Aku Telimaa

Luonnonvalinnan mallinnus peliympäristössä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

13.11.2023

Tiivistelmä

Tekijä:	Aku Telimaa
Otsikko:	Luonnonvalinnan mallinnus peliympäristössä
Sivumäärä:	35 sivua
Aika:	13.11.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaaja:	Lehtori Antti Laiho

Insinöörityössä tutkittiin luonnonvalinnan toimintaa ja sen soveltuvuutta osaksi peliympäristöä. Työn tavoitteena oli käydä läpi käytöspuutekniikalla luotu tekoäly, luoda luonnonvalinnan avulla toimiva peliympäristö ja pohtia luonnonvalinnan mekanismien hyödyntämistä peleissä.

Työn toteuttamiseen käytettiin Unity-pelimoottoria. Pelimoottorilla luotiin elinympäristö kahdelle eri lajille: ketuille ja jäniksille. Lajien tekoälyn pohjaksi luotiin yksinkertainen käytöspuu. Työssä lajin yksilöt muuttuvat luonnonvalinnan periaatteiden pohjalta.

Insinöörityön tuloksena saatiin luotua lajeille toimiva tekoäly, joka pääosin täyttää tekoälylle asetetut kriteerit. Luonnonvalinnan mallintaminen tehtiin populaation geeneissä tapahtuvien mutaatioiden avulla. Simulaatio onnistui näyttämään, miten epätasapainoisia monet ekosysteemit ovat ja että tasapainoisen elinympäristön luominen ilman ulkoisia tekijöitä voi olla haasteellista ja usein sattuman varassa.

Työstä ilmeni, että täysin luonnonvalinnan mekanismeilla toimiva peliympäristö ei välttämättä ole hyvä tapa luoda ekosysteemiä peliympäristöön sen satunnaisuuden takia. Vaikka luonnonvalinta on tehokas tapa muuttaa peliympäristöä, sen suuntaa on vaikea ennustaa.

Avainsanat: tekoäly, Unity, luonnonvalinta, käytöspuu

Abstract

Author:	Aku Telimaa
Title:	Scientific Modeling of Natural Selection in Game Environment
Number of Pages:	35 pages
Date:	13 November 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Game Applications
Supervisor:	Antti Laiho, Senior Lecturer

This thesis examines the mechanisms of natural selection and its suitability as part of a game environment. The goal of the study is to explore the use of behavior tree-based artificial intelligence (AI), create a functioning gaming environment utilizing natural selection, and discuss the use of natural selection mechanisms in games.

The Unity game engine was used for the implementation of the game. An ecosystem was created using the game engine for two different species: foxes and rabbits. The AI for each species was created based on a simple behavior tree. In the study, the individuals of each species change based on principles of natural selection.

A functioning AI was created for each species, which overall meets the criteria set for the AI. The modeling of the natural selection occurred through mutations in the population's genes. The simulation successfully showed that creating a balanced ecosystem without external factors can be challenging, often relying on chance.

The conclusion of this study is that a gaming environment based on natural selection mechanisms alone may not be a good way to create ecosystems in a game environment due to its randomness. Even though natural selection is an effective way to alter a game environment, its direction is difficult to predict.

Keywords: AI, Unity, natural selection, behavior tree

Sisällys

1 Johdanto

Luonnontieteiden yhdistäminen peliteollisuuteen on ollut pitkään henkilökohtainen kiinnostuksen aihe. Luonnonvalinnan integroiminen peliympäristöön kuulosti mielenkiintoiselta haasteelta.

Insinöörityön lähtökohtana oli tutustua luonnonvalinnan toimintaan ja mallintaa luonnonvalinnan toimintaa peliympäristössä. Työssä käytettäväksi peliympäristöksi valittiin Unity-pelimoottori. Työn keskeisimpänä ajatuksena oli pyrkiä opettelemaan käytöspuiden käyttöä ei-pelaajahahmojen ohjaukseen. Työn tavoitteena oli siis luoda peliympäristö, jossa tietokoneen ohjaamat hahmot kehittyisivät samalla mekanismilla, jolla erilaiset organismit luonnossa kehittyvät.

Tekoäly on nopeasti yleistynyt peliteollisuudessa, ja mitä monimutkaisimpien hahmojen luominen peleihin on yhä tärkeämpää. Luonnonvalinnan käyttäminen AAA-peleissä on kuitenkin harvinaista; vaikka jonkin verran aiheeseen liittyviä pelejä on tehty, vain muutamat ovat nousseet laajan yleisön tietoisuuteen.

Työn tavoitteena on luoda kahden lajin välille toimiva ekosysteemi, jota luonnonvalinta pystyy ohjaamaan niin, että kumpikin laji kykenee muovautumaan ympäristöönsä. Lopuksi mietitään syitä siihen, minkä takia evoluution yhdistäminen peleihin ei ole noussut valtavirran käyttöön ja minkälaisilla tavoilla se olisi mahdollista vai onko sen tavoittelu edes tarpeellista tai haluttua.

2 Luonnonvalinta käsitteenä

Luonnonvalinnalla tarkoitetaan mekanismia, jolla eri lajit pystyvät sopeutumaan vallitseviin olosuhteisiin. Sen kehittivät itsenäisesti Charles Darwin vuonna 1859 julkaisemassaan teoksessa Lajien synty ja Alfred Wallace julkaisemassaan artikkelissa vuonna 1858. Charles Darwinia pidetään kuitenkin yleisesti luonnonvalinta-käsitteen esi-isänä. (Osterloff.)

2.1 Luonnonvalinnan mekanismit

Luonnonvalinnan mekanismi on yleisellä tasolla yksinkertainen. Ne yksilöt, jotka pystyvät selviytymään ja lisääntymään, välittävät omat geeninsä seuraavalle sukupolvelle. Geeneillä tarkoitetaan yksilön fyysisiä ja biologisia ominaisuuksia, jotka periytyvät vanhemmilta jälkikasvulle. Populaatioissa on geneettistä vaihtelua, joka aiheutuu mutaatioista geneettisessä perimässä. Ne geenit, jotka parantavat yksilön todennäköisyyttä selviytyä tarpeeksi pitkään lisääntyäkseen, lisääntyvät populaatiossa. Näin ollen tietyt ominaisuudet yleistyvät geenipoolissa. Geenipoolilla tarkoitetaan kaikkia populaatiossa ilmeneviä geenejä. Kun jokin geeni yleistyy geenipoolissa, sen aiheuttamat muutokset lajeissa ohjaavat koko populaatiota tiettyyn suuntaan. (Osterloff; Cox & Cohen 2016.)

Jos populaatio on erossa muista saman lajin populaatioista pitkiä aikoja, esimerkiksi maantieteellisen esteen vuoksi, voi populaatio eriytyä omaksi lajikseen. Laji-termillä ei ole vahvaa virallista määritelmää, mutta usein laji määritellään sen perusteella, mitkä yksilöt voivat pariutua keskenään ja tuottaa lisääntymiskykyisiä jälkeläisiä. Jotkin eri lajit pystyvät lisääntymään keskenään, mutta niiden jälkeläiset eivät ole lisääntymiskykyisiä. (Cox & Cohen 2016.)

Evoluutio voi toimia myös ilman, että geeneissä tapahtuu mutaatioita. Tällaista tilannetta kutsutaan geneettiseksi ajautumiseksi. Ideana on, että geenien esiintyvyys populaatiossa vaihtelee satunnaisesti sen mukaan, mitkä yksilöt pääsevät pariutumaan. Geneettinen ajautuminen on vähäistä isoissa populaatioissa, ja sen vaikutusta ei usein huomata. Populaatioihin kohdistuu paineita myös ulkopuolisten populaatioiden vaikutuksesta, jos eroavat populaatiot pääsevät uudestaan kosketukseen toistensa kanssa. Jos geenien vaihtelu populaatioiden välillä on tarpeeksi suurta, lopulta näiden populaatioiden geenipoolit muuttuvat samanlaisiksi (Choi ym. 2023).

2.2 Genetiikka ja geenit

Genetiikalla tarkoitetaan geenien ja perinnöllisyyden tutkimista. Lisääntymistilanteessa vanhempien ominaisuudet sekoittuvat keskenään seuraavalle sukupolvelle. Geenit ovat pieniä osia DNA-molekyylistä, joka pitää sisällään kyseisen organismin rakennusohjeet. Yhteen DNA-molekyyliin mahtuu useita tuhansia geenejä, esimerkiksi ihmisellä arvioidaan olevan noin 20 000 geeniä. Yhdessä geenit muodostavat genomin, joka pitää sisällään kaiken tarvittavan tiedon yksilön rakentamiseksi. (A brief guide to genomics.)

On tärkeää huomata, että vanhempien geenit eivät sekoitu niin, että peritty ominaisuus olisi sekoitus molempien vanhempien ominaisuuksia. Richard Dawkins esittää teoksessaan Maailman hienoin esitys: Evoluution todisteet (2009: 29) esimerkin siitä, että jos vanhempien geenit sekoittuisivat keskenään, olisimme kaikki vanhempiemme sekoituksia, ja näin ollen kaikki eroavaisuudet populaatiossa häviäisivät.

Mutaatiot ylläpitävät populaatioiden monimuotoisuutta. Mutaatiolla tarkoitetaan sellaista muutosta geenissä, mitä ei pystytä suoraan jäljittämään kumpaankaan vanhempaan. Mutaatio voi olla yksilölle hyödyllinen, haitallinen tai neutraali, mutta koska evoluutiolla ei ole päämäärää vaan se koettaa sopeutua ainoastaan vallitsevaan elinympäristöön, ovat mutaatiot sattumanvaraisia. Luonnonvalinnan tehtävänä on karsia luonnosta pois mutaatiot, jotka ovat yksilöille haitallisia. Ajan myötä nämä pienet muutokset voivat johtaa suuriinkin eroavaisuuksiin populaatioissa. (Heikkinen 2005.)

Voidaan ajatella, että evoluution ja luonnonvalinnan tehtävä onkin muokata yksilöitä kilpailukykyisemmiksi niiden oman populaation sisällä kuin muita lajeja vastaan. Leijona, joka on parempi metsästämään kuin saman populaation muut yksilöt, on todennäköisempi selviytymään ja lisääntymään. Luonnonvalinnan aiheuttama muutos ei välttämättä ole koko populaatiolle hyödyllinen, mutta parantaa yhden yksilön mahdollisuuksia selviytyä.

Richard Dawkins esittää teoksessaan esimerkin siitä, miksi puut ovat niin korkeita. Jokaisen yksilön ja koko metsän populaation kannalta olisi parempi, jos kaikki puut olisivat lyhyempiä. Näin kaikki säästäisivät energiaa, jonka ne voisivat käyttää selviytymiseen ja lisääntymiseen. Dawkins esittää tilanteen, jossa kaikki metsän puun sopivat, ettei ainutkaan yksilö kasva tiettyä pituutta korkeammaksi. Jos kaikki puut olisivat 10 metriä korkeita, se olisi yhtä tehokasta, kuin jos kaikki puut olisivat 100 metriä korkeita. Oletetaan, että yksi puu mutatoituu ja kasvaa esimerkiksi 11 metriä korkeaksi. Se joutuu käyttämään ylimääräistä energiaa kasvamiseen, mutta niin kauan, kuin muut puut eivät kasva aikaisemmin asetetun rajan yli, sen saama ylimääräinen energia korvaa sen ylimääräisen kasvun. Näin ollen luonnonvalinta suosii yksilöitä, jotka eroavat muusta populaatiosta. Ongelmaksi tulee, että vuosien saatossa muutkin puut kasvavat pidemmiksi ja kaikki joutuvat käyttämään kasvamiseen ylimääräistä energiaa, mutta kenenkään energiamäärä ei kasva. (Dawkins 2009: 379)

2.3 Luonnonvalinnan vaikutustavat

Luonnonvalinnan vaikutustavat jaetaan yleisesti kolmeen eri kategoriaan: suuntaava valinta, tasapainottava ja hajottava valinta. Näiden lisäksi voidaan puhua myös seksuaalivalinnasta eli sukupuolivalinnasta ja peto-saalisvalinnasta.

Suuntaava valinta ajaa populaatioita tiettyyn suuntaan, jos populaation elinympäristössä tapahtuu muutoksia. Elinympäristön muuttuessa populaatio voi kohdata uusia haasteita, jolloin luonnonvalinnan kautta populaation ominaisuudet muuttuvat soveltuvammiksi uuteen ympäristöön. (Buckley 2021.)

Tasapainottava valinta toimii elinympäristöissä, joihin populaatiot ovat jo sopeutuneet. Tällaisessa tilanteessa ääripään ominaisuudet häviävät. Tämän vuoksi saman populaation yksilöt ovat esimerkiksi hyvin usein hyvin samankokoisia. Erittäin pienet ja erittäin suuret yksilöt vähenevät, ja suurin osa yksilöistä on keskikokoisia. Tämän myötä populaation yksilöt voivat olla keskenään hyvin samannäköisiä.

Hajottavalla valinnalla tarkoitetaan tilannetta, jossa populaation ominaisuudet ajautuvat kohti eri ääripäitä. Hajottavaa valintaa voidaan pitää tasapainoittavan valinnan vastakohtana, jossa ääripään ominaisuudet yleistyvät. Tällainen tilanne johtaa usein populaatioiden lajiutumiseen, jos ne ovat erossa toisistaan tarpeeksi pitkään.

Seksuaalisessa valinnassa muutospaineen aiheuttaa populaatiossa toisen sukupuolen yksilö. Seksuaalisessa valinnassa muutospaineen alla olevat geenit vaikuttavat usein sellaisiin ominaisuuksiin, joilla ei ole selviytymisen kannalta merkitystä tai jotka ovat jopa haitallisia, mutta parantavat kyseisen yksilön mahdollisuuksia lisääntyä. Esimerkiksi riikinkukkouroksilla erittäin värikäs ja pröystäilevä pyrstö toimii naaraiden houkuttimena, mutta toisaalta se houkuttelee myös erittäin vahvasti mahdollisia saalistajia ja sen kasvatus ja kantaminen ovat energiaa vieviä ominaisuuksia.

Peto-saalisvalinnassa nimensä mukaisesti on kyse kahden eri lajin välisestä kilpailusta, jossa toinen on saalis ja toinen saalistaja. Saalistajan toiminta karsii saalispopulaatiosta esimerkiksi hitaimmat yksilöt, ja näin nopeammat yksilöt yleistyvät. Vastavuoroisesti tämä aiheuttaa myös saalistajille paineen olla nopeampia. Näin ollen molemmat lajit aiheuttavat toisilleen jatkuvaa painetta parantua. (Buckley 2021.)

3 Luonnonvalinnan hyödyntäminen tietokoneilla

3.1 Luonnonvalinnan hyödyntäminen ongelmanratkaisussa

Luonnonvalinnan liittämistä osaksi ongelmanratkaisua on esitetty jo 1940-luvulta lähtien. 1960-luvulla saatiin joitakin onnistuneita ohjelmia valmiiksi. Syntyi kolme erilaista, mutta samalla idealla olevaa toteutusta. Nämä kolme ovat evoluutio-ohjelmointi, geneettiset algoritmit ja evoluutiostrategiat. Myöhemmin kaikki ajautuivat erilleen, mutta kaikkia pidetään evoluutiolaskennan alalajeina. (Eiben & Smith 2015: 14.)

Geneettiset algoritmit ovat hakualgoritmeja, jotka käyttävät luonnonvalinnan mekanismeja hyödykseen ongelmanratkaisussa. Nämä algoritmit kehitti 1970-luvulla John Holland, jonka ideana oli tutkia adaptiivista käytöstä. Nykyajan geneettiset algoritmit eivät enää muistuta Hollandin kehittämää algoritmia, ja geneettiset algoritmit ovat kehittyneet pitkälle eikä niille ole nykypäivänä vahvaa yleisesti hyväksyttyä määritelmää. Geneettisten algoritmien tärkeimpiä sovelluskohteita ovat optimointiongelmat. (Eiben & Smith 2015: 14.)

3.2 Tekoelämä

Vaikka geneettiset algoritmit ovat hyvä tapa ratkaista optimointiongelmia, ne väistävät monia luonnollisen elämän periaatteita. Tärkeimpänä huomiona on se, että luonnollisen elämän täytyy monistaa itsensä ilman ulkoisia tekijöitä. Geneettisissä algoritmeissa lopullinen päätösvalta on systeemin luojalla, joka päättää, mitkä yksilöt ovat soveltuvimpia jatkamaan ongelmanratkaisua. (Adamatzky & Komosinski 2005: 5.)

1990-luvulla Steen Rasmussen kehitti ensimmäisenä Core War -tietokonepelin. Pelissä ideana on luoda Assembly-koodikielellä ohjelma, joka kilpailee toisen pelaajan luomaa ohjelmaa vastaan. Molemmat ohjelmat kilpailevat saatavilla olevasta muistista, ja tehtävänä on sammuttaa kilpailijan ohjelma. (Adamatzky & Komosinski 2005: 5.)

Vaikka Core War -pelissä ei ollut mahdollisuutena ohjelman aikana kehittää ohjelmaa, se loi pohjan tuleville ohjelmille, joissa kehittyminen oli mahdollista. Thomas Rayn kehittämä Tierra-ohjelma perustui samaan ideaan kuin Core War, mutta ohjelma pystyi muuttumaan ohjelman aikana ja menettämään ominaisuuksia tai lisäämään niitä. Ohjelman ytimessä oli idea siitä, ettei ohjelma voinut kaapata muiden hallitsemia muistipaikkoja, vaan se pystyi kirjottamaan ainoastaan vapaille muistipaikoille. Kun kaikki muistipaikat ovat varattuina, vanhimmat ohjelmat poistetaan, jotta uusille syntyy tilaa. Tierra-ohjelman innoittamana syntyi myöhemmin Avida-niminen ohjelma, joka oli edeltäjäänsä kehittyneempi. (Adamatzky & Komosinski 2005: 5.)

Tierran ja Avidan merkittävimpänä erona on niiden tapa hallita muistia. Tierrassa ohjelmat pystyivät lukemaan ja ajamaan muiden ohjelmien muistipaikkoja. Avidassa muistipaikat eivät ole muiden ohjelmien käytössä ja prosessoriaikaa jaetaan eri määrä eri ohjelmien kesken. Prosessoriaika määräytyy sen mukaan, kuinka hyvin jokainen ohjelma suoriutuu. Paremmille ohjelmille jaetaan enemmän prosessoriaikaa kuin huonoille ohjelmille. (Poliakov 2020: 7.)

3.3 Luonnonvalinta tietokonepeleissä

Luonnonvalintaa käsitteleviä tietokonepelejä on tuotettu historian aikana jonkin verran. Tietokonepelit kuitenkin miltei aina jättävät pois jonkin tärkeän osan luonnonvalinnan toimintaa. Monet pelit painottavat yksilöiden ulkoisten ominaisuuksien muuttamista, mutta suoraa perinnöllisyyttä yksilöiden välillä ei ole. Usein yksilöiden selviäminen on pelaajan toiminnan varassa (Leith ym. 2016).

Maxisin vuonna 2008 julkaisemassa Spore-pelissä (kuva 1) pelaajan tehtävänä on luoda organismi, jolla on useita erilaisia ominaisuuksia ja ne vievät omat ominaisuutensa seuraaville sukupolville. Spore pitää sisällään kaksi luonnonvalinnan toimintaa: ulkoisten ominaisuuksien muuttumisen ja ominaisuuksien periytymisen. Peli kuitenkin vaatii pelaajan puuttumista sen selviytymiseen, joten selviytymisen ei voida ajatella olevan kytkettynä sen ominaisuuksiin. (Leith ym. 2016)

Kuva 1. Sporen-editori, jossa pelaaja voi muokata omaa lajiaan (Lane 2018.).

Parhaiten luonnonvalinnan toimintaa ovat kyenneet mallintamaan pelit SimEarth (1990) (kuva 2) ja SimLife (1992) (kuva 3). Näiden pelien kehittäjänä on toiminut sama yritys, joka kehitti myöhemmin Spore-pelin. Sekä SimEarthissa että SimLifessa pelaaja pystyy hallitsemaan peliekosysteemiä, mutta pelit käsittelevät evoluutiota eri näkökulmista. SimEarth-pelissä pelaajan tehtävänä on muokata pelimaailman ominaisuuksia. Pelaajan siis muovaa ympäristöä haluamallaan tavalla ja katsoo, miten eläimet toimivat pelaajan päättämässä ympäristössä. SimLife-pelissä pelaaja muokkaa pelissä olevia eläimiä ja kasveja. Molemmat pelit käyttävät kolmea evoluution mekanismia: geneettinen perimä, niiden kyky selviytyä ympäristössä ja kyky lisääntyä. (Leith ym. 2016)

Kuva 2. Päänäkymä SimEarth-pelissä. Keskellä taustalla pelin päänäkymä ja edessä tietoa maailmasta. Vasemmalla työkalut maailman muokkaamiseen (Crowley 2019.).

Kuva 3. Päänäkymä SimLife-pelissä. Keskellä näkymä pelimaailmasta ja ylhäällä erilaisia asetuksia (Crowley 2020.).

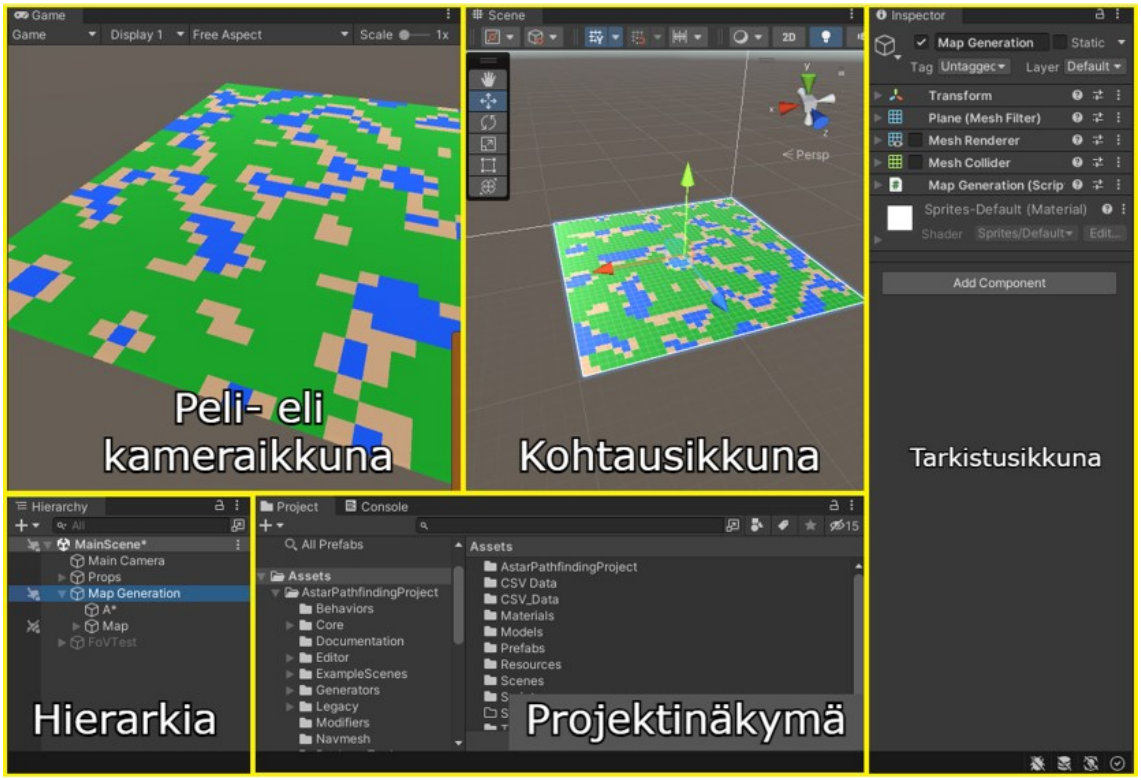
Hieman eri lähestymistapaa edustaa vuonna 2017 julkaistu Darwin’s Demons -niminen peli. Pelissä kehittyvänä osapuolena ovat pelaajan viholliset. Peli muistuttaa ulkoisesti tunnettuja Space Invaders -pelejä, joissa pelaaja taistelee kierros toisensa jälkeen omalla avaruusaluksellaan vihollisten avaruusaluksia vastaan, mutta vihollisten ominaisuudet muuttuvat kierrosten välissä sen mukaan, mitkä viholliset ovat pärjänneet pelaajaa vastaan parhaiten. Viholliset voivat saada uusia ominaisuuksia, kuten erilaisia aseita ja tapoja liikkua. Pelin edetessä vihollisista kehittyy monimuotoisia ja vaikeita vastustajia. Peli tarjoaa myös kokeilutilan, jossa pelaaja voi muokata, millä tavalla tietokoneen ohjaamat viholliset muuttuvat. (Darwin’s Demons Science Trailer 2017.)

4 Mallintamisen toteutus

4.1 Unity-pelimoottori

Insinöörityön pelin pohjaksi valikoitui Unity-pelimoottori lähinnä aikaisemman käyttökokemuksen pohjalta. Unity on pelimoottori, jolla voi tehdä 3D- ja 2D-pelejä. Ohjelmointia varten työssä käytettiin Visual Studio -ohjelmaa hyödyntäen C#-ohjelmointikieltä.

Unityn käyttöympäristö on hyvin toteutettu ja sopi hyvin työn toteuttamiseen. Laaja dokumentointi tarjosi hyvät mahdollisuudet opetella pelisovellusten kehittämistä. Unityn käyttöliittymä on helposti muokattavissa. Kuvassa 4 on esitetty Unityn päänäkymässä olevat tärkeimmät ikkunat. Peli-ikkuna näyttää kuvaa valitun kameran läpi katsottuna pelimaailmaan. Kohtausikkunaa käytetään pelin muokkaamiseen. Tarkistusikkunaa käytetään yksittäisten objektien säätämiseen. Tarkistusikkunasta löytyvät kaikki peliobjektiin liitetyt komponentit, ja peliobjektien muokkaus tehdään tätä kautta. Hierarkiaikkunassa näytetään jokainen kyseisessä kohtauksessa oleva peliobjekti ja sen rakenne. Projektinäkymässä nähdään kaikki projektiin lisätyt resurssit (asset). (Unity’s Interface.)

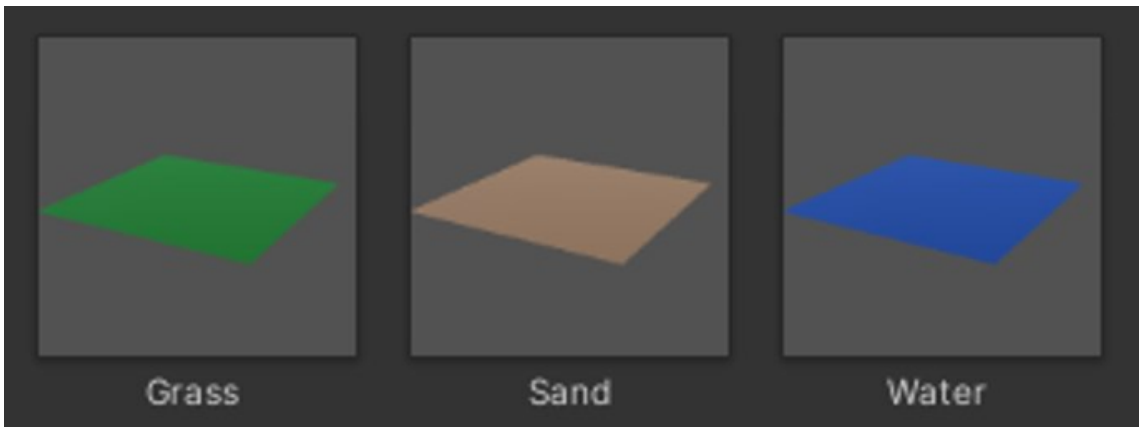


Kuva 4. Unity-pelimoottorin käyttöliittymä

4.2 Elinympäristö

Insinöörityössä eliöille luotu elinympäristö toteutettiin luomalla kolme erilaista laattaa (engl. Tile), joista pystyi käyttöliittymässä rakentamaan proseduraalisesti erikokoisia alueita. Eläimillä oli käytössä ruoho-, hiekka- ja vesialueita.

Kuvassa 5 on esitetty työssä käytetyt ruudut. Ruohoalueet toimivat jäniksille ruokana. Yksi ruoholaatta pystyi ruokkimaan täysin yhden jäniksen, minkä jälkeen sen takaisinkasvu kesti 30 sekuntia. Vesialueet olivat veden lähteenä molemmille lajeille, mutta toimivat myös elintilan rajoittimena. Työssä kumpikaan eläin ei osaa uida. Veden määrä on rajaton. Hiekka toimii rantana vesialueiden ympärillä eikä tarjoa jäniksille ruokaa.



Kuva 5. Kartan rakentamiseen käytetyt ruudut.

Alueen proseduraalinen luonti tehtiin luomalla kaksiulotteinen vektori, joka toimi koordinaatteina eri laatoille. Jokaisella koordinaatilla oli myös korkeusarvo, jonka perusteella määriteltiin, minkälainen laatta koordinaatteihin laitetaan. Ruoho-alueet toimivat korkeina alueina, jotka jäivät vesirajan yläpuolelle, vesialueet olivat rajan alapuolella, ja hiekka-alueet toimivat näiden välissä.

4.3 Lajit

Insinöörityössä tarkasteltiin jänisten ja kettujen välistä vuorovaikutusta. Lajien ominaisuudet eivät ole biologisesti tarkkoja, mutta pyrkivät kuvaamaan luonnonvalinnan toimintaa oikeassa maailmassa. Lajit eivät myöskään vastaa tosielämän jäniksiä ja kettuja muuten kuin nimellisesti. Molempien lajien mallinnus toteutettiin samalla tavalla, ja lajien erilaisuus näkyi käytännössä ominaisuuksien arvojen perusteella sekä joidenkin käyttäytymistä muuttavien seikkojen perusteella, jotka selitetään paremmin luvussa 4.5.

4.4 Geenit

Lajien geenit toteutettiin yksinkertaisesti käyttämällä desimaaliarvoja kuvaamaan jonkin tietyn piirteen intensiteettiä yksilössä. Geenien arvot oli alussa määritelty sopiviin arvoihin, ja ohjelman käynnistyessä ensimmäisen sukupol-

ven yksilöille arvottiin määriteltyjen arvojen ympäriltä ensimmäiset arvot, joita ohjelma lähtee luonnonvalinnan kautta muokkaamaan.

Geenien muuttuminen tapahtuu koodissa mutaation kautta. Koodissa mutaation todennäköisyys on asetettu 50 prosentin kohdalle simulaation nopeuttamiseksi. Luonnossa mutaation todennäköisyys on kuitenkin huomattavasti pienempi ja luonnonvalinta on paljon pitkäkestoisempi prosessi.

Kummankin lajin yksilöille luodaan omat geeninsä, jotka toimivat simulaatiossa yksilön perimänä. Mallinnuksessa käytettäviä ominaisuuksia, joihin erilaiset geenit vaikuttavat, ovat koko, kävelynopeus, juoksemisnopeus, näkökentän säde, näkökentän laajuus, yksilön herkkyys syödä, juoda tai lisääntyä, raskausaika ja pesueen koko.

Yksilöiden ikää mitataan yksinkertaisesti kokonaislukuna, ja sen tehtävänä simulaatiossa on luoda enimmäisaika, kuinka pitkään yksi yksilö pystyy elämään. Hajontaa maksimi-ikään luotiin käyrällä, joka nostaa yksilön todennäköisyyttä kuolla tietyn iän jälkeen. Tällä pyritään myös mallintamaan sitä, ettei jokainen yksilö oikeasti kuole samanikäisenä, vaan näennäisesti sattumalta yksilö voi vanhetessaan saada esimerkiksi jonkin kuoleman aiheuttavan taudin.

Lajien keskeisin ominaisuus on maksimienergian määrä, joka määräytyy yksilön koon mukaan. Energiankulutus lasketaan kaavalla

$$X = koko^{0.75} * \big(g * (1 + pesueenkoko * 0.3)\big) + nopeus/2$$

jossa

- X on energian kulutus.
- Koko on yksilön koko.
- g on 0 tai 1 riippuen siitä, onko yksilö raskaana.
- Pesueen koko on raskaana olevan yksilön jälkeläisten määrä.
- Nopeus on yksilön nopeus laskentahetkellä.

Kaava ei perustu tosimaailmaan, mutta pyrkii sopivalla tavalla määrittämään yksilön hetkellisen energiankulutuksen. Ensimmäinen luku, koko$^{0.75}$, perustuu Kleiberin lakiin, joka kuvaa perusaineenvaihduntaa. Kleiber huomasi 1930-luvulla, että suurin osa eläimistä noudattaa perusaineenvaihdossa kaavaa, jossa perusaineenvaihto kasvaa suhteessa yksilön massaan nostettuna ¾ potenssiin (Savage ym. 2004: 265). Yksilö kuolee, jos sen energia laskee nollaan.

Yksilön näkökentän pituus ja laajuus vaikuttavat yksilön kykyyn nähdä maailmassa ruokaa, vettä, kumppanin tai mahdollisen saaliin tai saalistajan. Näkökentän koolla ei sinänsä ole negatiivista vaikutusta, joka rajaisi sille mahdollisen maksimiarvon ympäristössä, mutta testeissä huomattiin, että jäniksillä liian suuri näköalue aiheutti panikointia, kun yksilöt huomasivat saalistajan kaukaa ja pyrkivät väistämään sitä. Ominaisuus piti yksilön elossa, mutta koko ajan varpaillaan, eikä yksilö kyennyt kunnolla syömään tai lisääntymään.

Yksilön lisääntymiseen vaikuttavia ominaisuuksia ovat raskausaika ja pesueen koko. Raskausaika vaikuttaa ainoastaan naaraspuolisilla yksilöillä, mutta geeni kulkee mukana myös urospuolisilla yksilöillä ja voi näin kulkeutua jälkipolville. Raskausaika määrittää, kuinka kauan naaraspuolinen yksilö on raskaana. Raskausajan pituus ei vaikuta jälkeläisten kehitykseen, mutta insinöörityössä huomattiin, että pitkä raskausaika luo paineen naaraalle, koska raskauden aikana yksilö kuluttaa enemmän energiaa. Tämän perusteella oli odotettavissa, että pitkä raskausaika luo otollisemman tilanteen pienille pesueille. Tilanteessa on ongelmallista, että suuri pesueen koko ja lyhyt raskausaika eivät aiheuta populaatiolle ongelmia, vaan on oletettavaa, että populaatio muuttuisi tähän suuntaan. Kun raskausaika on pitkä, on odotettavissa, että pesueen koko pysyy pienenä, jotta energiankulutus ei nouse liiallisesti.

Eläinten nälkä ja jano mallinnettiin työssä käyttämällä desimaaliarvoja nollan ja sadan väliltä. Nälkä kasvaa suhteessa senhetkiseen energiamäärään. Pieni energian määrä kasvattaa nälkää nopeampaa tahtia. Jano taas lisääntyy kaikilla yksilöillä samaa tahtia. Yksilö kuolee, jos jano tai nälkä ylittää maksimiarvon.

Mallinnuksessa yksilöiden perinnöllisyys toteutettiin käyttämällä Genes-luokkaa, jossa jokaisen yksilön eri geenit on määritelty. Lisääntymistilanteessa molempien vanhempien geenit jaetaan puoliksi uudelle yksilölle satunnaisesti, kuitenkin niin, että puolet geeneistä tulee aina toiselta vanhemmalta.

Esimerkkikoodissa 1 metodille annetaan tieto molempien vanhempien geeneistä. Jälkeläiselle jaetaan satunnaisesti geenit jommaltakummalta vanhemmalta. Samalla myös arvotaan, tuleeko geeniin mutaatiota vai ei. Jos mutaatiota tulee, se muuttaa geenin arvoa johonkin suuntaan. Geenit jaetaan aina tasan, vaikka oikeasti jokaisella organismilla on kaikki geenit kummaltakin vanhemmalta, mutta jokaisen geenin kohdalla vain toinen versio on aktiivinen. (Genetics.)

```
public void RandomizeGenes(Genes mother, Genes father)
{
     int fromMother = 0;
     for (int i = 0; i < mother.attributes.Length; i++)
     {
          float chance = (((float)mother.attributes.Length / 2) -
               fromMother) / ((float)mother.attributes.Length - i);
          if (rand.NextDouble() <= chance && chance != 0)
          {
               attributes[i].value = mother.attributes[i].value; ;
               if (rand.Next(1, 101) < mutationChance)
               {
               attributes[i].value += UnityEngine.Random.Range(-
                                      maxMutation, maxMutation);
               }
               fromMother++;
          }
          // if not, gene is inherited from father
          else
          {
               attributes[i].value = father.attributes[i].value;
               if (rand.Next(1, 101) < mutationChance)
               {
                    attributes[i].value += UnityEngine.Random.Range(-
                                           maxMutation, maxMutation);
               }
          }
     }
}
```

Esimerkkikoodi 1: Metodi jakaa satunnaisesti molemman vanhemman geenit puoliksi uudelle jälkeläiselle ja luo mahdollisen mutaation geeneihin.

4.5 Tekoäly

Vaikka tekoälyllä ei ole vahvaa vakiintunutta määritelmää, yleisesti sillä tarkoitetaan tietokoneen kykyä suorittaa näennäisesti älykkyyttä vaativia toimintoja tai valintoja (Bourg & Seemann 2004.).

4.5.1 Käytöspuu

Insinöörityössä eläinten tekoäly luotiin käyttämällä käytöspuutekoälytekniikaa (engl. behavior tree). Käytöspuu koostuu hierarkkisista solmuista, jotka ohjaavat tekoälyn päätöksenteon kulkua. Puun ensimmäistä solmua kutsutaan juureksi. Puun solmut haarautuvat haaroihin, joista jokainen päättyy lehteen. Lehtien tehtävänä on suorittaa varsinaiset tekoälyä vaativat toiminnot. (Simpson 2014.)

Jos solmu ei ole lehti, eli puun haaran viimeinen solmu, se voi olla jompikumpi kahdesta mahdollisesta solmusta: valitsija tai sarja. Muita mahdollisia solmuja voivat olla rinnakkainen tai koriste. Jälkimmäistä kahta solmutyyppiä ei tässä työssä käytetty. Puunhaaran viimeiset solmut voivat olla tehtävä tai ehto. (Ogren 2012.)

Valinta- ja sarjasolmut käyvät läpi kaikki niihin liitetyt lapsisolmut. Jokainen läpikäyty solmu palauttaa vanhemmalleen onkin kolmesta mahdollisesta tilasta, jonka perusteella käytöspuu jatkaa suorittamistaan tai lopettaa senhetkisen läpikäymisen ja aloittaa alusta. Mahdollisia paluutiloja ovat onnistumine, juokseva ja epäonnistuminen. (Ogren 2012.)

Esimerkkikoodissa 2 on esitetty Valintasolmu-luokan toimintaperiaate pseudokoodina. Valintasolmu etsii ja valitsee ensimmäisen lapsisolmun, jonka paluuarvo on onnistuminen. Valintasolmu palauttaa välittömästi vanhemmalleen joko success tai running, kun joku sen lapsisolmuista palauttaa success tai running. Valintasolmut eivät tässä tapauksessa käy läpi jäljellä olevia lapsisolmuja. Lapset käydään läpi niiden tärkeyden perusteella. Jos kaikki lapsisolmut epäonnistuvat, myös valintasolmu epäonnistuu.

```
SELECTOR (pseudocode)
foreach(Node node in children)
{
     switch (node.Evaluate())
     {
          case FAILURE:
               continue;
          case SUCCESS:
               return SUCCESS;
          case RUNNING:
               return RUNNING;
     }
}
return FAILURE;
```

Esimerkkikoodi 2: Selector-luokan toimintaperiaate.

Sarjasolmu ajaa järjestyksessä kaikki sen lapsisolmut. Toisin kuin valintasolmu, joka lopettaa suorittamisen, jos jokin lapsista onnistuu, sarjasolmu epäonnistuu, jos yksikin sen lapsista epäonnistuu (esimerkkikoodi 3). Sarjasolmun tehtävän on käydä läpi lapsisolmuja niin kauan, kunnes sen lapsisolmut palauttavat success tai running. Siinä missä valintasolmulle ei ole merkitystä, palautuuko sen lapsisolmusta success vai running, sarjasolmulle running kertoo, että jokin sen lapsisolmuista on kesken, eikä kyseisen solmun tila ole vielä success tai failure. Esimerkiksi liikkumiseen vaikuttava solmu voi olla vielä matkalla, eli näin ollen solmu ei ole vielä onnistunut tai epäonnistunut.

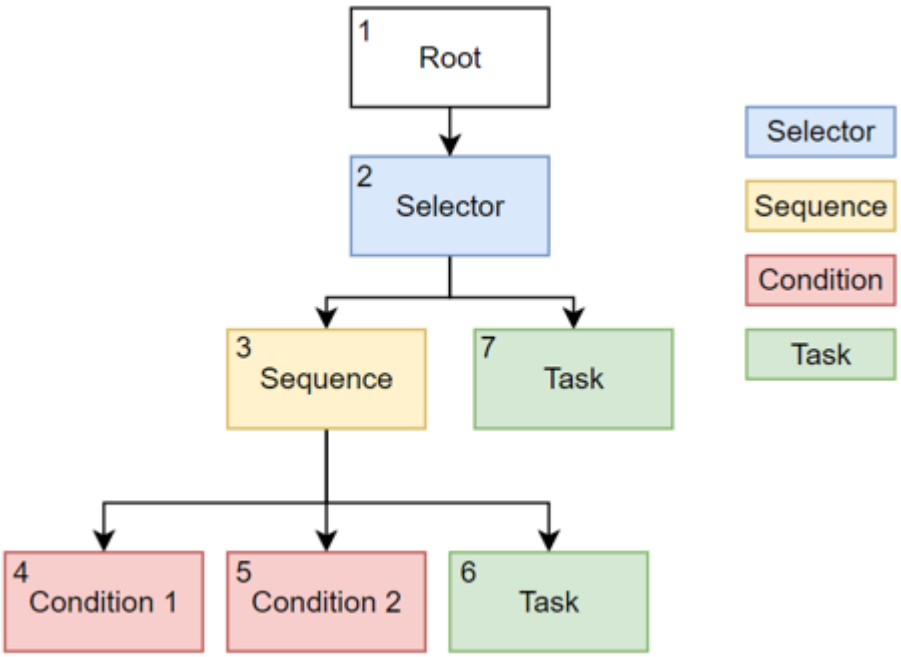
```
SEQUENCE (pseudocode)
foreach(Node node in children)
{
     switch(node.Evaluate())
     {
          case FAILURE:
               return FAILURE;
          case SUCCESS:
               continue;
          case RUNNING:
               isAnyChildRunning = true;
               return RUNNING;
     }
}
return isAnyChildRunning ? RUNNING : SUCCESS;
```

Esimerkkikoodi 3: Sequence-luokan toimintaperiaate.

Tehtäväsolmut nimensä mukaisesti suorittavat tekoälyn vaatimia tehtäviä. Solmu palauttaa success, jos tehtävä on suoritettu onnistuneesti, failure, jos tehtävän tekeminen ei ole mahdollista, tai running, jos tehtävä on kesken.

Ehtosolmujen tehtävänä on asettaa solmujen läpikäymiselle erilaisia ehtoja. Ehtosolmujen paluuarvo ei voi koskaan olla running, vaan ne ovat aina success tai failure.

Käytöspuun läpikäyminen aloitetaan juurisolmusta, ja sitä lähdetään käymään läpi ylhäältä alaspäin, vasemmalta oikealle. Jokainen solmu käy läpi sen lapsisolmut. Yhtä käytöspuun läpikäyntiä kutsutaan iteraatioksi.



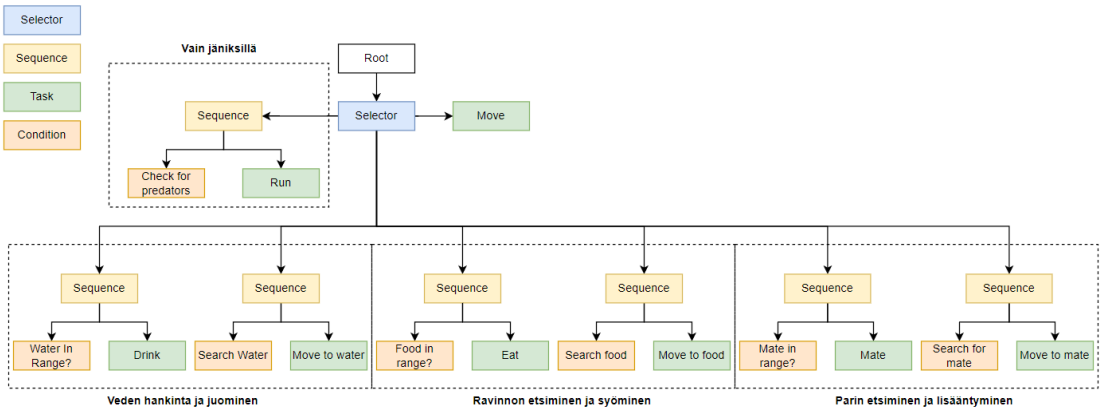
Kuva 6. Esimerkki käytöspuusta. Vasemmassa ylänurkassa solmujen läpikäymisjärjestys.

Kuvan 6 tapauksessa käytöspuun läpikäynti aloitetaan Root-solmusta (solmu 1). Root-solmu suorittaa Selector-lapsisolmun (solmu 2). Selector aloittaa suorittamalla Sequence-solmun (solmu 3). Sequence-solmu siirtyy suorittamaan vasemmanpuoleisen Condition 1 -solmun (solmu 4). Jos Condition 1 palauttaa onnistumisen, siirtyy Sequence-solmu suorittamaan Condition 2 -solmua (solmu 5). Jos myös Condition 2 palauttaa onnistumisen, Sequence-solmu siirtyy toteuttamaan Task-solmua (solmu 6). Task-solmu voi tilanteesta riippuen palauttaa success, running tai failure. Jos paluuarvo Task-solmusta on success tai running, palauttaa Sequence-solmu vastaavan arvon Selector-solmulle. Näin ollen käytöspuu on suoritettu onnistuneesti. Tilanteessa, jossa jokin Sequence-

solmun lapsista palauttaisi arvon failure, koko Sequence epäonnistuu ja Selector-solmu siirtyisi suorittamaan Task-solmua (solmu 7). Jos Task-solmu palauttaa success tai running, palautuu Selector onnistuneesti. Palautuksella failure Selector epäonnistuu, ja näin ollen koko käytöspuu palautuu epäonnistuneesti.

4.5.2 Käytöspuun toteutus työssä

Tässä luvussa käydään läpi, miten työssä toteutettiin erilaiset käyttäytymismallit. Kummankin lajin käytöspuu rakennettiin samalle pohjalle. Kummankin lajin solmut ovat samoja, mutta jäniksille rakennettiin kaksi ylimääräistä solmua saalistajien etsimistä ja pakenemista varten (kuva 7). Kaikki alarivin kolme ryhmää rakennettiin tavallaan väärin päin, jotta hakutehtäviä ei tarvitse suorittaa, jos se on jo onnistuneesti suoritettu. Näin esimerkiksi juomista voidaan jatkaa ilman, että joka iteraatiolla tarvitsee tarkistaa, että on liikuttu veden lähelle.



Kuva 7. Lajien käytöspuun rakenne.

4.5.3 Veden etsiminen ja juominen

Kummankin lajin veden hankinta toteutettiin työssä samalla tavalla. Kuvasta 7 nähdään, että kettujen tärkein tehtävä on hankkia vettä. Jäniksillä vain pakeneminen menee juomisen edelle. Kohdassa ”Water in range?” tutkitaan, onko yksilö tarpeeksi lähellä mahdollista vesilaatta, jotta juominen voidaan aloittaa. Koh-

dassa ehtona on myös se, ettei kyseinen yksilö ole parhaillaan syömässä tai lisääntymässä. Jotta solmu voi onnistua, on puun ensin käytävä vähintään kerran Search water -solmussa, jossa määritellään mahdollinen vesilaatta, josta yksilö voi juoda. Search water -solmussa yksilö etsii näköetäisyydeltä lähimmän vesilaatan ja tallentaa sen. Solmussa on ehtoina, ettei yksilö ole tekemässä muuta ja että yksilön jano on ylittänyt geeneissä määritellyn rajan sille, että yksilö alkaa etsimään vettä. Alue, jota yksilö tutkii, määritellään yksilön geenien perusteella. Alue on ympyrän muotoinen ja sen säde johdetaan eläimen geenistä näkökentän säde. Alue tutkitaan ohjelmassa kerralla eikä pään kääntämistä ole otettu huomioon. Jos vesilaatta löydetään, se lisätään myös eläinten muistiin, ja sitä voidaan käyttää myöhemmin tilanteissa, joissa yksilö ei löydä ympäriltään vettä, mutta on lähellä kuolla janoon. Tällä simuloidaan yksilöiden muistia ja sitä, etteivät vesialueet luonnosta häviä hetkessä ja yksilö muistaa niiden sijainnit.

Kun yksilö on löytänyt ympäriltään vettä, Search Water -solmu siirtyy suorittamaan Move to water -solmua. Move to water -solmu palauttaa running-tilaa, kunnes yksilö on päässyt tarpeeksi lähelle vettä ja juominen voidaan aloittaa. Juominen alkaa, kun käytöspuuta lähdetään käymään uudestaan läpi.

4.5.4 Ravinnon etsiminen ja syöminen

Koska kummankin eläimen ravinnonhankinta perustuu eri ravinnonlähteeseen, on lajien ravinnon hankinnassa pieniä eroja, mutta perusidea kummassakin on sama. Jos vedenhankinta ei ole tällä hetkellä tarpeellista, siirtyy käytöspuu käymään läpi Ravinnon etsiminen ja syöminen -kohtaa. (Kuva 7.)

Ensimmäisenä solmussa ”Food in range?” tutkitaan, onko yksilöllä tiedossa ruoka, jota se voi syödä, tai että yksilö ei ole pariutumassa. Tässä solmussa ei oteta huomioon, onko yksilöllä tarpeeksi kova nälkä, koska yksilöllä ei voi olla tiedossa mahdollista ravintoa, jos nälkä ei ole tarpeeksi kova. Jos tiedossa on ruoka ja se on tarpeeksi lähellä, siirtyy käytöspuu suorittamaan Eat-tehtävää. Jäniksien tapauksessa mahdollinen ruoka on ruoholaatat. Ketuille taas jänikset

toimivat ravintona, mutta molempien logiikka etsimisessä ja liikkumisessa on sama.

Jos ruokaa ei ole syöntietäisyydellä, ensimmäinen sequence-solmu epäonnistuu ja siirrytään suorittamaan seuraavaa sequence-solmua. Search food -solmussa tutkitaan, onko näköetäisyydellä mitään ruuaksi kelpaavaa. Hakuprosessi toimii samalla tavalla kuin vettä etsiessä. Hakualue on määritelty näkökentän säteen perusteella, ja tutkiminen tapahtuu joka puolelle yhtäaikaa.

Koska jänikset toimivat ketuille ruokana, on jäniksille tehty ylimääräinen Sequence-solmu, joka ottaa prioriteetin kaiken muun yläpuolelle. (Kuva 7.) Jokaisen iteraation alussa jänikset tarkastavat näkökenttänsä mahdollisia saalistajia varten. Tässä tarkastuksessa otetaan huomioon, mihin suuntaan jänis on katsomassa. Tämän myötä ketuilla on mahdollisuutena lähestyä jäniksiä takaapäin ilman, että jänis huomaa kettua. Jänisten näkökenttä määräytyy näkökentän säteen ja näkökentän laajuuden mukaan. Jos jänis huomaa edessään saalistajan, se pyrkii pakenemaan kaikkia saalistajia, jotka se on huomannut, valitsemalla suunnaksi sen, missä ei ole mahdollisia saalistajia. Tämä saattaa kuitenkin johtaa ongelmallisiin tilanteisiin, jos jänistä lähestyy saalistajia tasaisesti joka suunnasta. Jokainen jänis tallentaa muistiin mahdolliset lähellä olevat saalistajat, ja ne poistetaan muistista, jos jänis pääsee tarpeeksi kauas kyseisestä saalistajasta.

Mikäli eläin pääsee tarpeeksi lähelle, syömisen kesto määritellään Time to eat -nimisellä desimaalimuuttujalla. Simulaatioissa oletusarvona käytettiin viittä sekuntia. Syöminen palauttaa yksilölle energiaa ja vähentää nälkää. Työssä pyrittiin tilanteeseen, jossa yksi ruokailu pystyisi palauttamaan yksilön tilanteeseen, jossa sillä on maksimimäärä energiaa ja nälkä nolla, kun maksimienergian määrä on 100. Kun syöminen on suoritettu loppuun, ilmoitetaan pelille kuolevasta jäniksestä ja jäniksen kuolinsyyksi kirjataan ”syöty”. Jänisten syömistä ruoholaatoista ei pidetä kirjaa.

4.5.5 Lisääntyminen ja perinnöllisyys

Tilanteessa, jossa käytöspuu palauttaa Selector-solmulle juomisesta ja syömisestä failure-tilan, voidaan tutkia, onko lisääntyminen mahdollista. Eläinten lisääntymistä ohjaa niiden Reproductive Urge -muuttuja. Reproductive Urge on kokonaisluku 0:n ja 100:n väliltä. Geenien ohjaama lisääntymisraja määrittää, missä kohtaa yksilö pyrkii lisääntymään. Kun Reproductive Urge ylittää lisääntymisrajan määrittämän rajan, yksilö pyrkii lisääntymään.

Lisääntyminen aloitetaan katsomalla, onko yksilöllä jo lisääntymiseen soveltuva kumppani solmussa ”Mate in range” (kuva 7). Mikäli kumppania ei ole vielä löydetty, palauttaa solmu failure-tilan ja siirrytään etsimään mahdollista kumppania. Mallinnuksen yksinkertaistamiseksi ohjelma noudattaa kaavaa, jossa ainoastaan urospuoliset yksilöt etsivät kumppania. Urokset tutkivat ympäriltään aluetta, jonka määrittää myös tässä tapauksessa yksilön näkökentän säde. Uros lähettää kaikille löydetyille naaraspuolisille yksilöille pyynnön parittelusta. Naaras vastaa kutsuun joko hyväksymällä tai hylkäämällä ehdotuksen. Ehtoina naaraalla on, ettei se ole valmiiksi raskaana, ettei sen Reproductive Urge ole ylittänyt lisääntymisrajaa tai ettei se ole tekemässä jotain muuta. Näissä tapauksissa naaras hylkää pyynnön automaattisesti. Jos naaraan Reproductive Urge on suurempi kuin sen jano tai nälkä, se hyväksyy uroksen pyynnön.

Tapauksissa, joissa naaras hylkää uroksen pyynnön, lisätään naaras uroksen rejections-listalle. Rejections-listalla oleville naaraille uros ei lähetä pyyntöä, ennen kuin naaras poistuu listalta. Insinöörityössä unohtamiseen käytettiin viittä sekuntia. Näin säästetään hieman laskentatehoa, kun samaa naarasta ei tarvitse pyytää joka kerta uudestaan.

Lisääntyminen voidaan aloittaa, jos naaras hyväksyy uroksen pyynnön. Tällöin voidaan käytöspuussa siirtyä eteenpäin kohtaan ”Move to mate”, jonka tehtävä on liikuttaa yksilöt toistensa viereen. Kun yksilöt ovat saavuttaneet toisensa, voidaan lisääntyminen aloittaa. Lisääntymisen ajaksi päätettiin 3 sekuntia. Tämän aikana yksilöt eivät voi liikkua. Lisääntymisen päätteeksi naaraspuolinen

yksilö aloittaa raskauden. Raskauden pituus määräytyy geenillä raskausaika. Työssä molemmille lajeille asetettiin alussa raskausajaksi arvo väliltä 19,5 ja 20,5. Ennen raskautta myös määritellään tulevan pesueen koko, koska se vaikuttaa raskaana olevan naaraan energiankäyttöön. Pesueen koko määräytyy pesueen koko -geenin perusteella, kuitenkin niin, että pesueen koossa voi olla vaihtelua kahden yksilön verran suuntaan tai toiseen. Alussa jänisten pesueen kooksi asetettiin arvo väliltä 3,5 ja 4,5. Ketuilla arvo oli väliltä 1,5 ja 2,5.

Tilanteissa, joissa yhtäkään käytöspuun lehteä ei pystytä suorittamaan, suoritetaan viimeisenä mahdollisuutena Move-solmu. Solmun tehtävän on luoda yksilölle tekemistä ja kuljettaa sitä uusiin paikkoihin. Solmun logiikassa ongelmallista on se, että se valitsee paikan, johon liikkua, satunnaisesti yksilön ympäriltä, mikä voi johtaa päättömään pyörimiseen samalla alueella.

4.6 Navigaatio

Navigaatiolla tarkoitetaan tapoja, miten pelin hahmot löytävät tiensä maailmassa. Tähän ongelmaan on kehitetty useita erilaisia algoritmeja. Insinöörityössä käytettiin Aron Granbergin kehittämää A*-reitinhakualgoritmi-lisäosaa (lausutaan A-tähti tai englanniksi A-Star) (A* Pathfinding Project). Työssä päädyttiin käyttämään tätä Unityn oman NavMeshin sijaan, koska proseduraalisesti luodussa kartassa ei luoda kartalle omaa meshiä. Meshin luomisen sijaan tyydyttiin käyttämään tarjolla olevaa lisäosaa, koska siinä pystyttiin rakentamaan navigointi käyttäen pisteitä kartalla. Seuraavaksi käsitellään, miten työn pohjana käytetty A*-reitinhakualgoritmi toimii ja miten sitä käytettiin työssä

4.6.1 A*-algoritmi

Kuten edellä mainittiin, työssä käytettiin A*-reitinhakualgoritmi-lisäosaa. Lisäosa käyttää vuonna 1968 kehitettyä A*-reitinhakualgoritmia, joka laajentaa Dijkstran vuonna 1959 kehittämää reitinhakualgoritmia. Suurin eroavaisuus näiden kahden algoritmin välillä on se, että A* käyttää heuristista etsintälogiikkaa, jossa

reitinetsintä yritetään suorittaa mahdollisimman nopeasti. A* pyrkii löytämään ainoastaan yhden lyhyimmän reitin aloituspisteen ja määritellyn maalin välille. Djikstran algoritmi pyrkii tutkimaan aluetta tasapuolisesti joka suuntaan, kun taas A* ottaa huomioon tutkittavan pisteen etäisyyden maalista. (Zeng & Church 2007.)

Reitinhaku aloitetaan lähtöpisteestä ”paras ensin” -menetelmällä, jossa kahdeksan lähintä pistettä arvioidaan ja niistä valitaan paras mahdollinen. Tässä vaiheessa pitää ottaa huomioon, että paras piste ei välttämättä johda lyhyimpään matkaan maaliin. Pisteen arvoon otetaan myös huomioon, kuinka monen siirron päässä se on alkupisteestä. Kun tätä tapaa verrataan Djikstran algoritmiin, jossa alkupisteestä lähdetään tasaisesti lähintä avonaista pistettä kohti, on selvää, että A*:n nopeus on huomattavasti parempi.

A*:n reitinhaku perustuu pisteiden välimatkoihin, sekä maalin että aloituspisteeseen. A* käyttää A*-algoritmeille yleistä kaavaa

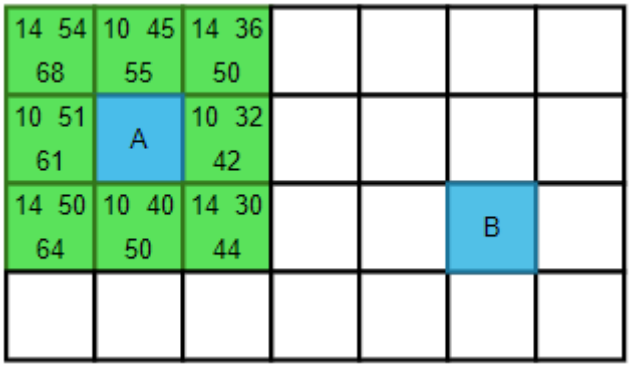
$$f(n) = g(n) + h(n)$$

jossa

- n on seuraava tutkittava piste.
- g(n) on pisteeseen liikuttavan matkan arvo aloituspisteestä.
- h(n) on maaliin arvioitu matka.
- f(n) on välimatkojen summa.

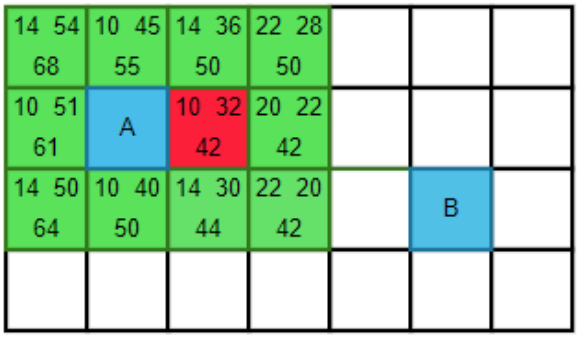
Seuraavaksi käydään läpi esimerkki A*-algoritmin toiminnasta. Kuvasta 8 näkyy reitinhaun alkutilanne. Tavoite on päästä ruudusta A ruutuun B. Jokaiselle ruudulle on määrätty koordinaatit, joiden mukaan ruutuja lasketaan. Algoritmin on mahdollista edetä vihreällä pohjalla merkittyihin ruutuihin. Ruutujen vasemmassa yläkulmassa on aloituspisteestä ruutuun kuljetun matkan arvo g. Yhden ruudun arvo on 10, ja poikittain kuljetun matkan arvo saadaan Pythagoraan lauseen mukaan kaavalla $a^2 + b^2 = c^2$. Poikittain kuljetun matkan arvo on pyöristetty

alaspäin yksinkertaistamisen vuoksi. Oikeassa yläkulmassa on maaliin arvioitu matka h. Molemmat arvot perustuvat euklidiseen matkaan, joka lasketaan kaavalla $h(n) = \sqrt{(x_n - x_v)^2 + (y_n - y_v)^2}$, jossa n on tutkittava ruutu ja v on ruutu, johon matkaa verrataan, joko alku- tai loppuruutu. (x, y) edustaa ruudun koordinaatteja. Keskellä alhaalla on näiden arvojen summa eli ruudun heuristinen arvo f. Ensimmäisenä askeleena on laskea kaikille kahdeksalle viereiselle ruudulle f-arvo. Kuvasta huomataan, että ruudut, jotka ovat vasemmalla A-ruudusta, saavat suurempia f-arvoja, koska ne ovat väärässä suunnassa ruutuun B.



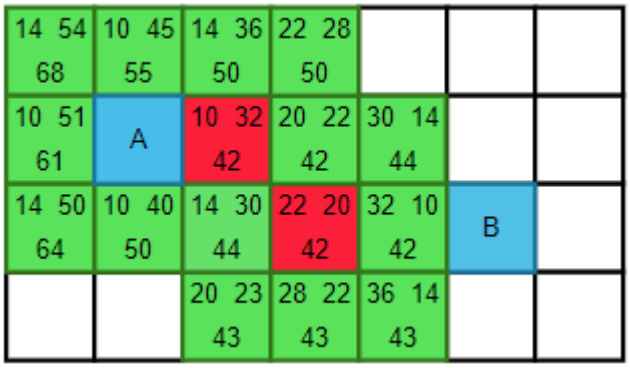
Kuva 8. Reitinhaun alkutilanne.

Ensimmäisenä lähdetään tutkimaan aloitusruudun A oikealla puolella olevaa ruutua, jonka arvoksi on laskettu 42, koska sen f-arvo on pienin (kuva 9). Kuvassa tutkittava ruutu merkittiin punaisella. Tästä eteenpäin algoritmi lähtee käymään läpi mahdollisia ruutuja ja valitsee joka kerta niistä pienimmän mahdollisen f-arvon. Kuvan 9 tilanteessa algoritmilla on kaksi mahdollista ruutua edetä. Tasapelitilanteet voidaan ratkaista usealla eri tavalla, esimerkiksi painottamalla ruutua, jolla on pienempi g- tai h-arvo. Esimerkissä painotetaan ruutua, jolla on pienempi g-arvo, eli oletettu matka maaliin on lyhyempi.



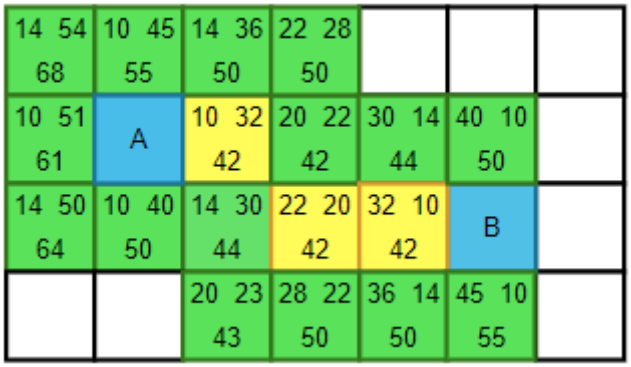
Kuva 9. Reitinhaun ensimmäinen askel

Tilanteessa, jossa uuteen ruutuun saavuttaessa ei löydy uusia mahdollisia ruutuja tai seuraavien ruutujen f-arvo on nykyistä ruutua pienempi, on saavuttu umpikujaan. Tässä tilanteessa siirrytään tutkimaan ruutua, jolla on seuraavaksi pienin f-arvo. Tässä esimerkissä tilannetta ei tule vastaan. Algoritmi voikin esteettä jatkaa reitin hakemista seuraavaan ruutuun (kuva 10).



Kuva 10. Reitinhaun toinen askel.

Algoritmia toistetaan, kunnes päästään tilanteeseen, jossa senhetkinen käsiteltävä ruutu on maaliruudun B vieressä. Tässä vaiheessa A* rakentaa lyhyimmän mahdollisen reitin läpikäydyistä ruuduista. Reitti rakennetaan kulkemalla läpikäyty reitti päinvastaisessa järjestyksessä takaisin lähtöruutuun. Kun reitti on valmis, se voidaan antaa tekoälyn käytettäväksi (kuva 11).



Kuva 8. Valmis reitti. Keltaisella reitti, jonka reitinhaku valitsee.

4.6.2 Reitinetsintä työssä

Kuten luvun 4.6.1 alussa mainittiin, työssä hyödynnettiin Unitylle saatavilla olevaa ilmaista lisäosaa, jonka mukana tulee sekä A*-reitinhaku että mahdollisuus käyttää sitä tekoälyn ohjaamiseen. Reitinhaku alustetaan luomalla koko kartan kattava graafi. Lisäosa tarjoaa ilmaiseksi useita erityyppisiä graafeja, joita ovat ruudukko, navigaatioverkko lyhemmin NavMesh ja pistegraafi. Tähän työhön valittiin käyttöön ruudukko sen keveyden ja yksinkertaisuuden vuoksi. Ruudukon päivittäminen on nopeaa, mikä sopi hyvin työhön, koska työn edetessä kokeiltiin erilaisia karttamalleja. Maksullinen versio tarjoaa muutamia hieman kehittyneempiä graafeja, mutta niitä ei käsitellä tässä raportissa. (Graph types 2023.)

Jotta reitinhakua voidaan käyttää yksilöiden ohjaamiseen, täytyy jokaisella yksilöllä olla Seeker- ja AIPath-komponentit. Seeker-komponentin tehtävänä on hoitaa reitinhaun kutsut ja muokata ne yksilölle sopiviksi. AIPath-komponentin tehtävänä on suorittaa itse liikkuminen. Komponentin käyttö ei ole pakollista, mutta itse yksilöiden liikuttaminen ei ollut työssä oleellinen osa, joten tyydyttiin käyttämään valmiiksi saatavilla olevaa ominaisuutta.

5 Mallintamisen onnistuminen

Tekoälyn onnistumiselle voidaan asettaa sen tarkoituksen mukaan erilaisia vaatimuksia. Peleissä tekoälyä usein arvioidaan sen perusteella, onko tietokoneen toiminta verrattavissa ihmisen toimintaan. Daniel Livingstone esittää työssään Turing’s Test and Believable AI in Games kolme kriteeriä hyvään tekoälyn toimintaan. Nämä kriteerit ovat suunnitelma, toiminta ja reagointi. (Livingstone 2006: 9-10)

Suunnitelmalla tarkoitetaan tekoälyn kykyä tehdä omia valintoja. Simulaatiossa jänikset ja ketut kykenivät tekemään reaaliajassa omia päätöksiään vallitsevan tilanteen mukaan. Eläinten valintojen seuraaminen systeemin ulkopuolelta ei ole ihmiselle kovinkaan helppoa, koska työssä ei käytetty minkäänlaisia animaatioi-

ta. Kulissin takana jokainen yksilö kuitenkin tekee omat päätöksensä. Simulaation sisällä yksilöt eivät ole tietoisia muiden eläinten aikeista, mikä on selkeä seikka, jota työssä voisi kehittää.

Suunnitelmallisuuden osalta pitää ottaa myös huomioon, että yksilöiden tekemien valintojen tulee olla mahdollisimman hyviä. Simulaation heikkoudeksi voidaan katsoa se, etteivät yksilöiden päätökset ole riippuvaisia muista yksilöistä, ainoana poikkeuksena jäniksen pako, jos se havaitsee ketun ympäristössään. Toisaalta voidaan katsoa, että työn pitäisi itsessään luonnonvalinnan kautta poistaa simulaatiosta yksilöt, jotka tekevät huonoja päätöksiä.

Suunnitelma-kriteerin voidaan katsoa täyttyneen työssä kohtalaisen hyvällä tavalla. Vaikka yksilöt jäivät välillä jumiin kahden päätöksen väliin, niiden päätökset olivat kokonaisuudessaan johdonmukaisia.

Toiminta-kriteerin täyttyminen edellyttää, että tekoälyn toiminta vaikuttaa luonnolliselta ja toteutuu vaaditussa ajassa. Toiminta-kriteeri täyttyi työssä hyvin. Itse reagointiaika oli jokaisella yksilöllä sama, mutta toimintojen laajuus oli suhteutettu jokaisen yksilön omiin ominaisuuksiin. Työssä ei ollut ideana matkia oikean maailman jäniksiä ja kettuja, vaan eläimet valikoituivat ainoastaan sen takia, että ne ovat kaikille tuttuja. Se, millä nimillä niitä kutsutaan, on toissijainen asia.

Reagointi-kriteerin voidaan katsoa olevan työn heikoin kohta, koska kuten aikaisemmin mainittiin, työssä eri yksilöt eivät reagoi monellakaan tavalla toistensa toimintaan. Ainoina poikkeuksina tähän on jänisten pakeneminen, jos ne havaitsevat ketun, ja molempien lajien lisääntymiseen liittyvä parinhaku. Tämä onkin varmasti suurin kehitysmahdollisuus itse työhön liittyen. Kummankin lajin reagointia ympäristöönsä ja muihin yksilöihin olisi pystytty kehittämään pidemmälle, jos aikaa olisi ollut enemmän.

5.1 Tulokset

Insinöörityössä oli alussa kaksi tavoitetta. Ensimmäinen tavoite oli kehittää toimiva tekoäly käytöspuita käyttäen, ja toinen tavoite oli pystyä mallintamaan luonnonvalinnan toimintaa peliympäristössä. Peliympäristöä ei aikataulun takia pystytty toteuttamaan, mutta mallintamisen pohjaksi valittu Unity-pelimoottori antaa hyvät lähtökohdat sen liittämiseen osaksi peliympäristöä.

Käytöspuun toteuttaminen onnistui halutulla tavalla. Tekoäly toimi hyvin ja noudatti kehitettyjä toimintamalleja.

Luonnonvalinnan onnistumisen testaamiseen valittiin kolme erilaista lähtökohtaa. Kaikissa esimerkeissä geenien mutaatiotodennäköisyys oli 50 %, jotta mutaatioita pystyttiin tuottamaan mahdollisimman nopeasti. Jokaisella mutaatiolla oli mahdollisuus muokata kutakin ominaisuuden arvoa väliltä -0.2 ja +0.2. Simulaatioiden alkuasetelmassa oli sattumanvaraisesti laskettu jokaiselle yksilölle omat geenit ja yksilöiden paikat arvotaan kartalta. Ensimmäiseen simulaatioon valittiin lähtökohdaksi tilanne, jossa jäniksiä oli 20 ja kettuja 40.

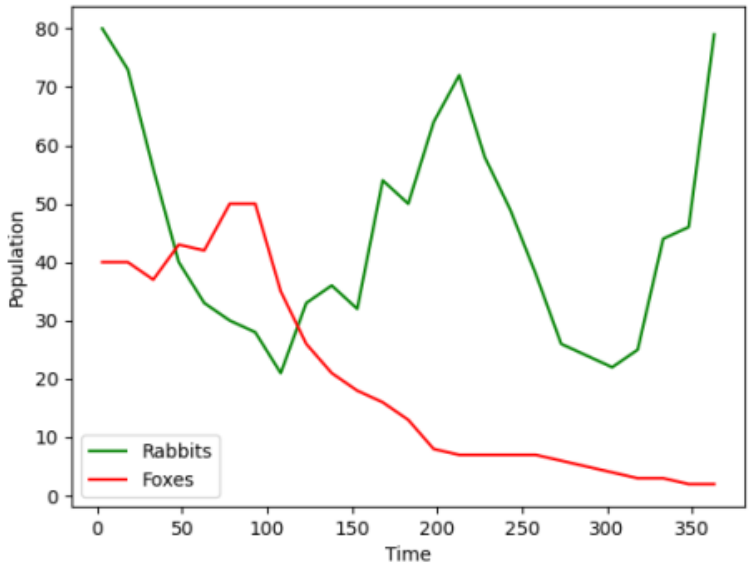
Tällaisessa tilanteessa odotuksena oli, että ympäristö on hyvin epäsuotuisa kahden lajin yhteiselolle, ja tulokset osoittivatkin, että tällaisessa tilanteessa saalistajien suuri määrä aiheutti jänisten sukupuuton todella nopeasti eivätkä lajit pysyneet elossa tarpeeksi kauan, jotta simulaatiosta olisi voitu esittää tuloksia, kun luonnonvalintaa ei ehtinyt syntyä.

Toisessa esimerkissä yksilöiden määräksi valittiin molemmille 30. Tämänkin esimerkin odotukset vastasivat ensimmäisen esimerkin tilannetta, ja huomattiin että ketut metsästivät jänikset sukupuuttoon, mutta hieman hitaammin kuin esimerkissä 1. Jäniksillä oli kuitenkin vielä ennen sukupuuttoa mahdollisuus lisääntyä, mutta yksilöt olivat niin hajallaan, etteivät ne löytäneet enää mahdollisuuksia lisääntyä.

Viimeisessä esimerkissä yksilöt jaettiin ketuille 20 yksilöä ja jäniksille 40 yksilöä. Tämä tilanne oli odotetusti kaikkein soveliain, joten tilannetta kokeiltiin vielä niin, että yksilöiden määrät nostettiin 40 kettuun ja 80 jänikseen. Tämä tilanne näytti testauksessa toimivan parhaiten, kuten oli oletettu.

Vaikka edellä mainittu tilanne toimi parhaiten, ei tasapainoisen ympäristön luominen onnistunut. Osaltaan on oletettavaa, ettei ainoastaan kahden lajin välinen ekosysteemi pysy kauan tasapainoisena. Simulaatioiden epäonnistumiseen ei pystytä rajaamaan yhtä ainoaa syytä. Usein simulaatioiden epäonnistuminen johtui huonosta tuurista. Simulaatioiden lopputulosten välillä oli huomattavan paljon vaihtelua, mikä on oletettavaa ottaen huomioon, että luonnonvalinnan mutaatiot ovat lopulta yksittäin erittäin sattumanvaraisia.

Kuvassa 12 esitetään kummankin lajin määrät yhden sellaisen simulaation aikana, jonka toiminta pysyi elossa jonkin aikaa. Satunnaisen alkuasetelman takia alussa jänisten määrässä tapahtuu merkittävä lasku, koska jotkin yksilöt voivat olla jo alussa vanhoja, nälkiintyneitä tai janoisia. Kummankin lajin määrän väheneminen simulaation alussa oli odotettavaa. Jänisten määrän väheneminen aiheutti lopulta myös kettujen määrän romahtamisen, ja lopulta jänikset pääsivät lisääntymään ilman kettujen puuttumista niiden toimintaan.



Kuva 9. Eläinten määrä simulaation aikana.

Simulaatioista huomasi usein samanlaisen kaavan: alussa ketut söivät ympäriltään jänikset, mikä aiheutti jänisten määrän romahtamisen, mutta lopulta jänikset löysivät paikan, jossa kettuja ei ollut, ja pystyivät näin ollen luomaan erillisiä populaatioita, joilla oli mahdollisuus lisääntyä. Kettujen sukupuutto oli usein sattumaa, mutta tiedostettiin myös, että kettujen kyky etsiä tai seurata jäniksiä ei ole toivotulla tasolla.

5.2 Kehitysmahdollisuudet

Työssä olisi kehitettävää jokaisessa ominaisuudessa. Luonnonvalinnan mallintaminen onnistui, mutta sen tarkkuus ja luotettavuus eivät olleet halutulla tasolla.

Suurimmat kehitysmahdollisuudet itse työn osalta liittyvät vahvasti tekoälyn reagointikyvyn parantamiseen. Lajien välinen toiminta parantuisi huomattavan paljon, jos lajeilla ja yksilöillä olisi enemmän tapoja olla tietoisia toistensa toiminnasta. Tällaisia tapoja voisivat olla esimerkiksi hajujälkien jättäminen tai yksilöiden mahdollisuus muokata toiminnallaan vallitsevaa ympäristöä. Monimutkaisen systeemin luominen toisaalta vaikeuttaa tulosten tarkastelua, jos mahdollisia muuttujia tilanteeseen on liian paljon.

Peliympäristöstä voidaan todeta, ettei yksilöiden liiallinen vastuu itsestään ole välttämättä haluttu toiminto. Pelien johdonmukaisuuden kannalta on olennaista, että esimerkiksi vihollisten ominaisuudet ovat jossain määrin säädettyjä sen suhteen, miten pelaaja etenee pelissä. Toisaalta kehitysmahdollisuutena voisi olla irrallisten ja suljettujen ympäristöjen luominen luonnonvalinnan pohjalta, jossa pelaajan toiminta vaikuttaisi esimerkiksi uudelleensyntyvien vihollisten ominaisuuksiin. Luonnonvalinnan pitäisi kuitenkin toimia suhteellisen nopeasti, mikä ilman ulkopuolista vaikutusta voi johtaa hyvinkin erilaisiin tuloksiin, jotka pahimmassa tapauksessa saattavat pilata pelaajan kokemuksen pelissä.

Pidemmälle kehitettynä vastaavanlaisella sovelluksella voisi kuitenkin olla paikka kouluissa demonstroimaan evoluution ja luonnonvalintaa nuorille. Tätä on jonkin verran yritetty, mutta mikään ohjelma ei ole saanut varmaa jalansijaa ai-

nakaan suomalaisissa kouluissa tähän mennessä. Tähän voi olla syynä laitteiston puute sekä se, että luonnonvalintaa tulisi pystyä havainnollistamaan mahdollisimman nopeasti, jotta jokainen oppilas pystyisi käyttämään ohjelmaa itsenäisesti.

6 Yhteenveto

Kokonaisuutena insinöörityö onnistui, vaikka sen hiominen jäikin lopulta vähäiseksi ja tekoälyn, sekä luonnonvalinnan, toimintaa olisi pystytty parantamaan, jos käytettävää aikaa olisi ollut enemmän. Luonnonvalinnan mekanismin havainnollistaminen pienessä mitassa on haastavaa, koska luonnossa sen toimintaan vaikuttavat monet eri tekijät ja lopputulokset ovat hyvin sattumanvaraisia.

Mallintamisen laajuus jäi lopulta hyvin pieneksi, koska erilaisten teknisten toteutuksien rakentaminen kesti odotettua kauemmin. Alun perin ideana oli havainnollistaa luonnonvalintaa useamman eri lajin välillä, mutta lopulta työstä jäi toteuttamatta muun muassa kettuja syövä huippupeto sekä esimerkiksi hirven kaltainen kasvissyöjä, jolla ei ole luonnollisia saalistajia. Yhtenä ideana oli myös esittää ekosysteemiin uusi laji simulaation aikana ja tarkkailla vieraslajien vaikutusta, mutta näistä jouduttiin ajanpuutteen vuoksi luopumaan ja päädyttiin tarkastelemaan luonnonvalinnan toimintaa pienemmässä mittakaavassa.

Työssä huomattiin, että luonnonvalinnan kautta muuttuvia lajeja on vaikea ennustaa, joten niiden käyttäminen peliympäristössä on haastavaa. Peliympäristössä niiden toimintaa jouduttaisiin rajoittamaan hyvän pelikokemuksen takaamiseksi, ellei pelin nimenomainen idea ole simuloida tämäntyyppistä toimintaa.

Työssä kävi ilmi, millaisia ominaisuuksia hyvällä tekoälyllä on ja minkälaisilla mekanismeilla luonnonvalinta toimii. Vaikka luonnonvalinnan osaamisesta ei välttämättä ole tulevaisuudessa hyötyä, sen ymmärtäminen on kuitenkin tärkeä osa tieteitä yleisesti. Peliteollisuudessa tekoäly on erittäin iso kokonaisuus, ja sen osaaminen varmasti auttamaan tulevaisuudessa.

Lähteet

A* Pathfinding Project. Verkkoaineisto. A* Pathfinding Project. < https://arongranberg.com/astar/>. Luettu 9.10.2023.

A Brief Guide to Genomics. Verkkoaineisto. National Human Genome Research Institute. <https://www.genome.gov/about-genomics/fact-sheets/A-Brief-Guide-to-Genomics>. Luettu 26.9.2023.

Adamatzky, Andrew & Komosinski, Maciej. 2005. Artificial Life Models in Software. New York: Springer.

Buckley, Gabe. 2021. Natural Selection. Verkkoaineisto. <https://biologydictionary.net/natural-selection/#other-types-of-natural-selection>. Luettu 4.11.2023.

Choi, Jung; Spencer, Chrissy; Kerr, Shana; Weigel, Emily & Montoya, Joe. 2023. Biological Principles. Verkkoaineisto. <https://bioprinciples.biosci.gatech.edu/module-1-evolution/neutral-mechanisms-of-evolution/>. Luettu 4.11.2023.

Cox, Brian. & Cohen, Andrew., 2016. Forces of nature. HarperCollins UK.

Crowley, Nate. 2020. Have You Played… SimLife: The Genetic Playground? Verkkoaineisto. <https://www.rockpapershotgun.com/have-you-played-simlife-the-genetic-playground>. Luettu 4.11.2023.

Crowley, Nate. 2019. Have You Played… SimEarth? Verkkoaineisto. <https://www.rockpapershotgun.com/have-you-played-simearth>. Luettu 4.11.2023.

Bourg, David & Seemann, Glenn. 2004. AI for Game Developers. United States: O'Reilly Media.

Darwin's Demons Science Trailer. 2017. Verkkoaineisto. Polymorphic Games. <https://www.youtube.com/watch?v=hPxaOBDeTC8>. Katsottu 16.10.2023.

Dawkins, Richard. 2009. The Greatest Show on Earth: The Evidence for Evolution. Simon and Schuster.

Eiben, A.E. & Smith, J.E. 2015. Introduction to Evolutionary Computing. Second Edition. Berlin, Heidelberg: Springer.

Genetics. Verkkoaineisto. National Institute of General Medical Sciences. <https://www.nigms.nih.gov/education/fact-sheets/Pages/genetics.aspx>. Luettu 26.9.2023.

Graph Types. Verkkoaineisto. A* Pathfinding project. <https://arongranberg.com/astar/docs/graphtypes.html>. Luettu 9.10.2023.

Heikkinen, Mikko. 2005. Evoluutio ja evoluutioteoria. Verkkoaineisto. <https://www.biomi.org/biologia/evoluutio/>. Luettu 27.9.2023.

Leith, Alex; Ratan, Rabindra & Wohn, Donghee. 2016. The (De-)evolution of Evolution Games: A Content Analysis of the Representation of Evolution Through Natural Selection in Digital Games. Journal of Science Education and Technology. Vol. 25, no. 4, s. 655-664.

Lane, Rick. 2018. 10 Years On: Spore. Verkkoaineisto. Bit-tech.net. <https://bit-tech.net/features/gaming/pc/10-years-on-spore/1/>. Luettu 4.11.2023.

Livingstone, Daniel. 2006. Turing's test and believable AI in games. Computers in Entertainment. Vol. 4, Issue 1, Article 3D.

Ogren, Petter. 2012. Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees. AIAA Guidance, Navigation, and Control Conference s. 4458

Osterloff, Emily. What is natural selection? Verkkoaineisto. Natural History Museum. <https://www.nhm.ac.uk/discover/what-is-natural-selection.html>. Luettu 25.9.2023.

Poliakov, Mykhailo. 2020. Evolution of digital organisms in truly two-dimensional memory space. Bachelor Thesis. Ukrainian Catholic University.

Savage, V. M.; Gillooly, J. F.; Woodruff, W. H.; West, G. B.; Allen, A. P.; Enquist, B. J. & Brown, J. H. 2004. The predominance of quarter-power scaling in biology. Functional Ecology, Vol 18, Issue 2, s. 257-282.

Simpson, Chris. 2014. Behavior trees for AI: How they work. Verkkoaineisto. <https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work>. 18.7.2014. Luettu 3.10.2023.

Zeng, W. & Church, R. L. 2009. Finding shortest path on real road networks: the case for A*. International Journal of Geographical Information Science. Vol. 23, Issue 4, s. 531-543.

Unity's Interface. Verkkoaineisto. Unity Documentation. <https://docs.unity3d.com/Manual/UsingTheEditor.html>. Luettu 23.10.2023.