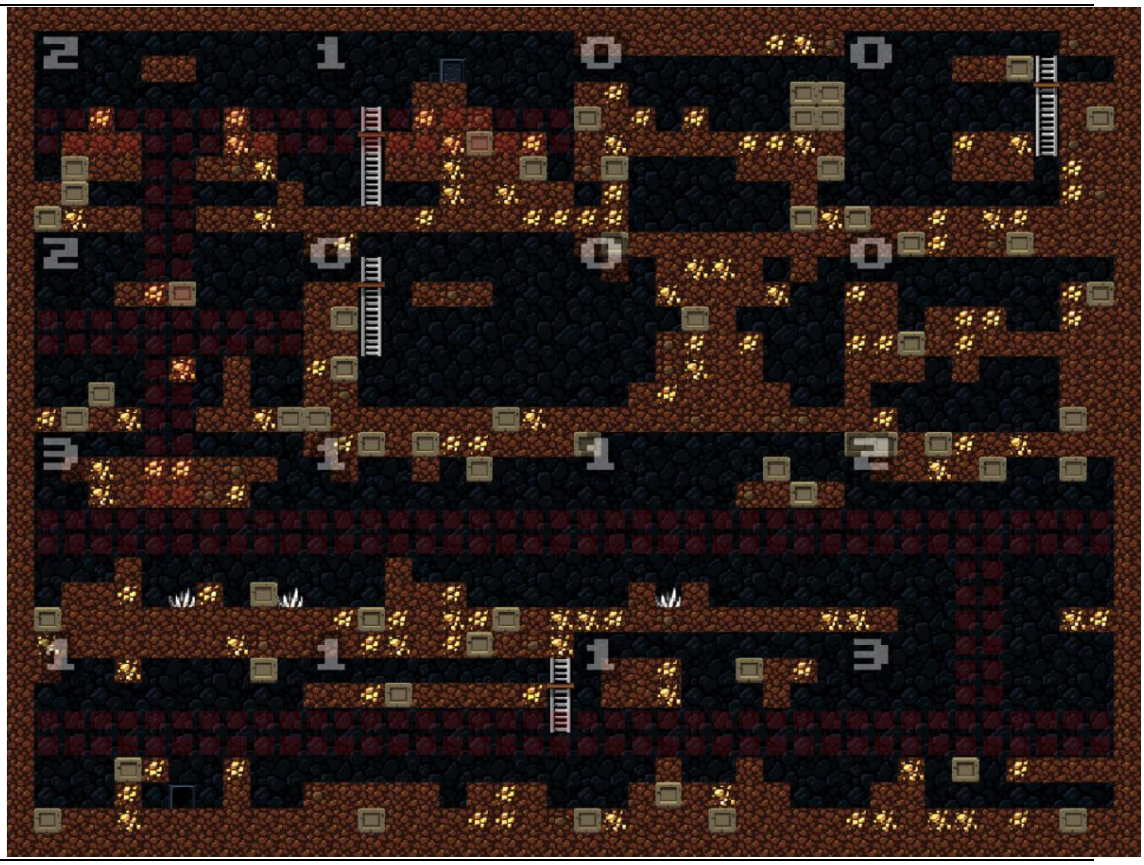
et al. 2013]. Tyypillinen proseduraalisen sisällöntuotannon käyttötarkoitus on luolastoympäristöjen generoiminen [Linden *et al.* 2013]. Joskus pelitiloista puhutaan myös esimerkiksi *karttoina* tai *tasoina*, hieman pelin genrestä riippuen.

Pelijärjestelmien (game systems) kuten ekosysteemien, tieverkostojen ja entiteettien käyttäytymismallien proseduraalisella generoinnilla voidaan saada peleistä uskottavampia. [Hendrikx *et al*. 2013].

Togelius ja muut [2011] jakavat sisällön vielä kahteen luokkaan sen tarpeellisuuden mukaan, *välttämättömään* (necessary) ja *valinnaiseen* (optional). Välttämätön sisältö on sellaista, jota pelaaja tarvitsee edetäkseen pelissä. Pelaaja ei siis voi mitenkään välttää tällaista sisältöä, vaan se tulee väkisinkin vastaan, mikäli pelaaja haluaa edetä pelissä. Valinnainen sisältö sen sijaan on sellaista, että pelaaja voi halutessaan ohittaa sen. [Togelius *et al.* 2011].

Koska pelin eteneminen riippuu välttämättömästä sisällöstä, täytyy sen aina olla toimivaa. Tämä asettaa rajoitteita välttämätöntä sisältöä tuottavalle algoritmille, sillä tällaisen algoritmin tulee aina tuottaa läpäistävää sisältöä, jotta pelaaja ei voi jäädä jumiin mahdottoman sisällön takia. Vaikka sisältö olisikin teoriassa läpäistävää, täytyy myös vaikeusasteen pysyä sopivana. Valinnaista sisältöä tuottavan algoritmin sen sijaan ei ole välttämätöntä tuottaa aina läpäistävää sisältöä, sillä pelaaja voi halutessaan sivuuttaa valinnaisen sisällön, jolloin peli on kokonaisuutena edelleen läpäistävissä. [Togelius *et al*. 2011].

Esimerkiksi Spelunky-pelissä [2008] tuotetaan proseduraalisesti kuvan 2 kaltaisia peliympäristöjä, joiden läpi pelaajan on aina kuljettava edetäkseen pelissä. Jokainen tuotettava ympäristö on siis pakollista sisältöä, joten sen on aina oltava läpäistävissä. Spelunky-pelin tapauksessa ympäristön tulee olla sellainen, että sen läpi kulkee varmasti avoin reitti aina alkupisteestä loppupisteeseen. Reitissä on huomioitava myös painovoima, sillä se rajoittaa liikkumista ylöspäin. Lisäksi ympäristö ei saa olla vaikeusasteeltaan liian haastava. Kuvassa 2 läpäistävä reitti on merkitty punaisilla ruuduilla.



Kuva 2. Yksi generoitu peliympäristö Spelunky-pelissä.

2.2 Proseduraalinen sisällöntuotanto

Proseduraalisella sisällöntuotannolla (procedural content generation) tarkoitetaan pelisisällön tuottamista tietokoneproseduurien tai algoritmien avulla, tyypillisesti satunnaislukugeneraattoria hyödyntäen. Satunnaisuus onkin tärkeä osa proseduraalista sisällöntuotantoa, ja se mahdollistaa sen, että muuttamalla vain muutamaa parametria, voidaan tuottaa lukematon määrä erilaista sisältöä [Amato 2017].

Sisällöntuotantomenetelmä voi olla joko *stokastinen* eli sattumanvarainen tai *deterministinen*. Stokastinen menetelmä tuottaa vaihtelevia tuloksia, vaikka sen ajaisi täysin samoilla parametreilla, kun taas deterministinen menetelmä tuottaa aina saman tuloksen, mikäli parametrit ovat samat. [Togelius *et al*. 2011]. Proseduraalisessa sisällöntuotannossa voidaan hyödyntää pelkän satunnaislukugeneraattorin lisäksi myös käyttäjän rajallista tai epäsuoraa syötettä, jolloin menetelmä on *mukautuva* (adaptive) [Shaker *et al.* 2016]. Esimerkiksi pelaajan aikaisempi suoritus voidaan huomioida ja tuottaa sen pohjalta sellaista sisältöä, jonka arvellaan olevan sopivan haastavaa kyseiselle pelaajalle.

Sisällöntuotanto voi tapahtua joko *online*- tai *offline*-aikana. Online-sisällöntuotanto tapahtuu ajonaikaisesti oikean pelaajan pelatessa peliä, kun taas offline-sisällöntuotanto tapahtuu jo pelin kehitysvaiheessa. Koska online-sisällöntuotanto tapahtuu kesken pelin, täytyy sen olla hyvin nopeaa, jotta pelin pelaaminen ei keskeydy sisällön tuottamisen

takia liian pitkäksi aikaa. Offline-sisällöntuotannolla ei ole samanlaista tarvetta nopeudelle, ja sitä voidaan käyttää esimerkiksi pelin kehityksen apuna. [Togelius *et al*. 2011].

Menetelmät voidaan jakaa *konstruktiivisiin* (constructive) ja *tuota-ja-testaa* (generate-and-test) -menetelmiin. Konstruktiivinen menetelmä tuottaa aina vain yhden kappaleen sisältöä ja suoritus päättyy. Tällainen menetelmä täytyy siis suunnitella siten, että se tuottaa varmasti joka kerta riittävän hyvää sisältöä. Tuota-ja-testaa -menetelmä taas tuottaa aina kerrallaan yhden sisällönkappaleen ja testaa, onko se riittävän hyvä. Mikäli sisällönkappale ei ole riittävän hyvä, se hylätään ja luodaan uusi, joka taas testataan. Tätä jatketaan, kunnes riittävän hyvä sisällönkappale löytyy. [Togelius *et al.* 2011].

Tuota-ja-testaa -menetelmien heikkous konstruktiivisiin menetelmiin verrattuna on niiden epävarma suoritusaika. Joskus riittävän hyvä sisällönkappale voi löytyä heti ensimmäisellä yrityksellä, toisinaan sen löytämiseen voi mennä lukuisia iteraatioita. Konstruktiivinen menetelmä taas tuottaa lopullisen sisällönkappaleen varmasti heti ensimmäisellä yrityksellä. Eräs tuota-ja-testaa -menetelmien alaluokka on hakuperustaiset menetelmät, joissa tuotettu sisältö arvioidaan ja pisteytetään, ja lopulta palautetaan parhaan pisteytyksen saanut sisällönkappale [Togelius *et al.* 2011].

Sisällön tuottamiseen on olemassa lukuisia lähestymistapoja, ja sopiva tapa riippuu aina tuotettavan sisällön laadusta ja halutusta lopputuloksesta. Jotkin menetelmät ovat parempia tietyissä asioissa kuin toiset, mikä rajoittaa sitä, mitä menetelmiä kussakin tilanteessa voidaan hyödyntää. Hakuperustaiset menetelmät, jotka ovat yksi tuota-ja-testaa -menetelmien erikoistapaus, kuuluvat tutkituimpiin sisällöntuotannon menetelmiin [Shaker *et al.* 2016]. Niihin kuuluvia geneettisiä algoritmeja käsitellään tarkemmin luvussa 3.

2.3 Etuja

Alun perin proseduraalisen sisällöntuotannon käyttötarkoituksena videopeleissä on ollut tilansäästö, sillä tuottamalla sisältöä proseduraalisesti, ei tarvita yhtä paljon etukäteen tuotettua sisältöä. Sisältöä tuottava algoritmi vie huomattavasti vähemmän tallennustilaa kuin manuaalisesti esituotettu sisältö, joten näin pelitiedostot saadaan pakattua pienempään tilaan. [Amato 2017]. Yksi ensimmäisistä peleistä, joissa proseduraalista sisällöntuotantoa on käytetty hyväksi, on Rogue [1980]. Rogue-pelissä luodaan kaikki pelikentät proseduraalisesti, jolloin tiedostokoko pysyy pienenä. Vaikka nykytietokoneilla lukuisten pelikenttien tallentaminen ei ole mikään ongelma, pystyi Rogue-peli tällä tavoin kiertämään silloisten tietokoneiden rajoituksia ja tarjoamaan lähes rajattomasti erilaisia peliympäristöjä.

Tuottamalla pelisisältöä proseduraalisesti, voidaan myös vähentää kehitysprosessin aikana sisällön tuottamiseen kuluvaa aikaa ja resursseja. Paitsi että siten voidaan vähentää

suurten peliprojektien kustannuksia, tarjoaa se myös pienille pelialan yrityksille, jotka eivät muuten pysty tuottamaan yhtä paljon sisältöä, mahdollisuuden tuottaa sisältörikkaampia pelejä. [Shaker *et al.* 2016]

Yksi proseduraalisen sisällöntuotannon suurimpia etuja on myös sen kyky tuottaa lähes rajaton määrä erilaista sisältöä. Tämä mahdollistaa sen, että pelaajalle voidaan tarjota jokaisella pelikerralla uutta, kiinnostavaa sisältöä, mikä lisää pelin uudelleenpelattavuusarvoa [Smith *et al.* 2011]. Jatkuvasti vaihteleva pelisisältö tarjoaa myös uudenlaisia tapoja pelata, joita ei voida saavuttaa tuottamalla sisältöä manuaalisesti. Kun sisältö on joka kerralla uutta, täytyy pelaajan aina sopeutua vastaan tuleviin haasteisiin. Rogue-like -peligenren pelit, jotka ovat inspiroituneet alkuperäisestä Rogue-pelistä [1980], hyödyntävät paljon proseduraalista sisällöntuotantoa juuri rajattoman sisällön takia. Näin jokainen pelikerta on erilainen, ja peli voi olla käytännössä loputon, sillä sisältöä voidaan tuottaa enemmän kuin kukaan pystyy pelaamaan.

Nykyään suuret AAA-luokan pelistudiot käyttävät proseduraalista sisällöntuotantoa yleensä juuri ajansäästöön ja kehityskustannusten pienentämiseen. Esimerkiksi SpeedTree-ohjelmistoa [2002] on käytetty monissa suurissa pelituotannoissa työkaluna realistisen kasvillisuuden automaattiseen mallintamiseen, jolloin kehittäjät voivat keskittyä muun sisällön tuottamiseen. Pienet ja itsenäiset kehittäjät käyttävät proseduraalista sisällöntuotantoa laajemmin eri tarkoituksiin, kuten uudenlaisten pelikokemusten kehittämiseen.

2.4 Haluttavia ominaisuuksia

Proseduraalisen sisällöntuotannon menetelmiltä voidaan haluta erilaisia ominaisuuksia niiden käyttötarkoituksen mukaan. Kaikkia hyviä ominaisuuksia on jotakuinkin mahdotonta saavuttaa kerralla, joten niiden välillä on aina tehtävä kompromisseja. [Shaker *et al.* 2016].

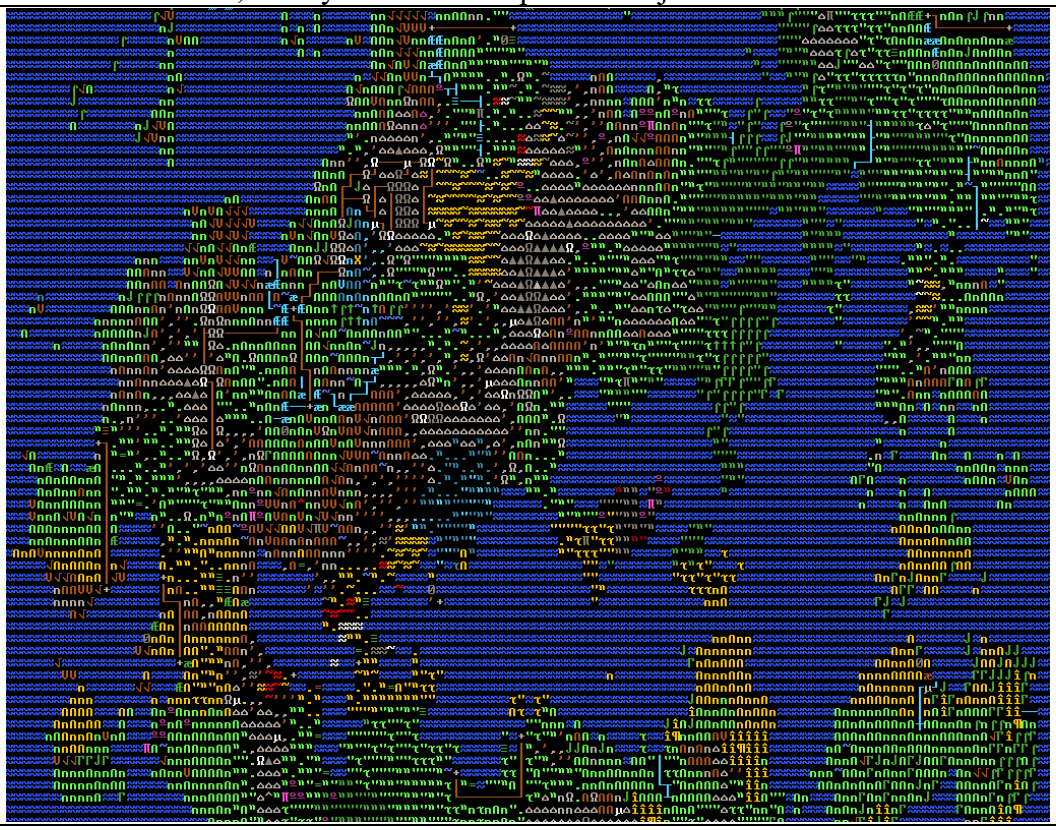
- *Nopeus* on ominaisuus, jota tarvitaan erityisesti silloin, kun sisältö halutaan tuottaa suoraan ajonaikana. Pelikokemus kärsii, jos peli keskeytyy jatkuvasti sisällöntuotannon takia. Nopeus ei ole niin olennaista silloin, kun sisältöä tuotetaan offline-aikana, esimerkiksi jo pelin kehitysvaiheessa.
- Menetelmän *hallittavuus* on joissakin tilanteissa olennainen ominaisuus. Tällä tarkoitetaan sitä, että sisällöntuotantomenetelmä voidaan saada tuottamaan juuri halutunlaista sisältöä.
- *Luotettavuus* on tarpeen, kun sisällön tulee aina täyttää tietyt laatuvaatimukset. Tämä on erityisen tärkeää silloin, kun tuotettava sisältö on pelaajalle välttämätöntä. Valinnaisen sisällön tapauksessa luotettavuus ei ole yhtä tärkeä ominaisuus, sillä pelaaja voi tarvittaessa ohittaa heikkolaatuisen sisällön.
- Tuotetun sisällön tulisi yleensä olla *uskottavaa*, eli se ei saa näyttää algoritmin luomalta.

- Yleensä menetelmän *ilmaisuvoiman* halutaan olevan korkea, jolloin se kykenee tuottamaan mahdollisimman erilaista sisältöä mahdollisimman laajasti. Tuotetun sisällön tulisi kuitenkin samaan aikaan olla järkevää ja riittävän laadukasta, joten ilmaisuvoimaa voidaan yleensä lisätä vain luotettavuuden ja uskottavuuden hinnalla.

2.5 Esimerkkejä proseduraalisesta sisällöntuotannosta videopeleissä

Useimmat proseduraalista sisällöntuotantoa hyödyntävät pelit tuottavat sillä yleensä jotakin tiettyä sisältötyyppiä, ja muu sisältö tuotetaan perinteiseen tapaan manuaalisesti. Esimerkiksi toimintaroolipelissä Diablo [1997] tuotetaan proseduraalisesti luolastoja ja esineitä, kun taas Minecraft-pelissä [2011] generoidaan proseduraalisesti valtava kolmiulotteinen pelimaailma. Molemmissa peleissä proseduraalisesti tuotettu sisältö on tärkeä osa pelin viihdyttävyyttä, mutta suurin osa sisällöstä on kuitenkin tuotettu manuaalisesti.

Ääriesimerkki proseduraalisen sisällöntuotannon käytöstä on Dwarf Fortress -peli [2006], jossa luodaan proseduraalisesti koko pelin maailma, kansat, yksittäiset henkilöt ja hirviöt sekä näiden kaikkien historiat. Kuvassa 3 näkyy erään Dwarf Fortress -pelissä generoidun maailman kartta. Maailma generoidaan käyttäjän antamien parametrien ja siemenluvun pohjalta. Prosessi on täysin deterministinen, joten sama maailma voidaan tuottaa aina uudelleen, kun syötetään samat parametrit ja sama siemenluku.



Kuva 3. Eräs Dwarf Fortress -pelissä generoitu maailma.

3 Geneettiset algoritmit

Geneettiset algoritmit (genetic algorithm) ovat hakuperustaisia optimointimenetelmiä, joiden toimintaperiaate perustuu evoluutioon ja luonnonvalintaan. Tavoitteena on jäljitellä luonnon kykyä sopeutua erilaisiin ympäristöihin perinnöllisen lisääntymisen ja *mutaation* aikaansaaman itseohjautuvuuden avulla. Kuten luonnonvalinnan, myös geneettisten algoritmien toiminta perustuu sattumanvaraisuuteen eli ne ovat stokastisia. [Gen & Lin 2007]. Proseduraalisen sisällöntuotannon näkökulmasta geneettiset algoritmit kuuluvat tuota-ja-testaa -menetelmiin, ja niitä ei yleensä käytetä online-sisällöntuotantoon niiden laskennallisen raskauden takia.

Tässä luvussa käsittelen geneettisten algoritmien toimintaa yleisellä tasolla sekä niiden käyttöön liittyviä yksityiskohtia. Esittelen joitakin yleisimpiä geneettisiä operaatioita, joita tarkastelen myös esimerkkien avulla proseduraalisen sisällöntuotannon näkökulmasta.

3.1 Yleiskuva

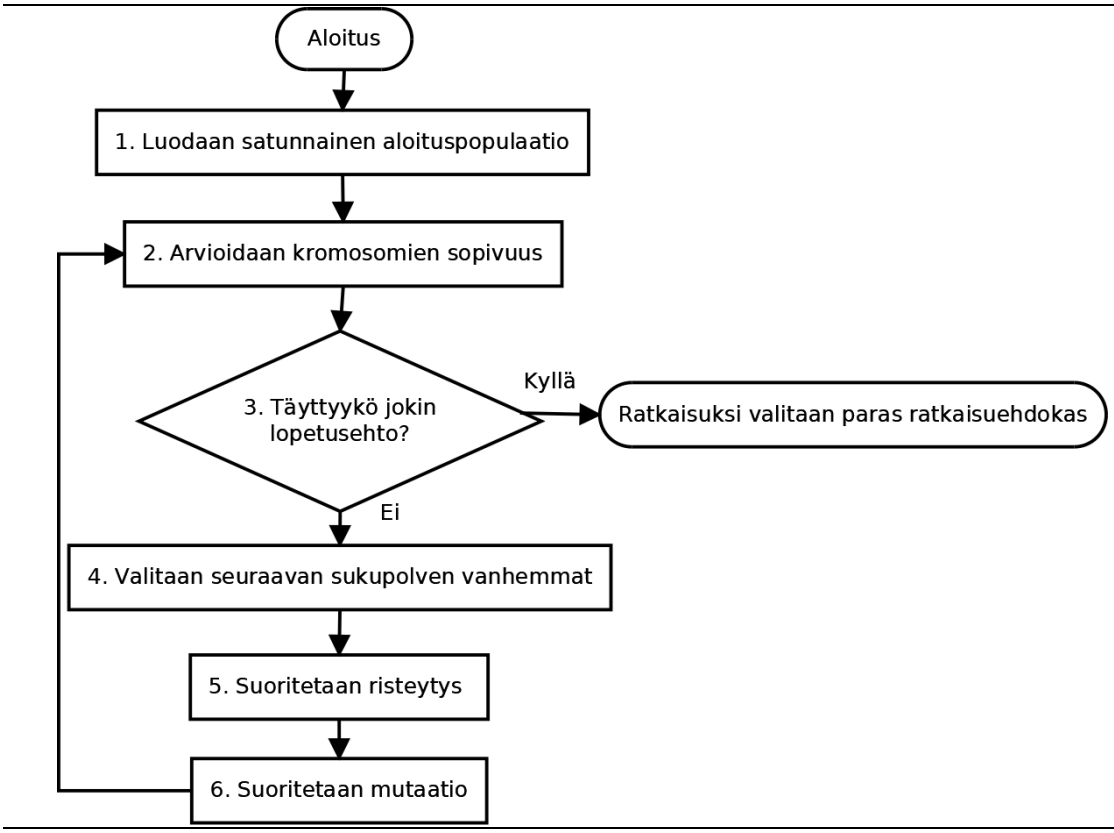
Geneettinen algoritmi käsittelee ongelman *ratkaisuehdokkaiden* (potential solution) joukkoa eli *populaatiota*. Yksittäisen ratkaisuehdokkaan koodausta algoritmin käsittelemään muotoon kutsutaan *kromosomiksi* vastaavan biologian termin mukaan. Kromosomin *sopivuus* (fitness) kuvaa sen esittämän ratkaisuehdokkaan kykyä ratkaista kohdeongelma, ja se määritellään *sopivuusfunktion* avulla. Populaatio alustetaan alussa täysin satunnaisilla ratkaisuehdokkailla, jonka jälkeen valitaan parhaat ratkaisuehdokkaat *risteytymään* keskenään, jolloin saadaan tuotettua seuraava *sukupolvi*. Uudet, seuraavan sukupolven ratkaisuehdokkaat perivät vanhempiensa ominaisuuksien osajoukot *rekombinaation* avulla ja saattavat myös muuttua *mutaation* myötä. Uusia sukupolvia tuotetaan, kunnes löydetään riittävän hyvä ratkaisuehdokas tai jokin muu lopetusehto, kuten etukäteen määrätty sukupolvien maksimimäärä, täyttyy. [Gen & Lin 2007].

Geneettisen algoritmin vaiheet on esitetty yksinkertaistetusti kuvan 4 vuokaaviossa. Algoritmi perustuu ennalta määrättyyn sopivuusfunktioon $fitness$, valintafunktioon $selection$, risteytysfunktioon $crossover$, mutaatiofunktioon $mutation$ sekä lopetusehtojen joukkoon $\{e_1, \dots, e_m\}$. Algoritmi etenee seuraavasti:

1. Luodaan aloituspopulaatio eli ensimmäinen sukupolvi g_1 satunnaisista kromosomeista $c \in g_1$.
2. Nykyisen sukupolven g_n, missä n on sukupolven järjestysluku, jokaisen kromosomin $c \in g_n$ sopivuusarvo arvioidaan sopivuusfunktiolla $fitness(c) = f_c$.
3. Tarkastellaan täyttyykö jokin lopetusehto e_i. Mikäli jokin lopetusehdoista täyttyy, palautetaan sellainen kaikkien sukupolvien korkeimman sopivuusarvon f_{max}=$max(fitness(c) \mid c \in \bigcup_{i=1}^{n} g_i)$ ratkaisuehdokas c_{max} , jolle

$fitness(c_{max}) = f_{max}$, ja algoritmin suoritus päättyy. Mikäli mikään lopetusehto ei täyty, jatketaan algoritmin suorittamista.

4. Valitaan seuraavan sukupolven g_{n+1} yksilöiden vanhemmat nykyisestä sukupolvesta g_n valintaoperaatiolla $selection(g_n) = P$, missä P on joukko pareja (p_a, p_b) joille $p_a, p_b \in g_n$.
5. Jokaiselle vanhempien parille $(p_a, p_b) \in P$ suoritetaan risteytysoperaatio, jolloin saadaan vanhempien ominaisuuksia yhdistelevät jälkeläiset $crossover(p_a, p_b)$=$\{c_{ab}, c_{ba}\}$.
6. Jälkeläisille suoritetaan niiden ominaisuuksia muokkaava mutaatio-operaatio $mutation(c) = c', \in g_{n+1}$, jolloin näistä muodostuu seuraava sukupolvi g_{n+1}. Kasvatetaan sukupolven järjestyslukua *n* ja palataan vaiheeseen 2. Silmukkaa toistetaan, kunnes jokin lopetusehto täyttyy.



Kuva 4. Geneettisen algoritmin vaiheet.

3.2 Ratkaisuehdokkaan koodaus kromosomiksi

Kromosomi eli *perimä* tai *genotyyppi* on joukko ominaisuuksia, jotka määrittelevät yksittäisen ratkaisuehdokkaan. Sana kromosomi itsessään on lainattu biologiasta, jossa kromosomilla tarkoitetaan DNA-rakennetta, joka sisältää eliön geeniperimän. Geneettisen algoritmin kromosomin ominaisuuksien joukko vastaa DNA:n sisältämien geenien joukkoa. Geneettisen algoritmin yksittäisiä kromosomin ominaisuuksia

kutsutaankin myös *geeneiksi*. Esimerkiksi eläintä kuvaavan kromosomin geenejä voisivat olla säkäkorkeus, turkin paksuus ja ruokavalio.

Samalla tavoin kuin biologian kromosomi määrää, minkälainen eliö sen pohjalta kasvaa, myös geneettisen algoritmin kromosomi määrää minkälaisen ratkaisuehdokkaan kromosomi kuvaa. Tapa jolla ratkaisuehdokas on *koodattu* kromosomiksi voi vaikuttaa ratkaisuehdokkaan sopivuuteen, sillä kromosomin perusteella arvioidaan sen esittämän ratkaisuehdokkaan sopivuus.

Ratkaisuehdokasta voidaan joskus käyttää kromosomina suoraan, jolloin kyseessä on *suora koodaus.* Yleensä ratkaistava ongelma on kuitenkin liian monimutkainen, jotta näin voitaisiin tehdä. Suoraan koodatut kromosomit tuottavat helposti mahdottomia jälkeläisiä ja sisältävät turhia geenejä. Tämän ongelman välttämiseksi ratkaisuehdokas voidaan koodata *epäsuorasti*, jolloin kromosomiin otetaan mukaan vain merkityksellisiä geenejä. [Holland 1992]. Epäsuoralla koodauksella voidaan tehostaa hakua ja kromosomin sopivuuden arviointia.

Koodattu kromosomi voidaan purkaa takaisin ratkaisuehdokkaaksi ja siten tuottaa kromosomin *fenotyyppi* eli ilmentymä. Kromosomin esitystapa määrää sen, minkälaisia ratkaisuehdokkaita kromosomien pohjalta voidaan muodostaa. Ratkaisuehdokkaan koodaustapa vaikuttaa siis suoraan geneettisen algoritmin kykyyn esittää ratkaisuavaruuden mahdollisia ratkaisuehdokkaita. [Shaker *et al.* 2016]. Esimerkiksi videopelihahmon ulkonäköä generoitaessa epäsuorasti koodatun kromosomin geenejä voisivat olla sellaiset ominaisuudet kuten pituus, hiusten väri ja nenän pyöreys. Tämän kromosomin perusteella voitaisiin luoda sitä vastaava 3D-malli, eli kromosomin fenotyyppi. Samassa tapauksessa suora koodaus tarkoittaisi 3D-mallin käyttämistä sellaisenaan kromosomina, jolloin kaikki mallin kulmapisteet ja niiden muodostamat monikulmiot olisivat kromosomin geenejä.

Hyvä kromosomin koodaus on sellainen, että jokainen muutos geeneissä johtaa merkitykselliseen muutokseen fenotyypissä [Holland 1992]. Pieni muutos kromosomissa vaikuttaa vain vähän fenotyyppiin ja siten myös kromosomin sopivuuteen. Kromosomilla tulisi siis olla korkea *paikallisuus*. [Shaker *et al*. 2016]. Tämä tarkoittaa sitä, että mutatoidun kromosomin tulisi edelleen muistuttaa sen vanhempia [Gen & Lin 2007]. Yksi esimerkki huonon paikallisuuden omaavasta koodauksesta on pelkkä siemenluku, jossa minkä tahansa yksittäisen osan vaihtaminen muuttaa lopputuloksen täysin. Tämä vastaisi satunnaishakua.

Muita hyvän kromosomin ominaisuuksia ovat minimaalinen tilankäyttö ja nopea geneettisten operaatioiden suoritus. Lisäksi kromosomin tulee pystyä esittämään vain mahdollisia ratkaisuja. [Gen & Lin 2007]. Pelisisällön tuottamisen näkökulmasta kromosomin koodauksen tulee pystyä esittämään mahdollisimman laajasti kiinnostavaa

sisältöä siten, että korkean sopivuuden omaavien kromosomien löytäminen on edelleen riittävän helppoa [Cardamone *et al.* 2011].

Kromosomin koodaustavan valinta on tärkeä osa geneettisen algoritmin käyttöä, sillä se vaikuttaa huomattavasti algoritmin tehokkuuteen ja ilmaisuvoimaan, eli kykyyn kuvata sisältöavaruutta. Holland [1992] käyttää binääristä koodausta, missä kromosomi esitetään yksiulotteisena bittien jonona. Yksi bitti vastaa yhtä geeniä ja kuvaa yhden ominaisuuden esiintymisen tai puuttumisen. Useimpia ongelmia on kuitenkin hyvin haastavaa esittää pelkkänä sarjana yksitäisistä biteistä koostuvia geenejä, joten myös muita ratkaisuja voidaan käyttää. Geeneinä voidaan käyttää bittien sijasta esimerkiksi reaalilukuja, kokonaislukuja tai tietorakenteita, ja koodaus voi olla joko yksiulotteinen tai moniulotteinen. [Gen & Lin 2007].

Ratkaisuehdokkaan koodaustavan valinta riippuu lopulta kuitenkin täysin ongelman sovellusalueesta. Pelisisällön tapauksessa tuotettava sisältö on usein luonteeltaan hyvin vaihtelevaa, joten myös koodaustavat vaihtelevat huomattavasti. Esimerkiksi Angry Birds -pelin kenttien generoinnissa kentän kromosomi koodataan taulukoksi vertikaalisia pylväitä [Ferreira & Toledo 2014]. Jokainen pylväs koostuu päällekkäisistä palikoista, jotka muodostavat rakennelmia. Toisaalta kilpa-ajopelin ratojen koodauksena käytetään sarjaa yhtä pitkiä, mutta vaihtelevan levyisiä radanpätkiä, jotka saattavat olla joko suoria tai eriasteisia kurveja [Togelius *et al.* 2006]. Liapis ja muut [2013] käsittelevät videopelikarttoja abstrahoituina *karttaluonnoksina* (map sketch), jotka muodostuvat kaksiulotteisesta ruudukosta *laattoja* (tile). Cardamone ja muut [2011] esittävät ensimmäisen persoonan ammuntapeliin jopa neljä vaihtoehtoista koodausta.

3.3 Sopivuuden arvioiminen

Yksittäisen kromosomin kuvaaman ratkaisuehdokkaan sopivuutta, eli kykyä ratkaista kyseessä oleva ongelma, arvioidaan sopivuusfunktion avulla. Kromosomin sopivuuden perusteella suoritetaan valintaoperaatio, jolla seuraavan sukupolven vanhemmiksi valitaan todennäköisemmin sellaisia kromosomeja, joiden sopivuus on korkea [Holland 1992]. Näin matalan sopivuuden kromosomit ikään kuin kuolevat pois, ja korkean sopivuuden kromosomit jatkavat lisääntymistä, jolloin seuraavan sukupolven populaatio koostuu suhteessa parempien kromosomien jälkeläisistä [Gen & Lin 2007].

Sopivuusfunktio saa parametrinaan kromosomin ja palauttaa tämän sopivuuden lukuarvona. Sopivuusfunktion arvojoukko riippuu ongelmasta, mutta eri ongelmien sopivuusarvot voidaan normalisoida yhtenäisyyden saavuttamiseksi eri ongelmien välillä [Gen & Lin 2007]. Jossakin tilanteissa voi olla mielekästä vähentää kromosomin sopivuusarvoa tiettyjen ominaisuusyhdistelmien ilmenemisen takia, jolloin lopullinen sopivuusarvo voi olla myös negatiivinen. Sopivuusfunktion arvojoukon alkio x voidaan normalisoida lineaarisesti välille $[0,1]$ kaavalla

$$x_{normal} = \frac{\mathrm{x} - \mathrm{x}_{\min}}{\mathrm{x}_{\max} - \mathrm{x}_{\min}},$$

missä x_{min} on arvojoukon pienin arvo ja x_{max} on arvojoukon suurin arvo. Näin pienimmästä sopivuusarvosta tulee nolla, suurimmasta yksi ja loput sijoittuvat lineaarisesti tälle välille alkuperäisestä arvojoukosta riippumatta.

Sisällön tuottamisen näkökulmasta sopivuusfunktio tulisi rakentaa mittamaan jotakin tuotettavan sisällön haluttavaa ominaisuutta, kuten esimerkiksi pelattavuutta tai viihdearvoa. Näitä on kuitenkin hyvin hankala mitata matemaattisesti niiden subjektiivisen luonteen takia. Sopivuusfunktion suunnittelu on siis pitkälti pelin suunnittelijasta ja tämän tavoitteista kiinni. Hyvän sopivuusfunktion kehittäminen onkin yksi haastavimmista ongelmista geneettisen algoritmin käytössä sisällön tuottamisen välineenä. [Shaker *et al.* 2016].

Proseduraalisen sisällöntuotannon sopivuusfunktiot voidaan jakaa kolmeen luokkaan: *suoriin*, *interaktiivisiin* ja *simulaatiopohjaisiin* sopivuusfunktioihin. Suorat sopivuusfunktiot muodostavat sopivuusarvon suoraan kromosomin arvojen perusteella. [Togelius *et al.* 2011]. Esimerkiksi sokkeloa tuotettaessa sopivuusarvo voitaisiin laskea suoraan sokkelon läpäisevien reittien pituuksien, mutkien ja kokonaislukumäärän mukaan. Monimutkaista pelisisältöä ei ole kuitenkaan aina mahdollista arvioida suoran sopivuusfunktion avulla, vaan sisällön sopivuus täytyy arvioida peliä pelaamalla, jolloin kyseessä on interaktiivinen sopivuusfunktio [Togelius *et al.* 2011]. Jos esimerkiksi tuotettavassa sokkelossa olisi reittien lisäksi myös interaktiivisia elementtejä, kuten erilaisia hirviöitä ja aarteita, suoran sopivuusfunktion kehittäminen kävisi haastavaksi. Tällöin olisi luotettavampaa arvioida sisältöä pelaamalla peliä, jolloin entiteettien väliset monimutkaiset interaktiot voidaan ottaa huomioon.

Koska jokaisen ratkaisuehdokkaan peluuttaminen ihmispelaajalla on hyvin työlästä ja aikaa vievää, voidaan oikeiden pelaajien sijaan käyttää keinotekoisia agentteja simuloimaan pelin pelaamista. Tällöin sopivuusfunktion suorittaminen on huomattavasti nopeampaa kuin interaktiivisesti oikeiden ihmispelaajien avulla. Simulaatiopohjaiset sopivuusfunktiot ovat tästä huolimatta usein huomattavasti raskaampia kuin suorat sopivuusfunktiot, eivätkä ne sen takia yleensä sovellu online-sisällön tuottamiseen. [Togelius *et al.* 2011].

3.4 Valintamenetelmät

Uuden sukupolven luomiseksi valitaan nykyisestä populaatiosta joukko vanhempien pareja risteytettäväksi keskenään, ja näiden jälkeläisistä muodostetaan seuraava sukupolvi. Tavoitteena on jokaisella uudella sukupolvella kasvattaa populaation keskimääräistä sopivuutta valitsemalla vanhemmiksi todennäköisemmin sellaisia kromosomeja, joilla on korkea sopivuus [Gen & Lin 2007]. Vanhempien pari voi koostua myös kahdesta samasta kromosomista. Vanhempien pareja valitaan niin kauan, kunnes

haluttu populaation koko täyttyy, jonka jälkeen parien kromosomit risteytetään keskenään. Käsittelen risteytystä tarkemmin kohdassa 3.5.

Sopivuuteen suhteutetut valintamenetelmät (fitness proportionate selection) noudattavat ajatusta korkean sopivuuden todennäköisemmästä valinnasta hyvin suoraviivaisesti, sillä niissä kromosomin todennäköisyys tulla valituksi seuraavan sukupolven vanhemmaksi määräytyy suoraan suhteessa sen sopivuusarvoon [Goldberg & Deb 1991]. Yksinkertaisessa *rulettivalinnassa* (roulette wheel selection) vanhempien parien muodostamisessa sukupolvesta g_n, kromosomin $c \in g_n$ todennäköisyys tulla valituksi vanhemmaksi on sen sopivuusarvon suhde kaikkien nykyisen sukupolven kromosomien sopivuusarvojen summaan

$$\mathrm{p_c} = \frac{fitness(c)}{\sum_{i=1}^{|g_n|} fitness(c_i)}$$

[Shukla *et al.* 2015]. Jokainen kromosomi on valinnassa siis ikään kuin rulettipyörän sektori, jonka kulman suuruus on suhteutettu sen sopivuusarvoon. Ruletissa kuula pysähtyy todennäköisimmin sen sektorin kohdalle, jonka kulma on suurin ja epätodennäköisimmin sen kohdalle, jonka kulma on pienin. Samalla tavoin todennäköisimmin valitaan kromosomi, jonka sopivuus on korkein ja epätodennäköisimmin se, jonka sopivuus on pienin.

Sijoitusvalinta (ranking selection) käyttää valintatodennäköisyyden laskemiseen kromosomin sijoitusta, kun populaation kromosomit järjestetään parhaimmasta huonoimpaan niiden sopivuusarvojen mukaan [Goldberg & Deb 1991]. Mitä korkeamman sijoituksen kromosomi saa, sen todennäköisempää on, että se valitaan vanhemmaksi. Näin todennäköisyys ei ole riippuvainen sopivuusarvojen suuruuksien keskinäisestä suhteesta, vaan ainoastaan niiden järjestyksestä. Kromosomin $c \in g_n$ sijoitus voidaan laskea lineaarisesti

$$rank(c) = |\mathrm{g_n}| - \mathrm{c_{position}} + 1,$$

missä $\mathrm{c_{position}} \in \{1, \dots, |g_n|\}$ on kromosomin järjestysluku sopivuuden mukaan järjestetyssä sukupolvessa g_n. Näin kaikista korkeimman sopivuuden kromosomi saa sijoituksen $|g_n|$ ja matalimman sopivuuden kromosomi sijoituksen 1. Tällöin kromosomin c todennäköisyys tulla valituksi vanhemmaksi on sen sijoituksen suhde kaikkien kromosomien sijoitusten summaan

$$\mathrm{p_c} = \frac{rank(c)}{\sum_{i=1}^{|g_n|} rank(c_i)}.$$

Valintatodennäköisyyttä voidaan myös painottaa eksponentiaalisesti, jolloin kyse on *eksponentiaalisesta sijoitusvalinnasta* (exponential ranking selection) [Shukla *et al.* 2015].

Turnausvalinta (tournament selection) on valintamenetelmä, jossa nykyisestä sukupolvesta g_n poimitaan satunnaisesti n kromosomia $\{c_1, \dots, c_n\}$, ja näistä

korkeimman sopivuusarvon yksilö valitaan vanhemmaksi [Goldberg & Deb 1991]. Voidaan siis ajatella, että satunnaisesti poimittujen kromosomien joukko on ikään kuin turnauksen lohko, missä lohkon koko on n. Lohkon kromosomit kamppailevat keskenään sopivuusarvojaan vertailemalla siitä, mikä yksilöistä valitaan jatkoon. Kun lohkon koko n on kaksi, eli kerralla vertaillaan vain kahta kromosomia, joista toinen valitaan, puhutaan *binäärisestä turnauksesta* (binary tournament) [Goldberg & Deb 1991].

Koska populaation kokonaissopivuus voi laskea sukupolvesta toiseen satunnaisen valinnan, risteytyksen ja mutaation myötä, on kannattavaa siirtää aina jokin määrä parhaita ratkaisuehdokkaita sellaisenaan sukupolvesta seuraavaan, jotta niitä ei menetetä. Tätä menettelyä kutsutaan *elitismiksi*.

Vaikka on olemassa muitakin valintamenetelmiä, proseduraalisessa sisällöntuotannossa on käytetty paljon yksinkertaisia menetelmiä kuten rulettivalintaa ja turnausvalintaa. Esimerkiksi Liapis ja muut [2013] käyttävät yksilöiden valintaan rulettivalintaa ja yhden yksilön elitismiä. Angry Birds -pelin kenttien generoinnissa taas käytetään binääristä turnausvalintaa [Ferreira & Toledo 2014].

3.5 Uuden sukupolven muodostaminen

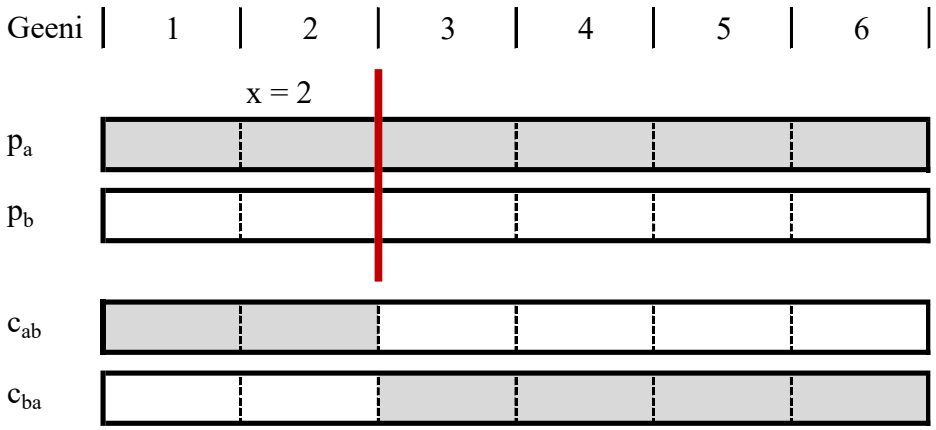
Geneettinen algoritmi tuottaa seuraavan sukupolven risteyttämällä aiemman sukupolven parhaita kromosomeja ja mutatoimalla näiden jälkeläisiä. Biologiassa rekombinaatiolla tarkoitetaan suvullisessa lisääntymisessä tapahtuvaa geenien periytymistä molemmilta vanhemmilta jälkeläiselle, joka tehostaa luonnonvalinnan vaikutusta verrattuna suvuttomaan lisääntymiseen, missä rekombinaatiota ei tapahdu [Barton & Charlesworth 1998]. Geneettinen algoritmi pyrkii mukailemaan tätä evoluution mekanismia uusien kromosomien muodostamisessa.

Geneettisen algoritmin risteytyksessä yhdistetään keskenään kaksi korkean sopivuusarvon kromosomia, jolloin näiden genotyypit rekombinoituvat jälkeläisen genotyypiksi. Risteytysoperaatiossa muodostuvat jälkeläiskromosomit omaavat siis sattumanvaraisesti valitut osajoukot vanhempiensa ominaisuuksia, jolloin saattaa syntyä uusia, aiempaa sopivampia ominaisuusyhdistelmiä. Operaation tulee olla sellainen, että se tuottaa vain mahdollisia jälkeläisiä. [Gen & Lin 2007].

Erilaiset kromosomiesitykset vaativat erilaisia risteytysoperaatioita [Poon & Carter 1995]. Binäärimerkkijonokoodaukselle voidaan tehdä yksinkertaisia risteytysoperaatioita kuten *yhden pisteen risteytys* (single point crossover), *monen pisteen risteytys* (n-point crossover) tai *yhtenäinen risteytys* (uniform crossover). Näitä risteytysoperaatioita voidaan soveltaa myös esimerkiksi reaalilukukoodauksen kanssa [Herrera *et al.* 2003].

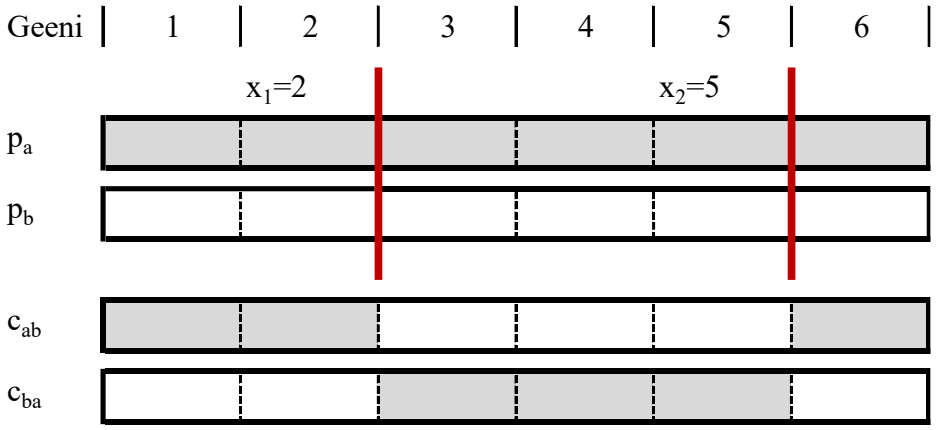
Yhden pisteen risteytyksessä valitaan kuvan 5 mukaisesti bittijonon pituudelta l satunnainen leikkauspiste $x \in \{1, \dots, l-1\}$, jonka mukaan geenit jaetaan vanhemmilta (p_a, p_b) jälkeläisille c_{ab} ja c_{ba}. Jälkeläisen c_{ab} geenit ovat tällöin yhdistelmä sen vanhempien geenejä siten, että geenit väliltä $\{1, \dots, x\}$ periytyvät vanhemmalta p_a ja

geenit väliltä $\{x+1, \dots, l\}$ vanhemmalta p_b, kun taas jälkeläisen c_{ba} geenit väliltä $\{1, \dots, x\}$ periytyvät vanhemmalta p_b ja väliltä $\{x+1, \dots, l\}$ vanhemmalta p_a. [Lim *et al.* 2017].



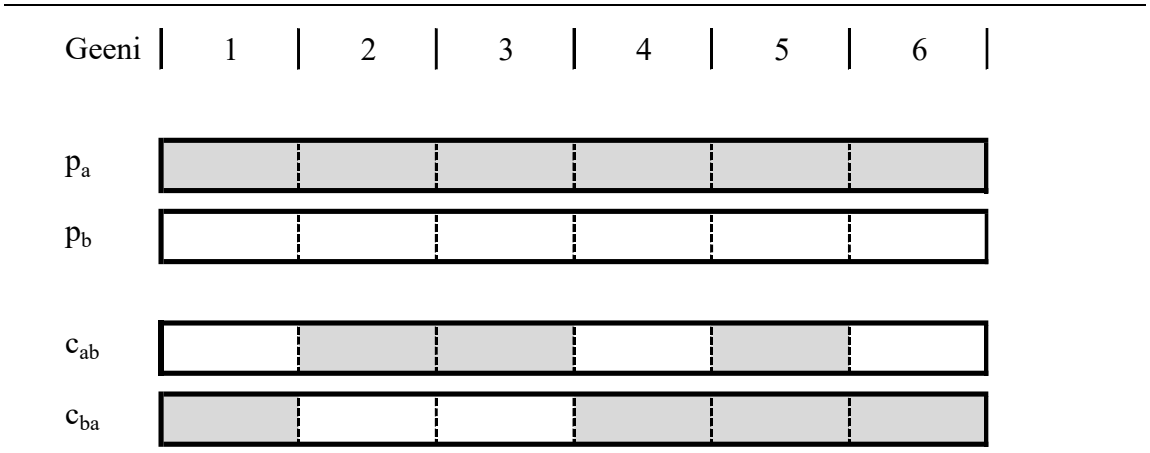
Kuva 5. Yhden pisteen risteytys, kun $x = 2$ ja $l = 6$.

Monen pisteen risteytys muistuttaa muutoin yhden pisteen risteytystä, mutta siinä valitaan useita leikkauspisteitä $x_1, \dots, x_n$, missä $1 < x_1 < \cdots < x_n < l$, joiden perusteella risteytys tehdään [Lim *et al.* 2017]. Kumpikin jälkeläinen perii tällöin joka toisen leikkauspisteiden erottaman jakson toiselta vanhemmalta ja joka toisen jakson toiselta vanhemmalta siten, että geenit peritään aina vastakkaisilta vanhemmilta, kuten kuvassa 6 on esitetty.



Kuva 6. Monen pisteen risteytys kun $n = 2,\ x_1 = 2, x_2 = 5$ ja $l = 6$.

Yhtenäisessä risteytyksessä ei valita lainkaan leikkauspisteitä, vaan jokaisella geenillä on yhtäläinen todennäköisyys tulla valituksi kummalta tahansa vanhemmalta [Lim *et al.* 2017]. Jälkeläiset perivät aina vanhempiensa vastakkaiset geenit. Kuvassa 7 näkyy, miten yhtenäinen risteytys voisi esimerkiksi tapahtua, kun $l = 6$.



Kuva 7. Yhtenäinen risteytys kun $l = 6$.

Risteytyksessä syntyneille jälkeläisille suoritetaan mutaatio-operaatio, joka muuntaa satunnaisesti pientä osaa kromosomin geeneistä. Mutaation tarkoituksena on välttää diversiteetin vähenemisestä seuraavaa populaation ominaisuuksien ennenaikaista suppenemista. Mikäli mutaatiota ei tehdä, katoaa populaatiosta sitä mukaa ominaisuuksia, kun yksikään niitä kantava ratkaisuehdokas ei tule valituksi valintaoperaatiossa heikon sopivuuden takia. Lisäksi satunnaisesti luodussa aloituspopulaatiossa ei välttämättä alun perinkään esiinny kaikkia mahdollisia ominaisuuksia. Koska jokin tällä tavoin kadonnut ominaisuus voi tuottaa korkean sopivuuden jossakin ominaisuuksien yhdistelmässä, on tärkeää, että kadonneita ominaisuuksia saadaan takaisin populaation mutaation avulla. [Gen & Lin 2007]. Mutaatiolla voidaan siis taata evoluution jatkuminen estämällä liian yhtenäisen populaation muodostuminen [Holland 1992]. Binäärimerkkijonossa mutaatio voi tarkoittaa yksinkertaisesti joidenkin bittien kääntämistä [Holland 1992].

Risteytys- ja mutaatio-operaatioita eri tilanteisiin ja koodauksiin on esitetty lukemattomia erilaisia. Proseduraalista sisällöntuotantoa käsittelevissä artikkeleissa on kuitenkin harvoin otettu kantaa, mitä risteytys- ja mutaatio-operaatiota kulloinkin on käytetty. Liapis ja muut [2011] kertovat käyttäneensä karttaluonnosten tuottamiseen kahden pisteen risteytystä ja 1%:n mahdollisuutta mutatoitua, jolloin 5-20% karttaluonnoksen laatoista muuttuu.

3.6 Yhteenveto

Geneettisiä algoritmeja voidaan käyttää ratkaisemaan hyvin monenlaisia ongelmia. Jotta geneettistä algoritmia voidaan käyttää jonkin tietyn ongelman ratkaisemiseen, ongelman ratkaisuehdokkaat tulee kyetä koodaamaan sellaisiksi kromosomeiksi, joiden sopivuutta voidaan mitata sopivuusfunktiolla, ja jotka voidaan risteyttää keskenään jollakin risteytysoperaatiolla. Geneettisen algoritmin tärkeimpiä toteutusyksityiskohtia ovat siis ratkaisuehdokkaan koodaus kromosomiksi, hyvän sopivuusfunktion kehittäminen sekä risteytys- ja mutaatio-operaatioiden valinta.

Yksinkertaisimmillaan kromosomin koodaus voi olla binäärikoodaus tai reaali- tai kokonaislukukoodaus. Tällaisten yksinkertaisten jonojen tapauksessa voidaan käyttää esimerkiksi yhden pisteen risteytystä, monen pisteen risteytystä tai yhtenäistä risteytystä jälkeläisten tuottamiseen. Sisältöä kuvaavan kromosomin sopivuutta voidaan mitata joko suoraan sen ominaisuuksien perusteella, interaktiivisesti peluuttamalla sen kuvaama ratkaisuehdokas oikealla ihmispelaajalla tai virtuaalisesti pelaamista simuloimalla.

Seuraavan sukupolven vanhempien valinta perustuu aina nykyisen sukupolven kromosomien sopivuusarvoihin. Rulettivalinnassa kunkin kromosomin todennäköisuus tulla valituksi on suoraan suhteessa sen saamaan sopivuusarvoon. Sijoitusvalinnassa vain kromosomien saamien sopivuusarvojen järjestyksellä on merkitystä. Turnausvalinnassa valitaan satunnaisesti eräänlaisia turnauslohkoja, joista aina korkeimman sopivuuden kromosomi valitaan. Oletusarvoisesti sama kromosomi voi tulla valituksi useita kertoja ja näin risteytyä jopa itsensä kanssa, jolloin se säilyy ennallaan.

Populaation geneettinen diversiteetti vähenee jatkuvasti vain vahvoja yksilöitä suosivan valintaoperaation seurauksena, joten mutaatiota käytetään tuomaan kadonneita ominaisuuksia takaisin populaatioon. On myös mahdollista, että parhaat ratkaisuehdokkaat menetetään sukupolvenvaihdoksessa geneettisten menetelmien satunnaisen luonteen vuoksi. Elitismiä voidaan käyttää takaamaan, että korkeimman sopivuusarvon ratkaisuehdokkaat säilyvät aina muuttumattomina seuraavaan sukupolveen.

Seuraavassa luvussa tarkastelen, miten geneettisiä algoritmeja voisi hyödyntää ympäristöön sopeutuvien fiktiivisten eläinlajien generoimisessa. Esitän myös muita, aikaisemmissa peleissä käytettyjä menetelmiä ja perustelen, miksi juuri geneettiset algoritmit sopivat tarkoitukseen hyvin.

4 Ympäristöön sopeutuvien eläinlajien generoiminen

Videopeliympäristöjen proseduraalista tuottamista on tutkittu laajasti, mutta näiden ympäristöjen asuttamista uskottavilla proseduraalisilla eläinlajeilla ei juurikaan ole käsitelty. Tässä luvussa tarkastelen, miten erilaisia fiktiivisiä eläinlajeja voidaan tuottaa proseduraalisesti soveltamalla proseduraalisen sisällöntuotannon menetelmiä.

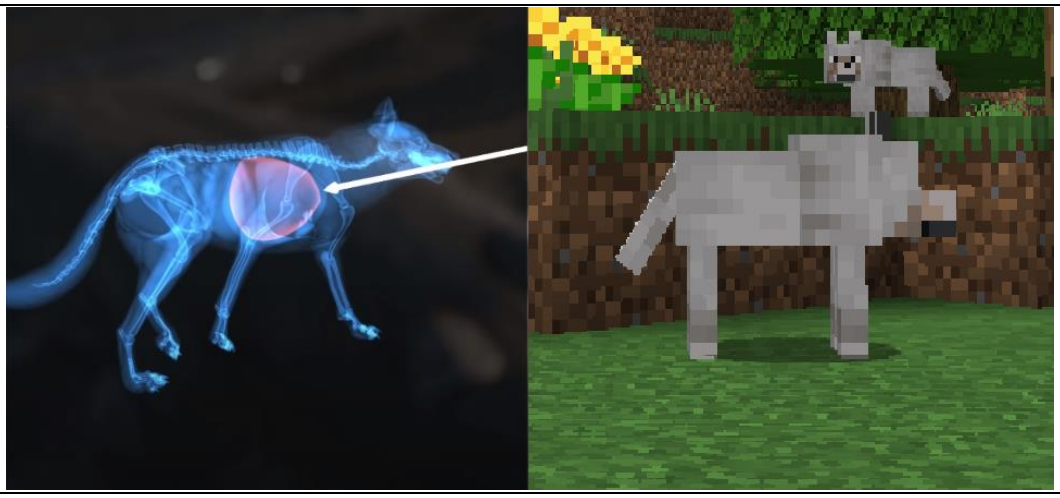
4.1 Eläimet videopeleissä

Termi *elain* (animal) määritellään Lexico-sanakirjassa seuraavasti:” *A living organism that feeds on organic matter, typically having specialized sense organs and nervous system and able to respond rapidly to stimuli*” [Oxford University Press 2019]. Vapaasti suomennettuna eläin on tämän määritelmän mukaan elävä organismi, joka käyttää ravinnokseen orgaanista ainetta, ja jolla tyypillisesti on erikoistuneita aistielimiä sekä keskushermosto, ja joka kykenee reagoimaan nopeasti ärsykkeisiin. Koska videopelit ovat vain yksinkertaistus tosimaailmasta, myös eläin videopelin palasena on

yksinkertaistus todellisesta eläimestä. Tässä tutkielmassa määrittelen eläimen videopelien kontekstissa sellaiseksi dynaamiseksi pelihahmoksi, joka ulkonäkönsä ja käytöksensä perusteella vaikuttaa siltä, että se aistii ympäristöään ja kuluttaa ravintoa, ja joka näennäisesti reagoi nopeasti ärsykkeisiin. Käytän eläimestä jatkossa myös sanaa *olento*.

Lexico-sanakirja määrittelee biologian termin *laji* (species) seuraavasti:" A *group of living organisms consisting of similar individuals capable of exchanging genes or interbreeding. The species is the principal natural taxonomic unit, ranking below a genus and denoted by a Latin binomial, e.g. Homo sapiens*" [Oxford University Press 2019]. Laji on vapaasti suomennettuna tämän määritelmän mukaan samankaltaisista yksilöistä koostuva joukko eläviä organismeja, jotka kykenevät vaihtamaan geenejä tai risteytymään keskenään. Yksinkertaistettuna eläinlaji voidaan mieltää joukkona samankaltaisia eläimiä. Videopelien tapauksessa eläinlaji voidaan myös ajatella eräänlaisena kaavakuvana, jonka pohjalta voidaan muodostaa peliympäristöön lajin ilmentymiä eli yksilöitä.

Videopelissä eläimiä esitetään vaihtelevilla tarkkuustasoilla pelin realismin ja lajityypin mukaan. Realistisimmissa metsästyspeleissä jopa eläinten luiden ja sisäelinten sijainnit on otettu huomioon, jotta osumat eri ruumiinosiin eri kaliiberin aseista voidaan mallintaa tarkasti [theHunter: Call of the Wild 2017]. Useimmissa peleissä eläimiä ei kuitenkin ole mallinnettu yhtä yksityiskohtaisesti, sillä näissä peleissä eläimet eivät ole yhtä olennaisessa osassa. Esimerkiksi kuvassa 8 näkyy vasemmalla susi theHunter: Call of the Wild -pelissä [2017] ja oikealla susi Minecraft-pelissä [2011]. Huolimatta suuresta erosta tarkkuustasossa, on kummassakin pelissä eläimen mallinnuksen tarkkuus sopiva pelin tarkoitukseen nähden.



Kuva 8. Vasemmalla läpivalaisu theHunter: Call of the Wild -pelin [2017] sudesta ja oikealla kaksi Minecraft -pelin [2011] sutta.

4.2 Eläinten generoiminen videopeleissä

Vaikka eläinten generoimista proseduraalisesti ei ole juuri tieteellisesti tutkittu, on olemassa joitakin pelejä, joissa sitä hyödynnetään. Tässä kohdassa tarkastelen kahta peliä, joissa erilaisia olentoja generoidaan proseduraalisesti. Nämä pelit ovat No Man's Sky [2016] ja Dwarf Fortress [2006]. Molemmissa peleissä olennot on kuvattu hyvin eri tavalla ja ne tuotetaan hyvin erilaisiin tarkoituksiin.

Pelissä No Man's Sky [2016] tuotetaan kokonaisia planeettoja proseduraalisesti. Näihin planeettoihin luodaan eloa asuttamalla ne proseduraalisilla eläin- ja kasvilajeilla. Esimerkiksi kuvan 9 eläinlajit ovat kaikki proseduraalisesti generoituja, ja ne esiintyvät proseduraalisesti tuotetussa ympäristössä.

No Man's Sky -pelin sisällöntuotantoprosessissa eläinlajille luodaan geometria, tekstuurit ja animaatiot yhdistämällä valmiiksi tuotettuja sisällönpalasia. Palaset valitaan kulkemalla sisällönpalasista koostuvaa puurakennetta juuresta lehteen, missä jokainen puun solmu on sisällönpalanen. Yksittäisen osan todennäköisyys tulla valituksi riippuu generoitavan lajin elinympäristöstä, jolloin valituksi tulee todennäköisemmin sellaisia osia, jotka sopivat ympäristöön. [Gregkwaste 2016].

Kuva 9. Proseduraalisesti generoituja eläinlajeja No Man's Sky -pelissä [No Man's Sky 2016].

Menetelmä on konstruktiivinen, joten tuotettujen lajien laatua ei jälkeenpäin mitata. Palaset ja todennäköisyydet on siis etukäteen huolella suunniteltu, jotta järjestelmä ei tuottaisi liian epäuskottavia tuloksia. No Man's Sky -pelissä tuotetut olennot ovat täysin valinnaista sisältöä, sillä pelaaja voi koska tahansa poistua eläinlajien läheisyydestä ja halutessaan siirtyä esimerkiksi kokonaan toiselle planeetalle. Tämän takia järjestelmän tuottamien eläinlajien laatua ei ole tarpeen valvoa erityisen tarkasti.

Merkkigrafiikalla toteutetussa Dwarf Fortress -pelissä [2006] generoidaan proseduraalisesti vihamielisiä hirviöitä. Koska pelissä ei ole lainkaan grafiikkaa,

kuvataan myös hirviöiden ulkonäkö ainoastaan tekstuaalisesti. Kuvassa 10 näkyy erään Dwarf Fortress -pelissä proseduraalisesti generoidun hirviön kuvaus, jossa myös hirviön nimi *Ozsit Genlathgakit*, on tuotettu proseduraalisesti. Olennon monitulkintainen kuvaus jättää paljon pelaajan mielikuvituksen varaan, mikä yhdistettynä olentojen myyttiseen luonteeseen, tarkoittaa, ettei Dwarf Fortress -pelissä ole olennaista, mihin ympäristöön olento visuaalisesti sopisi. Lisäksi Dwarf Fortress -pelissä jokainen hirviö on oma, ainutlaatuinen yksilönsä, eikä varsinaisesti minkään lajin edustaja.

Kuva 10. Dwarf Fortress -pelissä generoitu hirviö kuvaillaan tekstillä [Dwarf Fortress Wiki 2020].

Kuten No Man's Sky -pelissä, myös Dwarf Fortress -pelin olennot luodaan konstruktiivisesti valmiista, etukäteen suunnitelluista osista, jotka vaikuttavat ennalta määrätyllä tavalla lopullisen olennon ominaisuuksiin. Dwarf Fortress -pelissä tuotetut hirviöt ovat kuitenkin pakollista sisältöä, sillä ne ovat aggressiivisia ja tuhoavat jokaisen peliympäristössä esiintyvän pelihahmon, mikäli pelaaja ei aktiivisesti toimi hirviön pysäyttämiseksi. Useimmissa peleissä olisi tärkeää, että hirviöt tuotetaan sillä tavoin, että pelaajan on mahdollista päihittää ne. Dwarf Fortress -peli on kuitenkin tässä suhteessa siinä mielessä poikkeuksellinen, että kehittäjä on tietoisesti mahdollistanut lähes päihittämättömien ominaisuusyhdistelmien luomisen. Hirviö voi yhtä lailla olla vaaraton pölypallo kuin tulta syöksevä teräksinen jättiläinenkin, ja tämä on tietoinen ratkaisu pelin suunnittelussa.

Vaikka kummassakin edellä mainitussa pelissä tuotetaan hyvin erilaiset kuvaukset olennoista, ovat menetelmät toimintalogiikaltaan melko samankaltaiset. Molemmissa olennot luodaan etukäteen tuotetuista osista, jotka liitetään yhteen konstruktiivisesti. Valitut osat vaikuttavat olennon ulkonäköön ja ominaisuuksiin. Dwarf Fortress -pelissä generointiprosessissa ei lainkaan oteta huomioon olennon elinympäristöä, mutta No Man's Sky -pelissä ympäristö vaikuttaa osien todennäköisyyksiin tulla valituksi.

4.3 Geneettisen algoritmin käyttö sopeutumisen mallintamisessa

Eläimiä on generoitu peleissä proseduraalisesti aikaisemminkin, mutta niiden sopivuutta tiettyyn ympäristöön ei ole erityisesti huomioitu. Koska peliympäristöt voivat olla hyvin vaihtelevia ja monimutkaisia, on ympäristön vaikutusta haastavaa ottaa huomioon konstruktiivisilla menetelmillä. Konstruktiivisia generointimenetelmiä käytettäessä täytyy ympäristö ottaa huomioon jo menetelmää kehittäessä, sillä tuotetun eläinlajin sopivuutta ei evaluoida jälkikäteen. Menetelmän tulee siis tuottaa sopiva eläinlaji suoraan ensimmäisellä suorituskerralla. Tuota-ja-testaa -menetelmillä sen sijaan voidaan arvioida, onko eläinlaji sopiva ympäristöön. Mikäli eläinlaji ei ole sopiva, tuotetaan uusia lajeja, kunnes sopiva eläinlaji löytyy.

Luonnossa erilaiset eläinlajit sopeutuvat vaihteleviin ympäristöihin evoluution avulla. Luonnonvalinta pitää huolen, että huonosti ympäristössä selviytyvät yksilöt kuolevat pois ja vahvat säilyvät ja lisääntyvät. Geneettisten algoritmien toimintaperiaate perustuu tähän ilmiöön, joten tuota-ja-testaa -menetelmistä juuri geneettinen algoritmi on luonteva valinta sopeutuvien eläinlajien generoimiseen. Jotta ympäristöön sopeutuvia eläinlajeja voidaan tuottaa geneettisellä algoritmilla, tarvitaan eläinlajin koodaus kromosomiksi sekä sellainen sopivuusfunktio, joka mittaa kromosomin selviytymistä tietyssä ympäristössä.

Eläinlajin kromosomin on oltava sellainen, että sen pohjalta voidaan arvioida lajin sopivuutta tietyssä ympäristössä. Eri ympäristöissä menestyvät kromosomit tulee pystyä purkamaan takaisin toisistaan ulkoisesti poikkeaviksi eläinlajeiksi. Lajin ulkoisten ominaisuuksien tulee siis suoraan vaikuttaa sen selviytymiskykyyn. Lajin aistielimet vaikuttavat sen kykyyn aistia ympäristöään, joka voi auttaa ravinnoksi kelpaavien kasvien löytämisessä, saalistamisessa tai petojen välttelyssä. Yhteen ja samaan ympäristöön voidaan tuottaa useita sopivia, toisistaan poikkeavia kromosomeja, sillä samassa ympäristössä voi selviytyä erilaisten ominaisuusyhdistelmien avulla.

Vain eläinlajin ulkoinen olemus näkyy pelaajalle, joten esimerkiksi lajin keuhkojen tilavuutta tai aivojen poimuttuneisuutta on turha lähteä mallintamaan, vaikka ne todellisuudessa vaikuttaisivatkin sen kykyyn selviytyä. Sen sijaan pelin kannalta mielenkiintoisia ominaisuuksia voisivat olla esimerkiksi eläimen koko, liikkumistapa ja väritys. Vehreällä suoalueella pelaaja olettaa näkevänsä vihertäviä räpyläjalkaisia eläimiä, lumisessa ympäristössä taas valkoisia ja paksuturkkisia eläimiä. Aiemmin esitetyn eläimen määritelmän perusteella mallinnettavia ulkoisia ominaisuuksia ovat tämän aistielimet, kuten silmät ja korvat. Kyky reagoida nopeasti ärsykkeisiin voidaan myös mieltää siten, että eläimellä tulee olla jonkinlaisia välineitä liikkumiseen ja ympäristönsä manipuloimiseen. Erityisesti selkärankaisten tapauksessa näitä voisivat olla erilaiset raajat kuten jalat ja kädet.

Eläinlaji on proseduraalisen sisällöntuotannon näkökulmasta pelin palanen, joka toimii pelitilassa. Pelitila taas tässä tapauksessa on lajin elinympäristö. Sopivuusfunktion tulee olla sellainen, että eläimen ulkoiset piirteet vaikuttavat sen selviytymiseen. Näin tuotetut lajit näyttävät ulkoisesti sopivan ympäristöönsä. Olennainen tekijä lajin selviytymisen kannalta on myös ympäristön ekosysteemi, joka koostuu muista eliöpopulaatiosta sekä erilaisista elottomista kappaleista. Esimerkiksi alueella, jolla elää paljon petoja, on hyödyllistä joko kyetä taistelemaan petoja vastaan tai piiloutumaan.

4.4 Simulaatio sopivuuden mittana

Koska tarkoituksena on tuottaa sellaisia eläinlajeja, jotka ovat sopeutuneet tiettyyn ympäristöön, on mielekästä käyttää sopivuuden mittana lajin yksilöiden kykyä selviytyä kyseisessä ympäristössä. Selviytymiskykyä vaihtelevissa ja monimutkaisissa, muiden eläinlajien asuttamissa ekosysteemissä on kuitenkin hankalaa mitata luotettavasti suoralla sopivuusfunktiolla. Ympäristön ekosysteemin vaikutus lajin selviytymiseen voidaan paremmin huomioida käyttämällä simulaatiopohjaista sopivuusfunktiota, missä eläinlajin yksilöistä muodostetaan pelihahmoja peliympäristöön.

Suoraviivainen tapa mitata lajin selviytymiskykyä on selvittää, kuinka pitkään yksilöt selviytyvät simulaatioympäristössä. Koska simulaation kulku on aina olla jokseenkin sattumanvaraista, on satunnaisuuden vähentämiseksi järkevää luoda simulaatioon kerralla aina useampi lajin yksilö. Useamman yksilön simulaatiossa sopivuus voidaan laskea kaikkien yksilöiden elinikien summana. Vaikka yksi hyvin ympäristöön soveltuva yksilö menehtyisikin sattumalta huomattavasti odotettua aikaisemmin, voivat muut yksilöt selviytyä ominaisuuksiensa edellyttämällä tavalla, jolloin sopivuusarvo pysyy edelleen suhteellisen korkeana. Toisaalta jos ominaisuuksiensa puolesta huonosti sopeutuva yksilö sattumalta selviytyykin odotettua pidempään, kuolevat sen lajitoverit todennäköisesti sitä aikaisemmin, jolloin sopivuusarvo ei pääse kasvamaan suhteettoman korkeaksi.

Mikäli tuotettavat eläinlajit ovat pelissä pakollista sisältöä, asettaa tämä lisärajoitteita sopivuutta mittaavalle simulaatiolle. Esimerkiksi pelissä, jossa pelaajan on pakko taistella proseduraalisesti tuotettuja eläimiä vastaan, tulee varmistaa tuotettujen eläinlajien vaikeusasteen sopivuus. Yksi lähestymistapa olisi lisätä simulaatioon myös pelaajahahmoa simuloiva agentti, jonka pärjääminen otettaisiin huomioon sopivuusarvossa. Mikäli agentti ei pärjää taistelussa proseduraalisia eläimiä vastaan, on vaikeusaste liian korkea. Toisaalta mikäli taistelu ei tuota agentille merkittävää vahinkoa, on vaikeusaste liian helppo. Kummassakin tapauksessa eläinlajin sopivuus laskee.

5 Simulaatiopohjainen lajievoluutio -järjestelmän toteutus

Testatakseni geneettisen algoritmin käyttöä ympäristöön sopeutuvien eläinlajien tuottamisessa, toteutin ohjelman C#-kielellä Unity-pelimoottorissa. Kutsun ohjelmaa

nimellä *Simulaatiopohjainen lajievoluutio*, eli lyhyesti SPLE. Tavoitteenani oli kehittää järjestelmä, joka pystyy tuottamaan vaihteleviin videopeliympäristöihin sopivia eläinlajeja. Tuotettujen eläinlajien yksilöiden tulisi ulkoisesti näyttää siltä, että ne pystyvät selviytymään elinympäristössään.

Koska eläinlajin sopivuutta monimutkaisessa ekosysteemissä on vaikeaa, ellei jopa mahdotonta arvioida suoran sopivuusfunktion avulla, päädyin käyttämään simulaatiopohjaista sopivuusfunktiota. Näin järjestelmässä saadaan otettua paremmin huomioon muut eläinlajit ja lajien välinen interaktio.

Toteutuksessa eläinlajin sopivuus lasketaan sen mukaan, kuinka kauan sen yksilöt selviytyvät simulaatioympäristössä. Simulaatioympäristön säännöt vaihtelevat sen mukaan, minkälaiseen peliin eläinlajeja ollaan tuottamassa. Olennaista on se, että eläinlajien ominaisuudet vaikuttavat johdonmukaisesti sen kykyyn selviytyä ympäristössä kulloisillakin simulaatiosäännöillä. Tässä tutkielmassa käytän esimerkkinä yhdistelmää tyypillisistä roolipelien säännöistä. Yksilöitä voi olla simulaatioympäristössä useita, jolloin sopivuus lasketaan niiden yhteenlasketusta selviytymisajasta.

SPLE-järjestelmän eläinlajin genotyyppi on yksinkertaistus neliraajaisesta nisäkkäästä. Lajin genotyypin perusteella luodaan simulaatioympäristöön lajin yksilöitä kuvaavia pelihahmoja eli fenotyyppejä, joiden avulla lajin sopivuutta mitataan. Koska erilaisten pelien tapauksessa voidaan käyttää erilaisia simulaatiosääntöjä, en tässä tutkielmassa käsittele perinpohjaisesti kaikkia tekemiäni ratkaisuja ja käyttämiäni kaavoja.

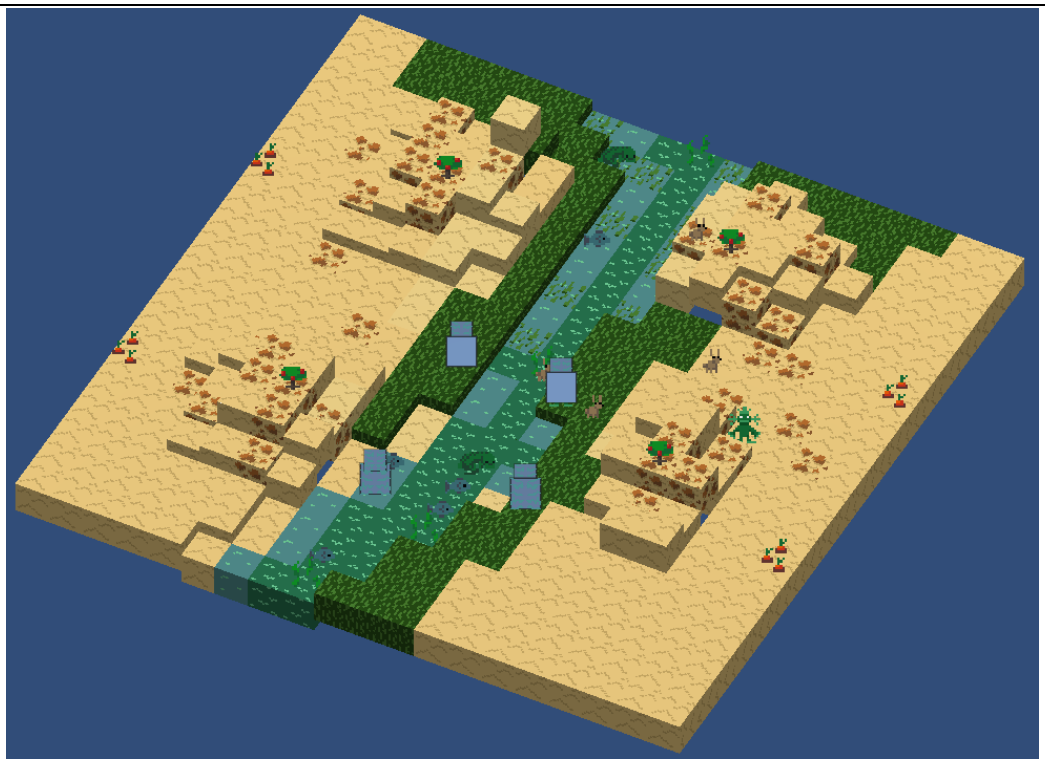
5.1 Järjestelmän yleiskuva

SPLE-järjestelmä tuottaa abstrakteja kuvauksia fiktiivisistä eläinlajeista. Eläinlajien sopivuus ympäristöönsä pyritään maksimoimaan käyttämällä geneettistä algoritmia, jonka sopivuusfunktio mittaa eläinlajin selviytymistä ympäristössä. Mitä pidempään eläinlajin yksilöt pysyvät hengissä, sen parempi on lajin saama sopivuusarvo.

Eläinlajin kromosomi on esitetty listana kokonaislukuarvoisia ominaisuuksia. Nämä ominaisuudet voivat olla joko laadullisia tai määrällisiä. Ominaisuudet vaikuttavat lukuisilla tavoilla lajien selviytymiseen erilaisissa ympäristöissä.

Ympäristö on esitetty ruudukkona, joka pitää sisällään erilaisia maastotyyppejä, eläimiä edustavia pelihahmoja sekä ravinnonlähteenä toimivia kasveja. Jokaisella yksittäisellä elinympäristön ruudulla on aina yksi maastotyyppi, korkeus, lämpötila ja siihen osuvan valon määrä. Lisäksi yksittäisessä ruudussa voi olla yksi tai useampi pelihahmo sekä yksi tai useampi kasvi. Ympäristöä kuvaava tiedosto voidaan toteuttaa Tiled-karttaeditorilla [Tiled 2009]. Kuvassa 11 näkyy eräs ympäristö SPLE-järjestelmässä. Tässä ympäristössä esiintyy useita maastotyyppejä, kuten hiekkaa, vettä ja korkeaa heinää, sekä eläinlajeja, kuten jäniksiä ja krokotiileja.

Lajin selviytyminen ympäristössä mitataan simulaatiolla. Lajin yksilöistä muodostetaan ympäristöön pelihahmoja, jotka ovat vuorovaikutuksessa ravinnonlähteiden ja muiden pelihahmojen kanssa. Pelihahmot voivat suorittaa erilaisia toimintoja, jotka kuluttavat aikaa. Toiminnon suorittamisen jälkeen pelihahmon täytyy odottaa toimintoon kuluneen ajan verran toimiakseen uudelleen. Hahmot voivat liikkua ympäristön ruudusta toiseen, jolloin liikkumiseen kuluva aika riippuu maastonmuodoista ja hahmon liikkumisominaisuuksista. Hahmot voivat myös tarkkailla ympäristöään, kuluttaa samassa ruudussa olevaa ravintoa tai taistella samassa ruudussa olevan pelihahmon kanssa.



Kuva 11. Ympäristö SPLE-järjestelmässä, jossa on joki hiekka-aavikon keskellä.

Simulaatiota jatketaan, kunnes kaikki ratkaisuehdokasta esittävät pelihahmot kuolevat. Esimerkeissä taistelu mallinnetaan Dungeons & Dragons -roolipelin [D&D Basic Rules 2014] sääntöjä soveltaen, eli pelihahmojen kestävyyttä mallinnetaan *elämäpisteillä* (hitpoints). Pelihahmot voivat menettää elämäpisteitä joko taistelussa muiden pelihahmojen hyökätessä, tai nääntyessään nälkään. Kun pelihahmon elämäpisteet loppuvat, se kuolee.

Jokaiselle populaation kromosomille suoritetaan yksi kerrallaan simulaatiopohjainen sopivuusfunktio. Kun kaikkien kromosomien sopivuus on mitattu, valitaan seuraavan sukupolven vanhemmat rulettivalinnalla. Seuraavaan sukupolveen voidaan ottaa mukaan myös nykyisen sukupolven parhaita yksilöitä elitismillä, jotta parhaimmat ratkaisuehdokkaat säilyvät sukupolvesta toiseen.

Vanhempien parit risteytetään keskenään yhtenäisellä risteytyksellä, jolloin kukin geeni valitaan satunnaisesti jommaltakummalta vanhemmalta. Osalle jälkeläisistä suoritetaan mutaatio-operaatio, joka muuttaa yhden geenin arvoa satunnaisesti. Todennäköisyys mutaation suorittamiseen on 50%, ja valinta tapahtuu muodostamalla satunnaisluku $n \in [0, 1]$. Mutaatio suoritetaan mikäli $n > 0{,}5$. Muussa tapauksessa mutaatiota ei suoriteta.

Ohjelman suoritus lopetetaan vasta käyttäjän antaessa lopetuskomennon, jolloin palautetaan parhaan sopivuusarvon saanut eläinlaji. Mitään tiettyä sopivuusarvorajaa lopetusehdoksi on vaikea muotoilla, sillä kromosomien sopivuuksien arvoalue riippuu paljon ympäristön ja simulaation sääntöjen yhdistelmästä.

5.2 Eläinlajin kromosomiesitys

Yksittäisen eläinlajin kromosomi on esitetty yksiulotteisena listana kokonaislukuarvon sisältäviä geenejä. Valitsin tämän esitysmuodon, sillä se on binäärikoodausta hallittavampi, mutta samalla se mahdollistaa useiden yleisesti käytettyjen geneettisten operaatioiden käytön. Tämän esityksen kanssa voidaan käyttää kaikkia luvussa 3 esiteltyjä risteytysoperaatioita, kuten yhden tai usean pisteen risteytystä, sekä yhtenäistä risteytystä.

Eläinlajin kromosomin jokainen geeni kuvaa jotakin lajin ominaisuutta, ja se voi saada arvoja vain ennalta määrätyltä arvoalueelta. Koska geenillä on aina rajattu lineaarinen arvoalue, on myös mutaatio-operaation toteuttaminen yksinkertaista, sillä muutettu arvo voidaan pakottaa pysymään arvoalueen sisällä. Yksittäisen geenin arvoalue määritellään ulkoisessa JSON-tiedostossa, ja jokaiselle geenille voidaan asettaa erilliset, toisistaan poikkeavat arvoalueet. Näin järjestelmä tuottaa aina mahdollisia ratkaisuja.

Osa geeneistä on määrällisiä ja osa laadullisia. Kaikki valitsemani geenit ja niiden arvoalueet on esitetty taulukossa 1. Esimerkiksi eläinlajin raajan pituuden määrittävän geenin arvoalueeksi olen asettanut {0,...,100}, missä 0 tarkoittaa hyvin lyhyttä raajaa ja 100 hyvin pitkää raajaa. Raajan muotoa kuvaava geeni on laadullinen, ja se voi saada arvot väliltä {0,…,4}, missä 0 tarkoittaa varpaita, 1 kaviota, 2 räpylää ja niin edelleen. Geenin arvojen tulkinta vaihtelee sen kuvaaman ominaisuuden ja käytettyjen simulaatiosääntöjen mukaan. Tällä tavoin monimutkaisia rakenteita voidaan yksinkertaistaa erilaisiin luokkiin, samalla kun esimerkiksi lajin koko voidaan ilmaista lineaarisesti. Koska molemmat ominaisuustyypit on esitetty pohjimmiltaan kokonaislukuina, toimivat risteytys ja mutaatio molemmille ominaisuustyypeille samalla tavalla.

5.3 Pelihahmon tuottaminen kromosomin pohjalta

Käyttämilläni simulaatiosäännöillä kromosomin geenit vaikuttavat pelihahmojen ominaisuuksiin hyvin monilla eri tavoilla. Tässä kohdassa selitän pintapuolisesti,

minkälaisiin pelihahmon ominaisuuksiin mikäkin geeni vaikuttaa ja millä tavalla. Tavoitteenani oli muodostaa pelihahmo kromosomin pohjalta siten, että geenit vaikuttavat pelihahmon ominaisuuksiin johdonmukaisesti. Muunnos kromosomista pelihahmoksi perustuu vain hyvin löyhästi todelliseen biologiaan.

Geeni	Minimi	Maksimi	Kuvaus
Ruumiin koko	0	100	%
Pään koko	0	100	%
Raajan pituus	0	100	%
Raajan paksuus	0	100	%
Silmien lukumäärä	0	2	kpl
Silmien koko	0	100	%
Raajojen tyyppi	0	4	Varpaat, kaviot, räpylät, kynnet, evät
Suun tyyppi	0	4	Kuono, nokka, torahampaat, kärsä
Korvan tyyppi	0	4	Ei korvia, pienet korvat, suunnatut korvat, suuret korvat
Peitteen tyyppi	0	3	Iho, turkki, suomut, panssarisuomut
Pohjavärityksen punainen	0	255	
Pohjavärityksen vihreä	0	255	
Pohjavärityksen sininen	0	255	
Kuvioinnin punainen	0	255	
Kuvioinnin vihreä	0	255	
Kuvioinnin sininen	0	255	

Taulukko 1. SPLE-järjestelmän geenit ja niiden arvoalueet.

Ruumiin koko vaikuttaa eläimen kestävyyteen, nopeuteen, hyökkäysvoimaan ja ravinnonkulutukseen. Suuremmilla eläimillä on enemmän elämäpisteitä, ne ovat keskimäärin nopeampia ja ne tuottavat enemmän vahinkoa taistelussa. Vastapainona näille eduille, suuremmat eläimet ovat helpommin havaittavia näköaistilla, huonompia väistelemään ja ne kuluttavat enemmän ravintoa. Suuri koko on edullinen ominaisuus lähinnä sellaisessa ympäristössä, jossa taistelu on välttämätöntä selviytymisen kannalta. Mikäli ympäristössä ei ole tarvetta taistelulle, on pieni koko edullisempi ominaisuus pienemmän ravinnonkulutuksen takia.

Suuri pään koko lisää lajin korkeutta ja purentavoimaa, mutta myös ravinnonkulutusta. Silmien koko ja lukumäärä vaikuttavat olennon näkökykyyn. Kokonaan silmättömillä eläinlajeilla silmien kokoa ei lainkaan huomioida. Näkökyky on

jaettu kahteen osaan, valonäköön ja pimeänäköön, joita nimensä mukaisesti käytetään valaistukseltaan erilaisissa ympäristöissä. Jokainen silmä lisää lajin ravinnonkulutusta. Kokonaan silmätön olento kuluttaa vähemmän ravintoa, mutta joutuu turvautumaan muihin aisteihin ravinnon etsinnässä ja vaarojen välttelyssä. Kaksi silmää taas mahdollista stereonäön, joka lisää näköetäisyyttä. Suuret silmät näkevät pidemmälle ja paremmin pimeässä, mutta ne myös lisäävät eläinlajin ravinnonkulutusta ja mahdollisuutta loukkaantua vakavasti taistelussa. Näin suuret silmät ovat hyödyllisiä erityisesti pimeissä ympäristöissä, mutta altistavat eläinlajin suurempaan vaaraan.

Raajojen vaikutus eläinlajin liikkumiskykyyn, ja sitä kautta sopivuuteen luonnonvalinnassa, riippuu tosimaailmassa lukuisista tekijöistä kuten huippunopeudesta, kiihtyvyydestä, kestävyydestä ja tasapainosta [McNeill 2003]. Vaikka kaikkien liikkumiskykyyn liittyvien seikkojen huomioiminen olisikin kiinnostavaa jossakin toisessa kontekstissa, ei se ole mielekästä videopelien tapauksessa. Olen tämän takia huomioinut vain huippunopeuden ja kyvyn liikkua haastavissa ympäristöissä.

Raajojen pituus lisää huippunopeutta, potkuvoimaa ja kykyä liikkua haastavissa maastoissa, mutta samalla myös ravinnonkulutusta ja näkyvyyttä. Pitkäraajaiset lajit ovat keskimäärin nopeampia kuin lyhytjalkaiset, mutta vaativat enemmän ravintoa. Raajojen paksuus taas lisää paitsi potkujen voimaa ja kykyä liikkua tietynlaisissa ympäristöissä, myös lajin elämäpisteitä sekä kylmänsietokykyä. Paksut jalat kasvattavat kuitenkin samalla ravinnonkulutusta. Koska paksut jalat auttavat sietämään kylmää, kylmissä olosuhteissa on todennäköisempää selviytyä paksummilla jaloilla.

Eläinlajin suu vaikuttaa sen purentaan ja ravinnonkulutukseen. Kuono on tasapainoinen suutyyppi, joka pystyy syömään sekä lihaa että kasveja yhtä tehokkaasti. Nokallinen eläinlaji kykenee syömään lihaa tehokkaammin kuin kasveja. Nokka on myös tehokas ase panssaroituja pelihahmoja vastaan. Torahampailla ei pysty lainkaan syömään kasveja, mutta niiden purenta on voimakas ja niillä on helppo syödä lihaa. Pitkä kärsä taas on hyvä kaivamaan juuria maasta ja poimimaan hedelmiä puista, mutta huono taistelussa ja lihan syömisessä.

Lajin korvien tyyppi vaikuttaa sen kuuloon, lämmönsäätelyyn ja näkyvyyteen. Ilman korvia laji ei voi käyttää kuuloaistia ympäristönsä aistimiseen, mutta tarvitsee vähemmän ravintoa. Pienet korvat kestävät hyvin kylmää, mutta ovat kuuloltaan muita heikommat. Suunnatut korvat takaavat hyvän kuulon, mutta vaativat eniten ravintoa. Suuret korvat auttavat eläinlajia viilentämään itseään kuumissa ympäristöissä ja antavat hyvän kuulon, mutta samalla tekevät lajista helpomman maalin taistelussa.

Eläimen peite suojaa vahingolta ja kylmältä. Karvaton iho on ravinnonkulutuksen kannalta edullisin peitetyyppi ja se kestää hyvin kuumuutta, mutta se ei tarjoa juuri muita etuja. Paksu turkki suojaa eläinlajia kylmältä ympäristöltä ja muiden pelihahmojen hyökkäyksiltä, mutta on kömpelö ja hidastaa liikkumista maalla ja vedessä. Pienet suomut

nopeuttavat liikkumista vedessä, kun taas panssarisuomut hidastavat liikkumista maalla ja vedessä, mutta tarjoavat parhaimman suojan hyökkäyksiä vastaan.

Eläinlajin väritys on jaettu kahteen osaan: pohjaväriin ja kuvioinnin väriin. Värit vaikuttavat lajin maastoutumiseen ympäristöönsä ja sitä kautta selviytymiseen. Kromosomissa värit on jaettu niiden RGB-komponentteihin, eli punaiseen, vihreään ja siniseen. Mitä paremmin väri täsmää maastotyypin väriin, sen vaikeampaa eläintä on havaita näköaistilla sen ollessa kyseisen maastotyypin ruudussa.

SPLE-järjestelmän toteutuksessa päädyin yksinkertaistamaan tuotettavia eläimiä niin, että ne voivat olla vain neliraajaisia, ja niillä on aina pää. Lisäksi niillä on aina yksi suu, yhdet korvat ja 0-2 silmää. Nämä rajoitteet yksinkertaistavat jonkin verran kromosomin selviytymisen simulointia ympäristössä ja samalla tuotetut lajit ovat uskottavampia, sillä useimmat todelliset eläinlajit kuuluvat näiden rajoitusten piiriin. Rajoitusten takia järjestelmän ilmaisuvoima kuitenkin heikkenee, sillä se ei pysty tuottamaan sellaisia eläinlajeja, jotka eivät noudata tätä ennalta määrättyä rakennetta. Esimerkiksi johonkin peliin saatettaisiin haluta pystyä tuottamaan kymmenpäisiä ja satasilmäisiä siivekkäitä olentoja, mikä on valitsemallani kromosomiesityksellä mahdotonta.

Koska kromosomin sopivuus selvitetään aina pelisimulaation avulla, vie se tässä toteutuksessa selvästi eniten suoritustehoa. Tällöin kromosomiesityksen esittämisen tehokkuus ei ole yhtä olennainen tekijä koko ohjelman suorituskyvyn kannalta kuin suoraa sopivuusfunktiota käytettäessä. Suorituskykyä tärkeämpää on pystyä tuottamaan sellaisia kromosomeja, joiden avulla voidaan luotettavasti mitata lajin selviytymistä simulaatiossa.

Esittämäni kromosomiesitys on yleisluontoinen, eikä sitä ole varsinaisesti suunniteltu mihinkään tiettyyn videopeliin tai peligenreen. Se voitaisiin muuntaa sisällönkappaleeksi eri tavoin kunkin pelin tarpeiden mukaan. Esimerkiksi Dwarf Fortress -pelin [2006] kaltaisessa merkkipohjaisessa pelissä kromosomia voitaisiin käyttää jopa sellaisenaan, jolloin koodaus olisi suora. Visuaalisemmassa pelissä täytyisi kromosomin pohjalta rakentaa graafinen esitys, kuten 2D-kuva tai 3D-malli. Monimutkaisemmissa peleissä kromosomiin voisi lisätä myös uusia geenejä ilmaisuvoiman kasvattamiseksi. Tässä tutkielmassa en kuitenkaan tämän tarkemmin tarkastele, miten kromosomi saataisiin muutettua valmiiksi sisällönpalasiksi erilaisiin peleihin.

5.4 Elinympäristö

SPLE-järjestelmässä elinympäristöt on kuvattu kaksiulotteisena ruudukkona, jossa jokaiseen ruudukon ruutuun on yhdistetty erilaisia tietoja. Kaikilla ruuduilla on aina *maastotyyppi*, *lämpötila*, *valon määrä* sekä *korkeus*. Lämpötila, valon määrä ja korkeus ovat kokonaislukutyyppisiä muuttujia, mutta maastotyyppi on oma tietorakenteensa.

Ympäristöt on tallennettu JSON-tiedostoihin ja niitä voidaan toteuttaa Tiled-karttaeditorilla [2009].

Maastotyypit kuvaavat ruudun vaikeakulkuisuutta ja piiloutumismahdollisuuksia, ja niitä voidaan määritellä ulkoisiin JSON-tiedostoihin. Eläinlajin kykyyn liikkua vaikeakulkuisissa maastoissa vaikuttaa niiden raajan muoto sekä pituus ja paksuus. Maastotyyppien haastavuus on jaettu SPLE-järjestelmässä kahteen luokkaan, ja lisäksi veden syvyys ja korkeuserot vaikuttavat haastavuuteen.

Maastotyypin *epätasaisuus* kuvaa sen peittämien kivien ja kasvien tuomaa haastetta maastotyypissä liikkumiseen. Esimerkiksi louhikko on maastotyyppinä huomattavasti epätasaisempi kuin tasanko, jolloin eläinlajilta vaaditaan enemmän epätasaisessa maastossa liikkumisen taitoa tällaisessa maastotyypissä liikkumiseen. Pitkät jalat auttavat eläinlajia liikkumaan epätasaisessa maastossa.

Upottavuus kuvaa maastotyypin pinnan pehmeyttä ja upottavuutta, joka epätasaisuuden tapaan hidastaa eläinlajien liikkumista. Esimerkiksi muta on hyvin upottava maastotyyppi, mutta kallio ei ole lainkaan upottava. Upottavassa maastossa räpylä on huomattavasti nopeampi raajatyyppi kuin esimerkiksi kavio. Paksut jalat auttavat eläinlajia liikkumaan upottavassa maastossa.

Mikäli maastotyyppi pitää sisällään eläinlajin korkeutta syvempää vettä, vaatii tässä maastotyypissä liikkuminen lajilta uimataitoa. Kaikki lajit ovat SPLE-järjestelmässä uimataitoisia, mutta uimisnopeus riippuu pääosin raajojen muodosta. Evät ja räpylät mahdollistavat kaikista nopeimman uimisnopeuden, kun taas kaviot ovat kaikista hitain vaihtoehto.

Ruutuun osuvan valon määrä määrittää, kuinka helppoa pelihahmojen on aistia ruudussa sijaitsevia pelihahmoja ja kasveja näkökyvyn mukaan. Yleisesti ottaen pimeässä on vaikeampi nähdä kuin valoisassa. Joillakin eläinlajeilla on kuitenkin parempi pimeänäkö kuin toisilla.

Lämpötila vaikuttaa siihen, kuinka paljon aikaa pelihahmolla kuluu toimintoihin. Mikäli lämpötila on epäoptimaalinen eläinlajille, on toimintojen suorittaminen vaativampaa ja siten siihen kuluu enemmän aikaa. Tästä seuraa se, että eläinlajit eivät selviydy yhtä hyvin liian kuumassa tai kylmässä ympäristössä.

5.5 Aistit ja muisti

Pelihahmot käyttävät seuraavia aisteja ympäristönsä tarkkailuun: näköaisti, kuuloaisti ja hajuaisti. Näköaisti koostuu kahdesta arvosta, valonäöstä ja pimeänäöstä. Mitä enemmän valoa kulloinkin tarkasteltavassa ruudussa on, sitä lähempänä näkyvyys on valonäön arvoa. Vähässä valossa taas arvo on lähempänä pimeänäön arvoa. Kuulo- ja hajuaistit perustavat vain yhteen arvoon.

Pelihahmojen aistien toiminta on toteutettu leveyshaulla. Haku käy läpi avoimien ruutujen jonoa, kunnes kaikki avoimet ruudut on käyty läpi. Jokainen käsitelty ruutu

lisätään aina suljettujen ruutujen joukkoon. Ruutujen läpikäyminen aloitetaan pelihahmon sen hetkisestä ruudusta, eli avoimien ruutujen jonon ainoa alkio on alkutilanteessa hahmon silloinen sijainti. Tarkasteltavan ruudun kaikki suorat naapuriruudut lisätään avoimien ruutujen jonoon, mikäli naapuriruudun etäisyys pelihahmosta on lyhyempi kuin pelihahmon aistien maksimietäisyys ja ruutu ei ole ennestään avoimien tai suljettujen ruutujen joukossa.

Jokaisen tarkastellun ruudun kohdalla lasketaan ruudun näkyvyys valonmäärän ja korkeuseron perusteella. Näkyvyys v lasketaan lineaarisella interpolaatiolla pimeänäön v_{dark} ja valonäön v_{light} väliltä tarkasteltavan ruudun valonmäärän l perusteella

$$v = v_{dark} + (v_{light} - v_{dark}) * l,$$

missä $0 \leq l \leq 1$. Kun valoa ei ole lainkaan, näkyvyys on pimeänäön arvo. Jos taas ruutu on täysin valaistu, näkyvyys on valonäön arvo. Jos valoa on jonkin verran, näkyvyys on valonäön ja pimeänäön väliltä. Useimmilla lajeilla valonäkö on parempi kuin pimeänäkö, jolloin ne näkevät paremmin valoisissa ympäristöissä. Näkyvyys paranee entisestään, mitä korkeammalla pelihahmo on suhteessa tarkasteltavaan ruutuun. Toisaalta näkyvyys heikkenee, mikäli pelihahmo on tarkasteltavan kohteen alapuolella. Korkeusero vaikuttaa myös hajuaistiin, mutta käänteisesti. Pelihahmo haistaa herkemmin muut pelihahmot, jotka ovat sitä korkeammalla.

Kun näkyvyys ja hajuaisti on laskettu, käydään läpi jokainen pelihahmo ja kasviresurssi, jotka ovat tarkasteltavassa ruudussa ja lasketaan, havaitseeko tarkkaileva pelihahmo niitä vai ei. Todennäköisyys havaita tarkasteltava kohde perustuu yhdistelmään näkö-, haju- ja kuuloaisteja. Tähän vaikuttaa myös kohteen maastotyypin väritys ja äänekkyys.

5.6 Polunetsintä

Olennot hakeutuvat kohti ravinnonlähteitä vältellen samalla mahdollisia uhkia. Polunetsintään käytetään SPLE-järjestelmässä A*-polunetsintäalgoritmia. Valitsin A*-polunetsintäalgoritmin, sillä se tuottaa pätevällä heuristiikalla aina optimaalisen tuloksen [Hart *et al.* 1968]. Algoritmi on myös yksinkertainen toteuttaa ja nopea ajaa.

Koska elinympäristö on esitetty ruudukkona, voidaan jokaista ruudukon solua käsitellä polunetsintäalgoritmin näkökulmasta polunetsintägraafin solmuna. Jokaisesta ruudukon ruudusta kulkee kaari sen suoriin naapureihin, mutta ei kulmittaisiin naapureihin. Graafin kaaret ovat suuntaamattomia, joten jos solmusta A pääsee solmuun B, pääsee myös solmusta B aina solmuun A. Jokainen pelihahmo tuntee aina jokaisen ruudun maastotyypin, lämpötilan ja valon määrän riippumatta siitä, onko hahmo koskaan nähnyt kyseistä ruutua. Näin hahmo osaa suoraan navigoida sellaistenkin ruutujen kautta, joissa se ei ole koskaan käynyt.

Jotta pelihahmon aisteilla olisi enemmän merkitystä yksilön selviytymisen kannalta, perustuu uhkaavien pelihahmojen välttely väliaikaisiin, aistihavaintoihin perustuviin

muistoihin. Simulaatiossa eläimet voivat toimia sekä ravinnonlähteenä että uhkana. Potentiaalisesti vakavasti olentoa taistelussa vahingoittavat olennot ovat uhkaavampia kuin sellaiset olennot, jotka aiheuttavat vain vähän vahinkoa. Jokaisen pelimaailman ruudun tila muuttuu jatkuvasti näiden muistojen perusteella. Kun muisto vanhenee, vähenee myös sen relevanssi polunetsinnässä. Muisto hetki sitten havaitusta uhasta siis vaikuttaa voimakkaammin pelihahmon polunetsintään kuin aikaa sitten havaittu uhka. Kun polunetsintäalgoritmi arvioi etäisyyttä kahden ruudun välillä, etäisyyteen lisätään myös pelihahmon muistoihin tallennettu ruudun riskiarvo, jolloin pelihahmo pyrkii väistämään potentiaalisesti vaarallisia alueita. Muistilla pyritään tekemään aistimisesta hyödyllisempää lajin selviytymisen kannalta. Mitä useammin ja pidemmän matkan päästä pelihahmo pystyy tekemään havaintoja uhkaavista pelihahmoista ja mahdollisista ravinnonlähteistä, sitä paremmin se pystyy välttelemään uhkia ja löytämään ravintoa.

6 Simulaatiopohjainen lajievoluutio -järjestelmän tuloksia eri ympäristöissä

Testasin SPLE-järjestelmän toimintaa mallintamalla erilaisia kuvitteellisia elinympäristöjä ja ajamalla ohjelman näissä ympäristöissä. Tulokset on jaettu lämpötilan mukaan kolmeen osaa: kylmä, leuto ja kuuma. Ajoin ohjelman kahdessa erilaisessa ympäristössä jokaista lämpövyöhykettä kohden, jolloin erilaisia testiympäristöjä oli yhteensä kuusi. Jokaista lämpövyöhykettä kohden ympäristöt ovat luola ja vesistö, ja niissä on lähtötilanteessa aina neljä proseduraalisen eläinlajin yksilöä.

Ensin esitän kylmissä, lämpötilaltaan alle -20C° ympäristöissä tuotettuja eläinlajeja. Näissä ympäristöissä maastotyypit ovat pääosin lunta, jäätä ja jäistä vettä. Näiden ympäristöjen alkuperäiset eläinpopulaatiot koostuvat kaloista, hylkeistä, napaketuista, mursuista, jääkarhuista ja miekkavalaista.

Kylmien ympäristöjen jälkeen esittelen leudoissa ympäristöissä tuotettuja lajeja. Leutojen ympäristöjen lämpötila on noin 0C° ja niille ominaisia maastotyyppejä ovat ruohotasanko, korkea heinikko sekä suo. Leudoissa ympäristöissä esiintyy kaloja, rottia, jäniksiä, susia ja karhuja.

Viimeiseksi tarkastelen kuumia ympäristöjä, joiden lämpötila on yli 30C°. Kuumien ympäristöjen maastotyyppejä ovat savanni, hiekka-aavikko sekä kivikko. Kuumille ympäristöille ominaisia eläinlajeja ovat jänikset, leijonat ja elefantit.

Kaikissa testeissä populaation koko on 20 ja sukupolvien lukumäärä 100. Käytän kahden yksilön elitismiä, jolloin aina kaksi suurimman sopivuuden yksilöä siirtyvät muuttumattomina seuraavaan sukupolveen. Mutaatio tapahtuu 50% todennäköisyydellä ja muuttaa satunnaisesti jonkin geenin arvoa maksimissaan 25% tämän arvoalueesta. Ominaisuuksien arvoalueet ovat samat kuin edellisessä luvussa esitellyssä taulukossa 1. Joissakin ympäristöissä on mahdollista selviytyä hyvin erilaisilla

ominaisuusyhdistelmillä, joten ajoin ohjelman neljästi kutakin ympäristöä kohden. Näin vaihtelu samassa ympäristössä tuotetuissa lajeissa tulee paremmin ilmi.

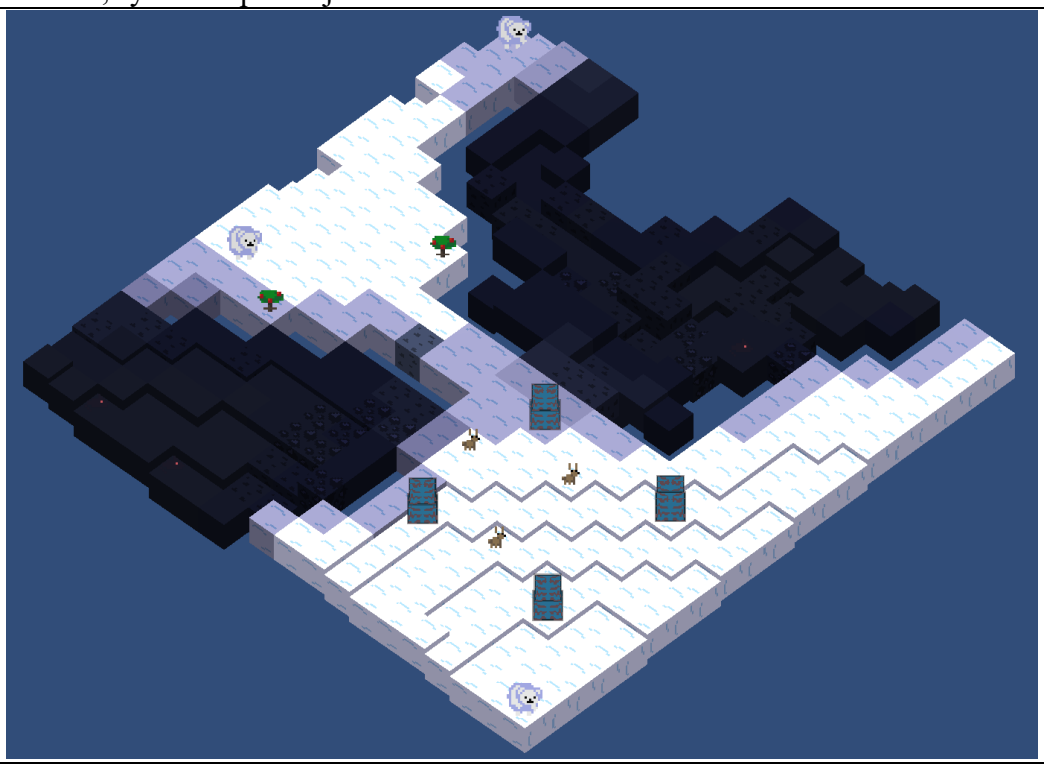
Esitän kussakin ympäristössä tuotetut eläinlajit taulukkoina, joihin on kirjattu neljän erillisen suorituskerran parhaiden kromosomien geenit. Esitän myös näiden kromosomien pohjalta piirretyt graafiset esitykset lajeista.

6.1 Kylmät ja lumiset ympäristöt

Tässä kohdassa esittelen kylmissä ympäristöissä tuotettuja eläinlajeja.

6.1.1 Kylmä luola

Kylmän luolastoympäristön luolien ulkopuolinen alue koostuu kuvan 12 mukaisesti lumi-maastotyypin ruuduista, ja luolaston sisäosa koostuu kallio-maastotyypistä. Ympäristössä on jonkin verran korkeuseroja ja epätasaisia ruutuja. Valonmäärä on luolien ulkopuolella maksimaalinen, mutta sisäpuolella hyvin alhainen. Kylmässä luolastoympäristössä elää lähtötilanteessa kolme rottaa, kolme jänistä sekä kolme jääkarhua. Lähtötilanteen luolien ulkopuolella on kaksi marjapensasta. Luolien sisäpuolella on yhdeksän vaikeasti kaivettavaa, syömäkelpoista juurta.



Kuva 12. Kylmä luolaympäristö koostuu lumesta ja kalliosta.

Jokainen taulukossa 2 esitetty kylmässä luolaympäristössä tuotettu eläinlaji on kuvattu graafisesti kuvassa 13. Jokaisella lajilla raajojen muotona on kavio, eikä millään niistä ole korvia. Kaikki ovat myös kooltaan melko pieniä. Kolmella on turkki, mutta yksi on täysin karvaton. Kahdella lajilla suun tyyppi on nokka, kahdella muulla taas torahampaat. Väritykseltään eläinlajit ovat kaikki punertavia.

	1	2	3	4
Raajan muoto	Kavio	Kavio	Kavio	Kavio
Suun muoto	Nokka	Torahampaat	Torahampaat	Nokka
Korvan muoto	Ei korvia	Ei korvia	Ei korvia	Ei korvia
Peite	Turkki	Karvaton	Turkki	Turkki
Ruumiin koko	38%	43%	27%	47%
Pään koko	100%	37%	37%	34%
Silmien lukumäärä	1	0	0	1
Silmän koko	65%	-	-	89%
Raajan pituus	96%	87%	87%	78%
Raajan paksuus	58%	78%	92%	73%
Pohja (R, G, B)	(193, 32, 1)	(120, 43, 65)	(120, 48, 65)	(123, 47, 155)
Kuvio (R, G, B)	(63, 105, 81)	(190, 32, 254)	(190, 32, 254)	(210, 163, 80)

Taulukko 2. Kylmässä luolassa tuotettujen olentojen geenien arvot.

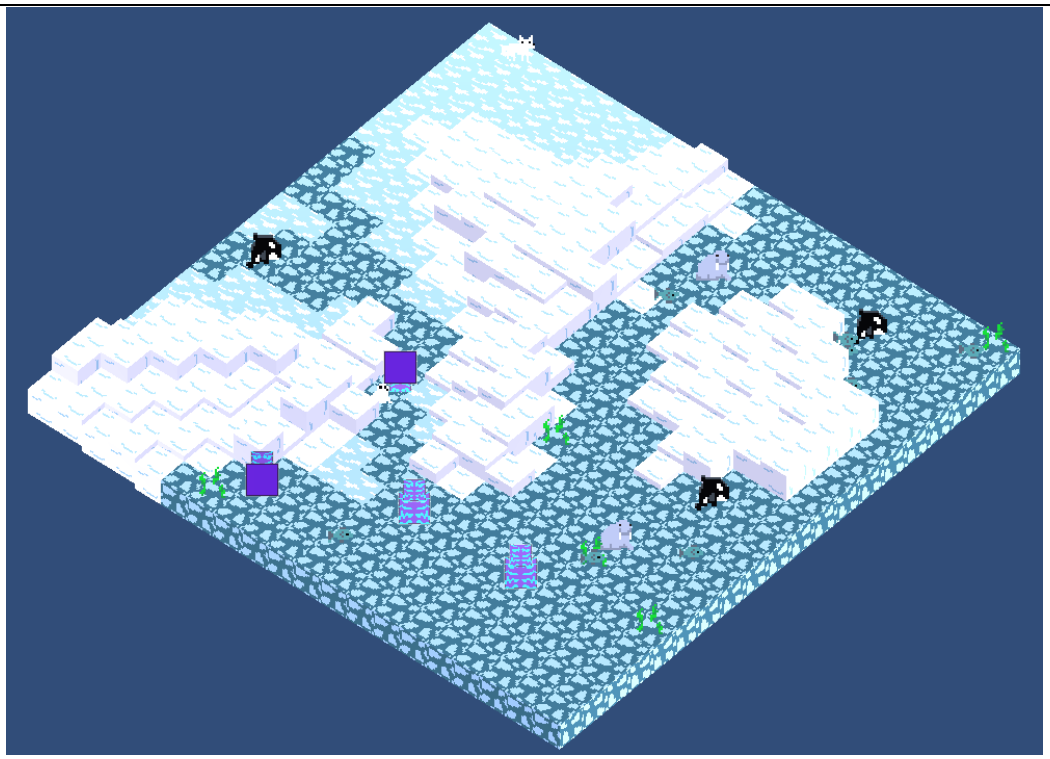
Kuva 13. Taulukkoa 2 vastaavat, kylmässä luolaympäristössä tuotetut eläinlajit 1-4 järjestyksessä vasemmalta oikealle.

Torahampaisilla lajeilla ei ole lainkaan silmiä, kun taas nokan omaavilla lajeilla on yksi silmä. Tämä johtunee siitä, että nokan hajuaisti oli määritelty heikoksi, kun taas torahampaiden hajuaisti oli hyvä. Koska kaikki lajit ovat korvattomia, torahampaiset eläinlajit saalistivat simulaatiossa pelkän hajuaistinsa varassa. Koska pimeässä luolassa kunnolla näkemiseen olisi vaadittu suuret silmät, oli ravinnonkäytön kannalta tehokkaampaa jättää silmät kokonaan pois.

6.1.2 Kylmä vesistö

Kylmä vesistöympäristö koostuu lumisista kukkuloista, jäätiköistä ja jäävedestä. Ympäristön esitys SPLE-järjestelmässä on esitetty kuvassa 14. Alkutilanteessa

ympäristössä esiintyy 16 kalaa, kaksi hyljettä, kaksi napakettua, kaksi mursua ja kolme miekkavalasta. Ainoat kasviyksiköt ovat yhdeksän vedenalaista leväresurssia.



Kuva 14. Kylmä vesistöympäristö.

Tässä ympäristössä järjestelmä tuotti ominaisuuksiltaan hyvin samankaltaisia, kookkaita ja paksuraajaisia eläinlajeja, joilla on suhteessa melko pienet päät. Kylmässä vesistöympäristössä tuotettujen eläinlajien ominaisuudet on listattu taulukossa 3 ja niiden ulkonäkö on kuvattu graafisesti kuvassa 15. Kaikilla tuotetuilla lajeilla on karvapeite, raajan muotona evät, eikä lainkaan korvia. Nämä yhtenäisyydet johtuvat luultavasti siitä, että ympäristössä on paljon vettä ja vedessä eläviä eläinlajeja. Lisäksi kaikki kasviresurssit sijaitsevat veden alla. Suuri koko saattaa selittyä suurella määrällä petoeläimiä. Kahdella tuotetulla eläinlajilla on nokka, jolla ne todennäköisesti metsästivät muita eläimiä. Kahdella muulla lajilla on sen sijaan kärsät, joita ne luultavasti käyttivät levän syömiseen.

Kuva 15. Taulukkoa 3 vastaavat, kylmässä vesistöympäristössä tuotetut eläinlajit 1-4 järjestyksessä vasemmalta oikealle.

	1	2	3	4
Raajan muoto	Evä	Evä	Evä	Evä
Suun muoto	Nokka	Kärsä	Kärsä	Nokka
Korvan muoto	Ei korvia	Ei korvia	Ei korvia	Ei korvia
Peite	Turkki	Turkki	Turkki	Turkki
Ruumiin koko	95%	91%	91%	99%
Pään koko	29%	30%	29%	49%
Silmien lukumäärä	2	1	1	2
Silmän koko	36%	65%	89%	64%
Raajan pituus	88%	79%	46%	64%
Raajan paksuus	100%	96%	96%	79%
Pohja (R, G, B)	(68, 183, 240)	(151, 104, 215)	(151, 129, 84)	(130, 170, 31)
Kuvio (R, G, B)	(227, 200, 98)	(79, 243, 157)	(117, 174, 113)	(237, 229, 92)

Taulukko 3. Kylmässä vesistössä tuotettujen eläinlajien geenien arvot.

6.2 Leudot ympäristöt

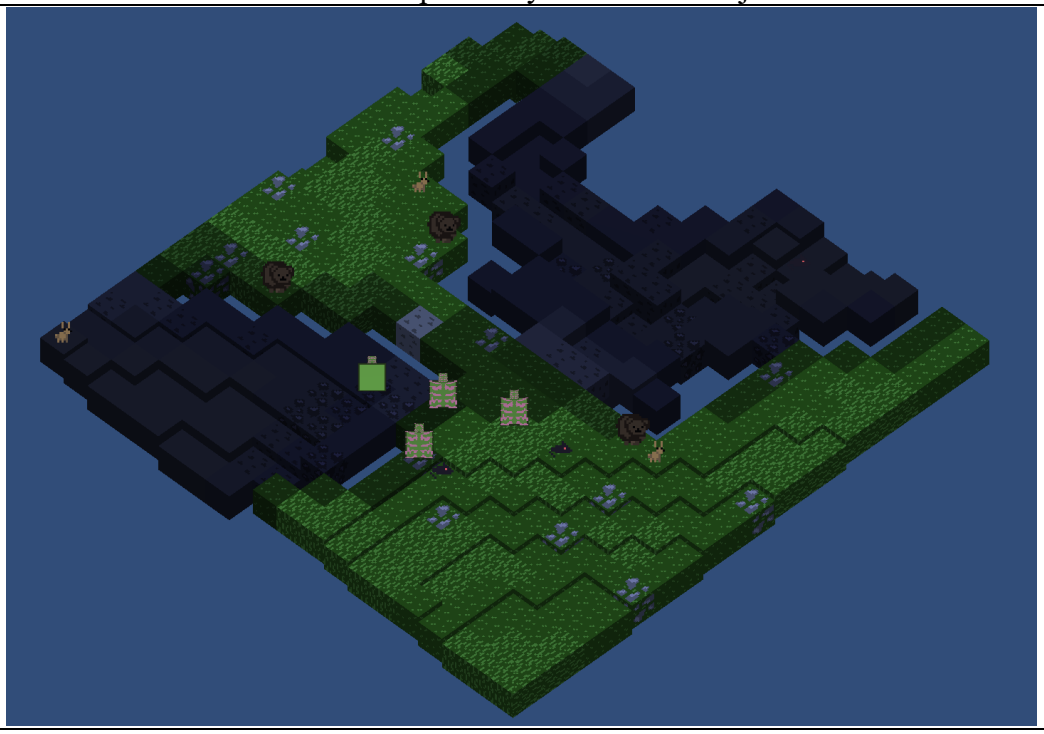
Tässä kohdassa esittelen leudoissa ympäristöissä tuotettuja eläinlajeja.

6.2.1 Leuto luolaympäristö

Kuvassa 16 esitetyn leudon luolastoympäristön luolien ulkopuolinen alue koostuu ruohikosta ja korkeasta heinikosta. Luolaston sisäosa koostuu kallio-maastotyypistä. Ympäristössä on jonkin verran korkeuseroja ja epätasaisia ruutuja. Valonmäärä on luolien ulkopuolella maksimaalinen, mutta sisäpuolella hyvin alhainen. Leudossa luolastoympäristössä elää lähtötilanteessa kolme rottaa, kolme jänistä sekä kolme karhua. Lähtötilanteen luolien ulkopuolella on kaksi kolme marjapensasta. Luolien sisäpuolella on yhdeksän vaikeasti kaivettavaa, syömäkelpoista juurta.

Leudossa luolastoympäristössä tuotettujen eläinlajien ominaisuudet on esitetty taulukossa 4 ja lajit on kuvattu graafisesti kuvassa 17. Leudossa luolastoympäristössä

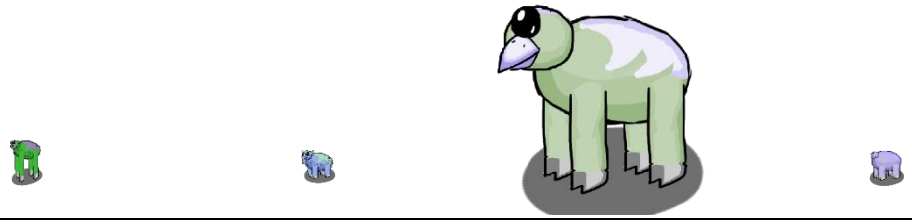
tuotetuilla eläinlajeilla on kaikilla kaviot. Kolme niistä on kooltaan hyvin pieniä, kun taas yksi on huomattavasti muita suurempi. Väritykseltään eläinlajit ovat sinivihertäviä.



Kuva 16. Luolaympäristö leudossa ilmastossa.

	1	2	3	4
Raajan muoto	Kavio	Kavio	Kavio	Kavio
Suun muoto	Kuono	Kärsä	Nokka	Kuono
Korvan muoto	Pieni	Suunnattu	Ei korvia	Ei korvia
Peite	Turkki	Turkki	Karvaton	Karvaton
Ruumiin koko	10%	10%	84%	11%
Pään koko	26%	33%	47%	11%
Silmien lukumäärä	1	2	1	1
Silmän koko	92%	88%	100%	67%
Raajan pituus	95%	40%	58%	41%
Raajan paksuus	57%	65%	67%	74%
Pohja (R, G, B)	(33, 148, 29)	(57, 99, 208)	(120, 159, 92)	(117, 100, 189)
Kuvio (R, G, B)	(109, 84, 141)	(90, 181, 144)	(196, 185, 243)	(126, 106, 235)

Taulukko 4. Leudossa luolaympäristössä tuotettujen eläinlajien geenien arvot.



Kuva 17. Taulukkoa 4 vastaavat, leudossa luolaympäristössä tuotetut eläinlajit 1-4 järjestyksessä vasemmalta oikealle.

6.2.2 Leuto vesistöympäristö

Toteutin leudon vesistön kuvan 18 suoalueena, jonka reunalla on kalliota. Ympäristön maastotyypit ovat ruohikko, korkea heinikko, vesi, suo ja kallio. Ympäristössä on jonkin verran korkeuseroja ja epätasaisia ruutuja. Valonmäärä on kaikkialla maksimaalinen.

Kuva 18. Suoalue leudossa ympäristössä.

Leudossa vesistöympäristössä on lähtötilanteessa kolme jänistä, kolme kalaa, yksi hirvi, yksi karhu, yksi susi ja yksi krokotiili. Lähtötilanteessa ympäristössä on kaksi syömäkelpoista juurta, kuusi yksikköä levää, kolme marjapensasta ja kaksi hedelmäpuuta.

Leudossa vesistöympäristössä tuotettujen eläinlajien ominaisuudet on esitetty taulukossa 5 ja lajit on kuvattu graafisesti kuvassa 19. Kaikilla tuotetuilla lajeilla on räpylät ja ne ovat karvattomia. Lajit ovat myös kooltaan pieniä ja niillä on kaikilla korvat.

	1	2	3	4
Raajan muoto	Räpylä	Räpylä	Räpylä	Räpylä
Suun muoto	Kuono	Kärsä	Kärsä	Kuono
Korvan muoto	Pieni	Suunnattu	Pieni	Suunnattu
Peite	Karvaton	Karvaton	Karvaton	Karvaton
Ruumiin koko	20%	13%	28%	17%
Pään koko	68%	13%	51%	55%
Silmien lukumäärä	1	2	1	2
Silmän koko	77%	65%	64%	91%
Raajan pituus	69%	74%	19%	28%
Raajan paksuus	66%	72%	95%	86%
Pohja (R, G, B)	(62, 87, 204)	(28, 228, 18)	(193, 115, 131)	(93, 132, 124)
Kuvio (R, G, B)	(169, 101, 154)	(231, 120, 218)	(134, 241, 213)	(53, 154, 136)

Taulukko 5. Leudossa vesistöympäristössä tuotettujen eläinlajien geenien arvot.

Kuva 19. Taulukkoa 5 vastaavat, leudossa vesistöympäristössä tuotetut eläinlajit 1-4 järjestyksessä vasemmalta oikealle.

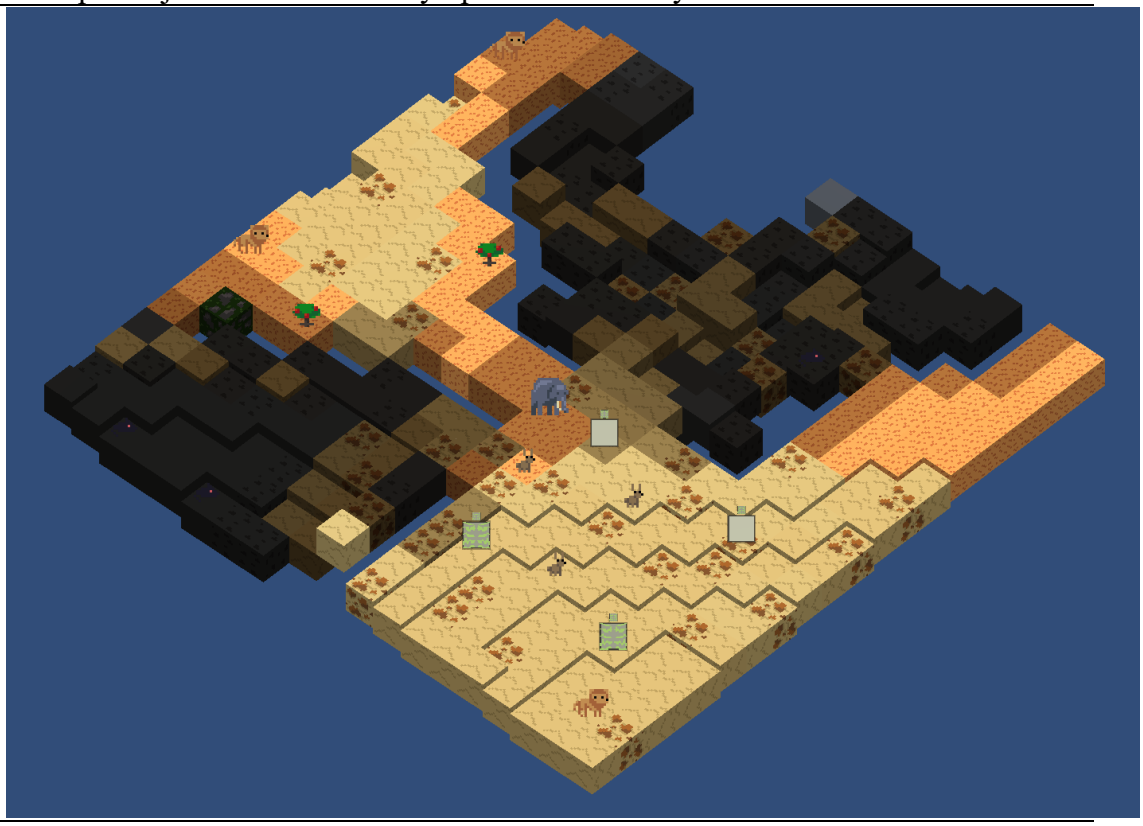
6.3 Kuumat ympäristöt

Tässä aliluvussa esittelen kuumissa ympäristöissä tuotettuja eläinlajeja.

6.3.1 Kuuma luolaympäristö

Kuuman luolastoympäristön luolien ulkopuolinen alue koostuu savanni-maastotyypin sekä hiekka-aavikko-maastotyypin ruuduista. Luolaston sisällä maastotyyppejä ovat hiekka-aavikko ja kallio. Ympäristössä on jonkin verran korkeuseroja ja epätasaisia ruutuja. Valonmäärä on luolien ulkopuolella maksimaalinen, mutta sisäpuolella hyvin alhainen. Kuumassa luolastoympäristössä elää lähtötilanteessa kolme rottaa, kolme jänistä, kolme leijonaa ja yksi elefantti. Lähtötilanteessa luolien ulkopuolella on kaksi

kaktusta ja kaksi marjapensasta. Luolien sisäpuolella on yhdeksän vaikeasti kaivettavaa, syömäkelpoista juurta. Kuuma luolaympäristö on esitetty kuvassa 20.



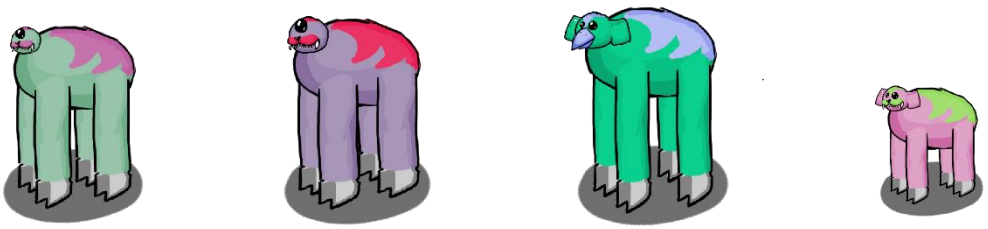
Kuva 20. Kuuma luolaympäristö.

Kaikilla tuotetuilla eläinlajeilla raajan muoto on kavio ja ne ovat karvattomia. Kaikki lajit muistuttavat toisiaan myös sekä ruumiin että pään koon puolesta. Ne ovat pienipäisiä ja kooltaan keskikokoisia. Kuumassa luolaympäristössä tuotettujen eläinlajien ominaisuudet on esitetty taulukossa 6.

Lajien väliltä löytyy myös eroja. Kahdella lajilla ei ole lainkaan korvia, kun taas muilla kahdella on suuret korvat. Korvattomilla lajeilla on vain yksi suurehko silmä, kun taas suurikorvaisilla lajeilla on kaksi keskikokoista silmää. Kolmella lajilla on torahampaat ja yhdellä nokka. Molemmat suutyypit ovat huomattavasti tehokkaampia lihansyömisessä kuin kasvien syömisessä.

	1	2	3	4
Raajan muoto	Kavio	Kavio	Kavio	Kavio
Suun muoto	Torahampaat	Torahampaat	Nokka	Torahampaat
Korvan muoto	Ei korvia	Ei korvia	Suuri	Suuri
Peite	Karvaton	Karvaton	Karvaton	Karvaton
Ruumiin koko	63%	66%	68%	45%
Pään koko	10%	23%	21%	22%
Silmien lukumäärä	1	1	2	2
Silmän koko	75%	75%	53%	57%
Raajan pituus	89%	87%	96%	70%
Raajan paksuus	85%	91%	75%	70%
Pohja (R, G, B)	(79, 139, 104)	(113, 90, 135)	(8, 149, 103)	(192, 71, 144)
Kuvio (R, G, B)	(155, 63, 127)	(195, 20, 67)	(86, 106, 230)	(132, 212, 64)

Taulukko 6. Kuumassa luolaympäristössä tuotettujen eläinlajien geenien arvot.



Kuva 21. Taulukkoa 6 vastaavat, kuumassa luolaympäristössä tuotetut eläinlajit 1-4 järjestyksessä vasemmalta oikealle.

6.3.2 Kuuma vesistöympäristö

Kuuma vesistöympäristö kuvaa kuvan 22 mukaisesti savannin halki kulkevaa jokea. Jokea ympäröivä alue koostuu savanni-maastotyypin sekä hiekka-aavikko-maastotyypin ruuduista. Joki koostuu vesi- ja ruokovesi-maastotyypeistä. Ympäristössä on jonkin verran korkeuseroja ja epätasaisia ruutuja. Valonmäärä on koko ympäristössä maksimaalinen. Kuumassa vesistöympäristössä elää lähtötilanteessa viisi kalaa, neljä jänistä, kolme kilpikonnaa, kaksi krokotiilia, kaksi leijonaa ja yksi elefantti. Lähtötilanteessa joessa kasvaa neljä yksikköä levää ja joen ulkopuolella kaksi yksikköä syömäkelpoisia juuria, kaksi marjapensasta sekä kaksi hedelmäpuuta.

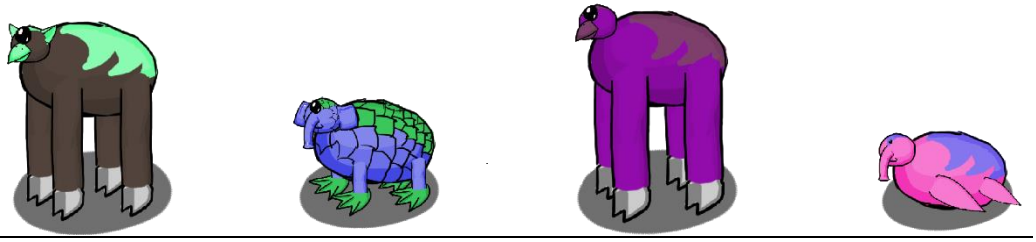
Kuumassa vesistöympäristössä tuotettujen eläinlajien ominaisuudet on esitetty taulukossa 7 ja niiden graafiset esitykset kuvassa 23. Tuotetut eläinlajit ovat kaikki suurin

piirtein saman kokoisia, ja niillä on pienet päät. Kaikilla lajeilla on myös vain yksi silmä, mutta muuten ne ovat ominaisuuksiltaan ja värityksiltään hyvin erilaisia.

Kuva 22. Kuuman vesistöympäristön joki keskellä savannia.

	1	2	3	4
Raajan muoto	Kavio	Räpylä	Kavio	Evä
Suun muoto	Nokka	Kärsä	Nokka	Kärsä
Korvan muoto	Suunnattu	Suuri	Ei korvaa	Ei korvaa
Peite	Karvaton	Suomut	Karvaton	Karvaton
Ruumiin koko	73%	63%	73%	59%
Pään koko	18%	24%	16%	24%
Silmien lukumäärä	1	1	1	1
Silmän koko	98%	71%	90%	22%
Raajan pituus	85%	27%	93%	60%
Raajan paksuus	65%	34%	78%	43%
Pohja (R, G, B)	(41, 34, 30)	(29, 34, 190)	(98, 8, 115)	(221, 12, 155)
Kuvio (R, G, B)	(58, 248, 125)	(26, 189, 65)	(96, 61, 86)	(88, 66, 232)

Taulukko 7. Kuumassa vesistössä tuotettujen olentojen geenien arvot.



Kuva 23. Taulukkoa 7 vastaavat, kuumassa vesistöympäristössä tuotetut eläinlajit 1-4 järjestyksessä vasemmalta oikealle.

6.4 Tulokset ja pohdintaa

Esittämissäni kuudessa testiympäristössä järjestelmä tuotti eri ympäristöissä toisistaan poikkeavia eläinlajeja, mutta samassa ympäristössä tuotetuilla lajeilla on useassa tapauksessa selviä yhteneväisyyksiä. Esimerkiksi leudossa vesistöympäristössä tuotetut eläinlajit ovat kaikki räpyläjalkaisia, karvattomia ja kooltaan pieniä. Kylmässä vesistöympäristössä tuotetuilla lajeilla sen sijaan on kaikilla evät, karvapeite ja ne ovat kooltaan suuria.

Kaikista näkyvimmät erot saman ympäristön lajien välillä ovat kuumassa vesistöympäristössä, mikä saattaa selittyä sillä, että kyseisessä ympäristössä on mahdollista selviytyä usealla erilaisella ominaisuusyhdistelmällä. Muissa ympäristöissä eläinlajit ovat selvästi enemmän toistensa kaltaisia. Tämä voi tietysti johtua myös sattumasta, sillä vain neljän eläinlajin otanta on melko pieni.

Ominaisuuksien vaikutus eläinlajin selviytymiseen ei ole biologisesti kovin uskottavaa, mikä johtuu ainakin osittain siitä, että kromosomiesitys ja simulaatiosäännöt on täysin mielivaltaisesti määritelty. Lajit ovat värityksiltään hyvin kirjavia eikä yli puolella lajeista ole lainkaan korvia. Ilmeisesti kuuloaistin hyödyt eivät käytetyillä simulaatiosäännöillä olleet riittävän hyvät suhteessa kasvaneeseen ravinnonkulutukseen. Jotta järjestelmää voisi käyttää tuottamaan eläinlajeja johonkin tiettyyn peliin, tulisi pelisimulaation rakenne ja säännöt sekä geenien vaikutus suunnitella juuri kyseiseen peliin sopiviksi. Lisäksi olennaista olisi miettiä, miten kromosomi muunnetaan sisällöksi varsinaiseen peliin.

Simulaatio on selvästi järjestelmän laskennallisesti raskain osuus. Suoritusaika edellä kuvatuissa testiympäristöissä vaihteli 20 – 35 minuutin välillä, kun populaation koko oli 20 ja sukupolvien lukumäärä 100. Yhden lajin selviytymisen simuloiminen kesti siis keskimäärin noin 0,6 - 1,05 sekuntia. Suoritusaika on huomattavan pitkä, eikä järjestelmää sen vuoksi voi käyttää sisällön tuottamiseen online-aikana.

Järjestelmää ei testattu muilla populaation ko'oilla tai sukupolvien määrillä, eivätkä eläinten lähtötilanteet vaihdelleet ympäristön sisällä. Jatkokehityksen kannalta olisi

kuitenkin kiinnostavaa tietää, miten nämä vaikuttavat lopputulokseen. Myös eri geneettisten operaatioiden käytön vaikutuksia tuloksiin olisi kiinnostavaa tarkastella. Lisäksi simulaatiosäännöt ja kromosomikoodauksen voisi määritellä paremmin, esimerkiksi jonkin valmiin pelin pohjalta tai mukailemaan todellista biologiaa.

Esitin tässä luvussa esimerkkejä, minkälaisia eläinlajeja SPLE-järjestelmällä voidaan tuottaa. Vaikka tulokset näyttävät lupaavilta, vaatii kuitenkin huomattavasti laajempaa testausta, jotta mitään varmoja johtopäätöksiä geneettisen algoritmin käytöstä ympäristöön sopeutuvien eläinlajien tuottamiseen voidaan tehdä.

7 Yhteenveto

Tässä tutkielmassa tarkastelin, miten eläinlajeja voidaan tuottaa proseduraalisesti. Erityisesti keskityin elinympäristöönsä sopeutuvien eläinlajien tuottamiseen geneettisten algoritmien avulla sekä kehittämäni SPLE-järjestelmän esittelyyn.

Luvussa 2 esittelin proseduraalisen sisällöntuotannon käsitteitä, käyttötarkoituksia ja etuja. Tuottamalla sisältöä proseduraalisesti voidaan vähentää kehityskustannuksia, mahdollistaa uudenlaisia pelikokemuksia ja lisätä pelin uudelleenpelattavuusarvoa. Konstruktiiviset sisällöntuotantomenetelmät tuottavat vain yhden kappaleen sisältöä, joka hyväksytään sellaisenaan. Tuota-ja-testaa -menetelmät sen sijaan tuottavat sisällönkappaleita, kunnes löytyy riittävän hyvä kappale, joka voidaan kelpuuttaa peliin. Geneettiset algoritmit ovat hakupohjaisia optimointimenetelmiä, jotka proseduraalisen sisällöntuotannon näkökulmasta kuuluvat tuota-ja-testaa -menetelmiin.

Luvussa 3 pureuduin tarkemmin geneettisiin algoritmeihin. Niiden toimintaperiaate perustuu evoluutioon ja luonnonvalintaan; vahvat selviävät ja lisääntyvät, mutta heikot kuolevat pois. Ratkaisuehdokkaat koodataan kromosomeiksi esimerkiksi binäärikoodauksella tai kokonaislukukoodauksella. Jokaisen kromosomin sopivuus, eli kyky ratkaista kyseessä oleva ongelma, mitataan sopivuusfunktiolla. Tämän jälkeen populaatiosta valitaan satunnaisesti joukko vanhempien pareja, joiden jälkeläisistä muodostuu seuraava sukupolvi. Vanhemmiksi valitaan todennäköisimmin korkean sopivuuden kromosomeja, jolloin seuraavan sukupolven keskimääräinen sopivuus kasvaa. Prosessia toistetaan, kunnes jokin lopetusehto täyttyy ja korkeimman sopivuuden kromosomi palautetaan.

Luvussa 4 tarkastelin eläinlajeja videopeleissä tuotettavana sisältönä. Eläinlajit ovat videopelien sisältöä, tarkemmin pelin palasia. Niiden tuottamiseen voidaan käyttää samoja proseduraalisen sisällöntuotannon menetelmiä kuin muunkin videopelisisällön tuottamiseen. Erilaisiin ympäristöihin sopeutuvia eläinlajeja voidaan tuottaa geneettisten algoritmien avulla. Videopelien kannalta eläinlajien kiinnostavia ominaisuuksia ovat näiden näkyvät ominaisuudet, joten sopivuusfunktion tulee mitata näiden ominaisuuksien

vaikutusta lajin selviytymiseen elinympäristössä. Lajin selviytymistä monimutkaisessa peliympäristössä voidaan mitata simulaatiopohjaisella sopivuusfunktiolla.

Luvussa 5 esittelin toteuttamani SPLE-järjestelmän, joka tuottaa vaihteleviin videopeliympäristöihin sopivia eläinlajeja geneettisen algoritmin ja simulaatiopohjaisen sopivuusfunktion avulla. Ajamissani testeissä simulaatio perustuu karkeasti tyypillisiin videopelisääntöihin, mutta täsmällisemmät simulaatiosäännöt tulee suunnitella aina sen pelin mukaan, johon eläinlajeja ollaan tuottamassa. Kromosomien geenit kuvaavat eläinlajien sellaisia ulkoisia ominaisuuksia, jotka määrittävät samalla pelihahmon selviytymiskyvyn simulaatiossa.

Luvussa 6 esittelin SPLE-järjestelmän tuottamia eläinlajeja. Ajoin järjestelmän kuudessa eri ympäristössä, kussakin neljä kertaa. Testiajoissa populaation koko oli 20 ja sukupolvien lukumäärä 100. Järjestelmä näytti tuottavan erilaisia ratkaisuja erilaisissa ympäristöissä. Samoissa ympäristöissä tuotetut eläinlajit taas pääsääntöisesti muistuttivat toisiaan. Tulokset ovat lupaavia, mutta järjestelmä kaipaisi jatkokehitystä ja lisää testausta, jotta tarkempia johtopäätöksiä sen toiminnasta voidaan vetää.

8 Viiteluettelo

Amato, Alba. 2017. Procedural content generation in the game industry. *Game Dynamic*s. Springer, Cham, 15-25.

Barton, Nicholas H. and Brian Charlesworth. 1998. Why sex and recombination?. *Science*, 281, 5385, 1986-1990.

Cardamone, Luigi, Georgios N. Yannakakis, Julian Togelius and Pier Luca Lanzi. 2011. Evolving interesting maps for a first person shooter. *European Conference on the Applications of Evolutionary Computation*. Springer, Berlin, Heidelberg, 63-72.

D&D Basic Rules. 2014. Wizards of the Coast. *https://media.wizards.com/2018/dnd/downloads/DnD_BasicRules_2018.pdf.*

Diablo. 1997. Blizzard Entertainment. *https://www.blizzard.com/en-us/games/legacy/.*

Dwarf Fortress. 2006. Bay 12 Games. *http://www.bay12games.com/dwarves/.*

Dwarf Fortress Wiki. 2020. DF2014:Forgotten beast. *https://dwarffortresswiki.org/index.php/DF2014:Forgotten_beast.*

Ferreira, Lucas and Claudio Toledo. 2014. A search-based approach for generating Angry Birds levels. *IEEE Conference on Computational Intelligence and Games*, 1-8.

Gen, Mitsuo and Lin Lin. 2007. Genetic Algorithms. *In Wiley Encyclopedia of Computer Science and Engineering*. 1-15.

Goldberg, David E. and Kalyanmoy Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*. Elsevier, 1, 69-93.

Gregkwaste. 2016. No Man's Sky – Procedural content. *3DGameDevblog. http://3dgamedevblog.com/?p=836.*

Hart, Peter E., Nils J. Nilsson and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4, 2, 100-107.

Hendrikx, Mark, Sebastiaan Meijer, Joeri van der Velden and Alexandru Iosup. 2013. Procedural content generation for games. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9, 1, 1-22.

Herrera, Francisco, Manuel Lozano and Ana M. Sánchez. 2003. A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. *International Journal of Intelligent Systems*, 18, 3, 309-338.

Holland, John. 1992. Genetic algorithms. *Scientific American*, 267, 1, 66-73.

Liapis, Antonios, Georgios N. Yannakakis and Julian Togelius. 2013. Towards a generic method of evaluating game levels. *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference,* 30-36. *https://www-aaai-org.libproxy.tuni.fi/ocs/index.php/AIIDE/AIIDE13/paper/view/7374/7585.*

Lim, Siew Mooi, Abu Bakar Md Sultan, Md. Nasir Sulaiman, Aida Mustapha and K. Y. Leong. 2017. Crossover and mutation operators of genetic algorithms. *International Journal of Machine Learning and Computing,* 7, 1, 9-12.

Van der Linden, Roland, Ricardo Lopes and Rafael Bidarra. 2013. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6, 1, 78-89.

Alexander, R. McNeill. 2003. *Principles of animal locomotion*. Princeton University Press.

Minecraft. 2011. Mojang. *https://www.minecraft.net/*.

Neogames Finland ry. 2018. The game industry of Finland. *http://www.neogames.fi/wp-content/uploads/2019/04/FGIR-2018-Report.pdf*.

No Man's Sky. 2016. Hello Games. *https://www.nomanssky.com/.*

Oxford University Press. 2019. *Lexico.com*.

PokéFanon Wiki. *Ice bell showcase. https://pokemonfanon.fandom.com/wiki/Ice_Bell_Showcase.*

Poon, Pui Wah and Jonathan Neil Carter. 1995. Genetic algorithm crossover operators for ordering applications. *Computers & Operations Research*, 22, 1, 135-147.

Rogue. 1980. A.I. Design. [1]

Schiesel, Seth. 2020. Why big tech is betting big on gaming in 2020. *Protocol*. *https://www.protocol.com/tech-gaming-amazon-facebook-microsoft*.

Shaker, Noor, Julian Togelius and Mark J. Nelson. 2016. *Procedural content generation in games*. Switzerland: Springer International Publishing.

Shukla, Anupriya, Hari Mohan Pandey and Deepti Mehrotra. 2015. Comparative review of selection techniques in genetic algorithm. *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE),* 515-519.

Smith, Gillian, Elaine Gan, Alexei Othenin-Girard, and Jim Whitehead. 2011. PCG-based game design: enabling new play experiences through procedural content generation. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, 1–4.

Spelunky. 2008. Mossmouth, LLC. *https://spelunkyworld.com/original.html*.

Suomen virallinen tilasto (SVT). 2017. Digitaalisten pelien pelaaminen nelinkertaistunut 25 vuodessa. *Vapaa-ajan osallistuminen [verkkojulkaisu]. https://www.stat.fi/til/vpa/2017/02/vpa_2017_02_2019-01-31_kat_001_fi.html*.

theHunter: Call of the Wild. 2017. Expansive Worlds. *http://callofthewild.thehunter.com/.*

Tiled. 2009. Thorbjørn. *https://www.mapeditor.org/.*

[1] Rogue -pelillä, sen kehittäjillä tai julkaisijalla ei ole enää toimivia verkkosivuja.

Togelius, Julian, Renzo De Nardi and Simon M. Lucas. 2006. Making racing fun through player modeling and track evolution. *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games.*

Togelius, Julian and Georgios N. Yannakakis. 2011. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2, 3, 147-161.

Togelius, Julian and Georgios N. Yannakakis, Kenneth O. Stanley and Cameron Browne. 2011. Search-based procedural content generation: a taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3, 3, 172-186.