Larri Jäntti

Lintuparven tekoälyn peliohjelmointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

11.4.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Larri Jäntti Lintuparven tekoälyn peliohjelmointi 33 sivua + 2 liitettä 11.4.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	pelisovellukset
Ohjaajat	tutkintovastaava Antti Laiho teknologiajohtaja Teemu Saukonoja

Insinöörityön tavoitteena oli luoda linnuille pelimaailmaan simuloitu parveilutekoäly ja laajentaa sitä yksilökohtaisilla käyttäytymisfunktioilla. Tekoälyn ympärille oli tarkoitus rakentaa luonnollinen puistoympäristö, jotta tekoälyn tarkkailu olisi mielenkiintoista myös niille, joita ei projektin tekninen toteutus kiinnosta. Tavoitteisiin kuului tehdä ympäristöstä interaktiivinen ja sellainen, jossa pelaaja voi liikkua vapaasti ja myös muuttaa parveilumallin käyttäytymiseen vaikuttavia kertoimia ja arvoja reaaliajassa nähdäkseen muutosten vaikutukset lintujen parveilussa. Työn tulokset annettiin lopuksi asiakasyrityksen käyttöön.

Työ päätettiin toteuttaa käyttäen Unity-pelimoottoria, ja parveilun alustaksi valittiin klassinen boid-malli. Unity oli työn tekijälle entuudestaan tuttu, joten työn suoritus ei jäänyt missään välissä pysähdyksiin teknisten seikkojen tai ymmärtämättömyyden takia. Boid-malli on 1980-luvulla kehitetty luonnollinen parveilumalli, jossa jokainen parven yksilö vaikuttaa välittömän lähiympäristönsä liikkumiseen ja toimintaan. Malli on yhä perustaltaan tarkin parveilusimulaation esitystapa, vaikkakin suorituskyvyllisesti erittäin raskas.

Työn kulku oli tasapainoista ja tavoitevetoista. Työn on kokonaisuudessaan suunnitellut ja toteuttanut yksi henkilö, joten työnjako ja aikataulutus eivät nousseet ongelmaksi. Työn toteutus sujui odotetusti, ja kaikki suunnitellut ominaisuudet lisättiin onnistuneesti. Viimeisimmät muutokset tehtiin yritysasiakkaan edustajan ja ulkopuolisen testaajan toiveiden perusteella.

Parveilumalli toimi odotetusti, ja se on mahdollista toteuttaa pienelläkin vaivalla, riippuen kohdeprojektin vaatimuksista. Parveilutekoälyn lisäksi muita lintujen käytökseen vaikuttavia ominaisuuksia insinöörityön projektissa toteutui muun muassa törmäysten havainnointi, ruokailun tarve, laskeutuminen ja lentoonlähtö sekä pelaajan läheisyyteen reagointi. Insinöörityössä luotu parveilutekoälyn lähdekoodi on sellaisenaan käytettävissä myös muissa projekteissa.

Avainsanat	parveilu, tekoäly, peli, boid, lintu, lintuparvi

Author Title Number of Pages Date	Larri Jäntti Programming AI for a bird flock 33 pages + 2 appendices 11 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Game Applications
Instructors	Antti Laiho, Senior Lecturer Teemu Saukonoja, CTO

The goal of the thesis was to create a flocking AI (*Artificial Intelligence*) for birds and to expand it with individual behavioral functions. It was planned to build a natural park environment around the AI, so that observing the bird and flock behavior would be interesting also for those who are not keen on the technical aspect of the project. Aim was to make the environment interactive, so that the player could move freely, and they could also change the variables and values of the AI in order to see the effects of their modifications in real time. In the end, the results of this thesis and project were given for the client company to freely utilize.

The project was decided to implement using Unity game engine and classic boid model was chosen for the basis of the flocking AI. Unity was already well-known for the maker of this project, so the execution was never halted by technical uncertainty. Boid model is a natural flocking model developed in 1980's, in which every individual of the flock affects the movement and actions of the flock in their near vicinity. The basis of the model is still considered the most accurate representation of flocking simulation, although it is very demanding performance-wise.

Execution of the project was balanced and goal-driven. The whole project was planned and carried out by a single person, so distribution of tasks and timing the project were not an issue. Implementation of the project went as expected and every planned feature was added successfully. The last changes were made regarding the wishes and comments of the client company representative and a third-party tester.

The flocking model worked as intended and it is possible to produce even with a minimalistic approach, taking in consideration the aim of the target project. In addition to the flocking, other aspects that affect the actions of the birds are for example collision detection, feeding, landing and takeoff and reacting to player presence. The source code created and used in the thesis project is ready to be used as-is in other projects as well.

Keywords	flocking, artificial intelligence, ai, game, boid, bird

Sisällys

Lyhenteet

AI	Artificial Intelligence. Tekoäly.
Boid	Bird-oid object, lintumainen objekti. Klassisen parveilutekoälymallin nimi.
Flock	Pieni tai keskikokoinen parvi.
FPS	Frames Per Second, sovelluksen päivitystaajuus piirrettyinä ruutuina sekunnissa.
Swarm	Suuri parvi.

1 Johdanto

Insinöörityön tarkoituksena on tekoälyn luominen lintuparven luonnolliseen liikkumiseen ja käytökseen pohjautuen. Tavoitteena on luoda parveilukäytöksestä malli, jota voitaisiin myöhemmin hyödyntää esimerkiksi videopelin ympäristöelementtinä tai jopa keskeisenä pelimekaniikkana. Työn on tilannut helsinkiläinen mobiilialan ohjelmistostartup Hisome Oy. Insinöörityön aihe on valittu henkilökohtaiseen mielenkiintoon perustuen, ja Hisome Oy tarjoutui asiakasyritykseksi tarkoituksena projektin tuloksien mahdollinen jatkokäyttö myöhemmissä projekteissa.

Työ tehtiin Unity-pelimoottoria käyttäen, ja ohjelmointi toteutettiin C#-kielellä. Projekti rakennettiin toimivaksi sovellukseksi Windows-, OSX- ja Linux-pohjaisille tietokoneille. Sovelluksessa käyttäjä hallitsee pelikameraa, joka on sijoitettu puistoon, jonne on luotu parveilualgoritmia käyttäviä lintuja. Parven käyttäytymistä ja kokoa on mahdollista säätää ajonaikaisesti pelin sisäisestä pysäytysvalikosta.

Insinöörityön avulla on tarkoitus antaa käyttäjälle tietoa parveilualgoritmin toiminnasta antamalla mahdollisuus muokata parveiluun ja liikehdintään vaikuttavia arvoja suoraan demon sisällä. Parveilutekoälyn demonstroinnin lisäksi sovelluksen tavoitteena on luoda käyttäjälle rauhallinen ja mahdollisimman kokonaisvaltainen puistokokemus, josta voi nauttia tietämättä tai välittämättä taustalla olevista tekoälymekaniikoista.

Tekoälyllä tarkoitetaan tekniikan ja ohjelmoinnin alalla yksilöä tai objektia, joka pystyy havainnoimaan ympäristöään ja tekemään havaintojensa pohjalta loogisia päätöksiä [1]. Tässä projektissa tekoälyä sovellettiin jokaisen lintuyksilön reitinetsintään ja lintujen vuorovaikutuksen luomiseen ja yritettiin luoda lintujen parveilukäytöksestä mahdollisimman luonnollisen ja autenttisen näköistä. Tekoälyä on laajennettu muilla ärsykkeillä, kuten esteiden väistöllä ja ruoanhankinnalla. Parveilulla tekoälyn yhteydessä tarkoitetaan ryhmän liikkumiseen vaikuttavia dynamiikkoja, ryhmän yksilöiden välistä vuorovaikutusta ja tavoitetta ohjata yksilön liikettä perustuen muiden yksilöiden sijaintiin ja nopeuteen [2].

Raportin sisältöön kuuluu työnkulun raportoinnin lisäksi tietoa tekoälyn historiasta peleissä sekä tekoälyn ja parveilualgoritmien käytöstä yleisesti.

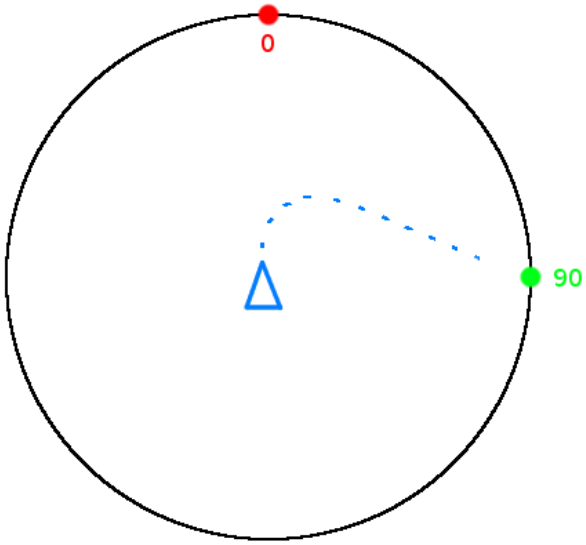
2 Lintujen käyttäytyminen

2.1 Parveilu ja liikkuminen

Insinöörityössä oli tärkeässä osassa löytää oikeanlainen käytösmalli pohjautuen oikeiden lintujen luonnolliseen parveilumekaniikkaan. Tätä tutkiessa täytyi ottaa huomioon muun muassa potentiaalinen lintulaji ja parven koko. Lopulta kuitenkin sovelluksen luomisen rajoittavaksi tekijäksi muodostuivat projektin tavoitteet ja aikataulu.

Projektia aloittaessa haluttiin, että demossa esiintyvä laji olisi tunnistettavissa ja oikeasti parveileva. Farleyn ym. tutkimuksessa lintulajien välisestä parveilusta [3, s. 5–6] on listattu eri lajien taipumus parveilukäyttäytymiseen käyttäen empiiristä eli havainnointipohjaista tutkimusta. Käyttäen tutkimuksen lajilistausta ja vertaamalla sitä Unityn Asset Storesta eli kauppapaikasta löytyviin ilmaisiin lintumalleihin päädyttiin käyttämään projektissa lajina amerikkalaista punarintaa (*Turdus migratorius,* engl. american robin [4]). Tutkimuksen mukaan punarinnalla on havaittu vahvaa parveilukäyttäytymistä, ja YouTube-videopalvelusta löytyykin paljon videoita tämän lajin parveilusta [5].

Lintuobjektin ohjelmoinnissa on otettava huomioon muutamia seikkoja liittyen erityisesti lintujen liikkumistapaan. Linnut lentävät ilmassa pääasiassa vain eteenpäin (nokan osoittamaan suuntaan). Ohjelmoitaessa linnun perusliikkumista on lintumallin rotaatiota siis aina muutettava, kun halutaan liikuttaa mallia. Esimerkiksi jos lintu on liikkumassa suuntaan 0 ja uusi reittipiste ilmestyy suuntaan 90, lintuun ei saa kohdistaa sivuttaisia voimia vaan lintu pitää kääntää ja kohdistaa uudelleen linnun yhä lentäessä eteenpäin koko prosessin ajan (kuva 1). Myös lintujen rotaation tulee pääosin pysyä niin, että linnun vatsapuoli on maata kohden ja selkäpuoli ylöspäin.



Kuva 1. Linnun liikkumisen ja kääntösäteen havainnollistus.

Tähän on tietenkin olemassa poikkeuksia: linnut pystyvät esimerkiksi kääntymään erittäin pienellä kääntösäteellä kesken lennon tai jopa lentämään paikallaan käyttäen ilman virtauksia ja paine-eroja [6]. Nämä kuitenkin ovat lajikohtaisia ominaisuuksia, eivätkä kovin tärkeitä työn tavoitteita ajatellen. Sovelluksessa linnut pystyvät myös laskeutumaan ja lähtemään lentoon itsenäisesti ja pelaajan läsnäolosta riippuen.

2.2 Ärsykkeet ja reagoiminen

Luonnossa lintujen parvilentoa ohjaa usea ympäristömuuttuja parven sisäisten sääntöjen lisäksi. Näistä huomattavimpia ovat esimerkiksi tuulen vaikutus, petoeläimet ja muut vastaavat pelotteet, ravinnonhaku sekä yleinen muuttoliike [7]. Työssä on otettu huomioon nämä kaikki, lukuun ottamatta muuttoliikettä, sillä se ei vaikuta paikallisparveilun muotoon.

Parvena linnut pystyvät väistämään esimerkiksi saalistavaa petolintua: kun riittävä osa parvesta havaitsee uhan, linnut pystyvät ohjaamaan koko parvea väistämään tiettyä uhkasuuntaa. Samanlaista, mutta jopa tehokkaampaa dynaamista parveilua esiintyy kalaparvissa. Projektityössä on otettu huomioon mahdollisimman paljon ympäristövaikutuksia, jotta parveiluun tulee tarpeeksi vaihtelevuutta sekä muodon- ja suunnanmuutoksia. Näin parvi ei vaikuta niin keinotekoiselta vaan saa aikaan elävän vaikutelman aidosta lintuparvesta.

3 Tekoäly ja parveilualgoritmit peleissä

3.1 Tekoälyn historia peleissä

Tekoälyn kehitys alkoi nopeasti ensimmäisten tietokoneiden tulon jälkeen, mutta pelisovelluksiin toimiva tekoäly alkoi muotoutua vasta 1970-luvulla. Ensimmäiset tekoälyt peleissä toimivat esiohjelmoituina käytöksinä ja liikkeinä, eivätkä kohteet itse tehneet päätöksiä. Näin oli esimerkiksi Taiton vuonna 1978 julkaisemassa pelissä Space Invaders. Pelissä ammutaan avaruusolioita, jotka valuvat ruudun yläreunasta alaspäin seuraamalla esiohjelmoituja reittejä. Space Invadersia seurasi liuta samankaltaiseen käytösmalliin perustuvia videopelejä. Toisenlainen esimerkki ensimmäisistä pelitekoälyistä on Atarin vuonna 1972 julkaisema peli Pong. Se on ylhäältäpäin kuvattu kaksiulotteinen pöytätennispeli, jossa oli mahdollista valita tietokonevastustaja siirtämään toista mailaa. Vaikka tekoälyn ohjaama maila liikkui pelaajan oman mailan ja pallon mukaan näytöllä, se nähtiin silti hyvin ”ihmisenkaltaisena” käytöksenä, ja tämä onkin yksi aikaisimmista peliteollisuuden kulmakivistä. [8.]

Esiohjelmoitua reittipohjaista tekoälyä vei eteenpäin Namcon vuonna 1980 julkaisema Pac-Man, jossa pelaajan on tarkoitus kerätä sokkelosta kolikoita varoen samalla sokkeloa kiertäviä haamuja. Jokaisella haamulla on oma ”persoonallisuus” ja käyttäytymismalli, jonka mukaan haamu joko jahtaa tai pakenee pelaajaa. Käytös määrittyy pitkälti sen mukaan, mikä on haamun etäisyys ja suunta pelaajaan nähden. [8.]

Tekoälyn kehitys peleissä lähti kunnolla liikkeelle vasta 1990-luvulla, kun RTS- (Real-Time Strategy) eli reaaliaikaiset strategiapelit tulivat markkinoille. Sellaisissa peleissä kuin Command & Conquer (Westwood Studios, 1995) ja WarCraft: Orcs & Humans (Blizzard Entertainment, 1995) oli molemmissa pelimuoto, jossa pelaaja pystyi pelaamaan tietokonevastustajaa vastaan. RTS-peleissä on pääasiassa tarkoituksena rakentaa ja kehittää omaa kaupunkia, luoda armeija ja vallata vihollisten alueet. Tekoäly toimi usein strategisen evaluoinnin avulla ja käytti myös polunetsintää suoriutuakseen tehtävistä ihmispelaajan tavoin. Tätä tekoälytyyppiä parantelivat myöhemmin muun muassa WarCraft 3 (2002), StarCraft (1998, Blizzard Entertainment) sekä monet muut eri studioiden RTS-pelit lähinnä edellä mainittujen inspiroimana. [8; 9; 10.]

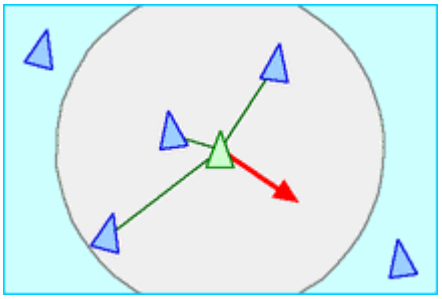
Nykyään erilaisia tekoälysovellutuksia käytetään peleissä yleisesti esimerkiksi NPC- (Non-Player Character) eli tietokoneohjattujen hahmojen toimintaan, ympäristövaikutelman luomiseen tai vaikka koko pelin juonelliseen etenemiseen, kuten pelissä Skyrim (Bethesda Game Studios, 2011). Skyrimissa toimiva tekoäly on nimeltään *Radiant A.I.,* ja se on Bethesdan itse pitkään kehittämä dynaaminen tekoäly. Se säätelee pelin kulkua pelaajan tason ja toimien mukaisesti ja muokkaa pelaajan pelikokemusta sitä mukaa, kuin peli etenee. Tekoäly kykenee muun muassa lisäämään pelaajalle omantasoisia tehtäviä, muuttamaan NPC-hahmojen käyttäytymistä ja myös kontrolloimaan pelin juonellista kulkua. Pelien tekoälyominaisuudet ovat kasvaneet nopeasti siitäkin syystä, että tietokoneet ja pelikonsolit ovat muuttuneet paljon tehokkaammiksi, eikä monimutkaisenkaan tekoälyn suorittaminen pelin sisällä tule esteeksi suorituskyvylle. [11.]

3.2 Parveilualgoritmit

Klassinen Boid-malli

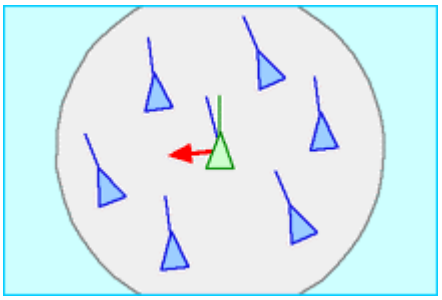
Ensimmäisen, niin kutsutun klassisen parveilualgoritmin mallin kehitti Craig Reynolds vuonna 1986. Se sisälsi parveiluun vaikuttavat kolme sääntöä ja niiden toiminnan käytännössä:

- Separation (erotus): jokainen parven yksilö muuttaa suuntaansa poispäin viereisistä yksilöistä. Tämä muun muassa estää parven sisäisiä törmäyksiä (kuva 2).



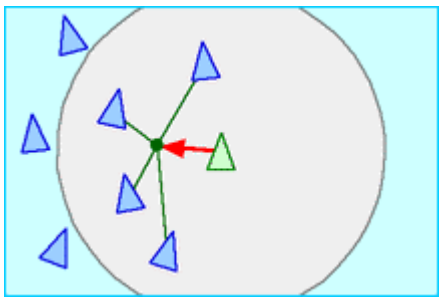
Kuva 2. Reynoldsin boid-mallin ensimmäinen sääntö, erotus [13].

- Alignment (suuntaus): jokainen parven yksilö suuntautuu parven keskimääräistä kulkusuuntaa kohden (kuva 3).



Kuva 3. Reynoldsin boid-mallin toinen sääntö, suuntaus [13].

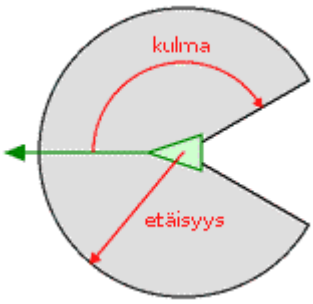
- Cohesion (yhtenäisyys): jokainen parven yksilö hakeutuu kohti parven massakeskipistettä eli parven yksilöiden keskisijaintia (kuva 4). [12.]



Kuva 4. Reynoldsin boid-mallin kolmas sääntö, yhtenäisyys [13].

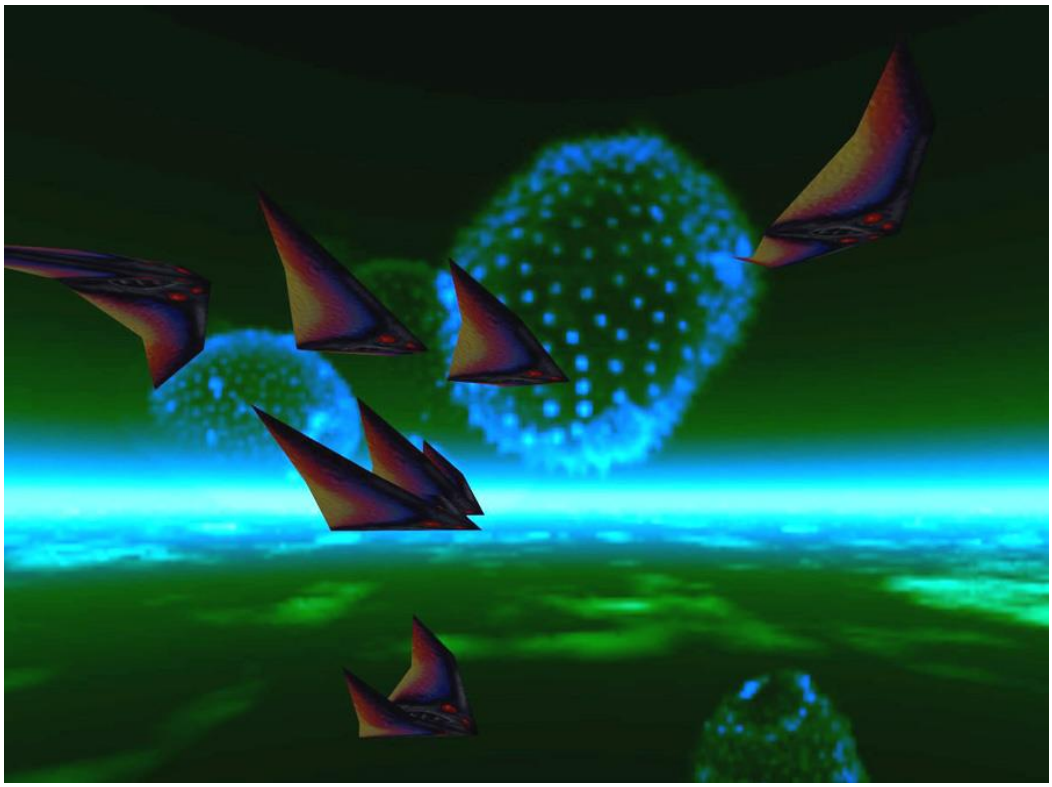
Boid-malliksi (bird-oid object, lintumainen objekti) kutsuttua mallia pidetään realistisimpana ja parhaana parveilumallina. Siihen on lisätty erilaisia elementtejä liittyen esimerkiksi esteidenväistöön ja yksilön omiin tarpeisiin, kuten nälkään, mutta perusmalli itse on säilynyt kyseenalaistamattomana perustana alalla jo yli kolmekymmentä vuotta. Mallin pohjalta on luotu muun muassa animaatiosimulaatioita elokuviin, esimerkiksi Walt Disneyn elokuvaan Leijonankuningas (1994).

Mallissa jokainen yksilö (eli boid) laskee omaa liikerataansa lähiseudun muiden yksilöiden tietoja käyttäen kolmen edellä mainitun säännön avulla. Lähiseutu määritellään säteenä yksilön ympärillä, ja säteen suuruuden muuttaminen vaikuttaa olennaisesti parveiludynamiikkaan. Lähiseutuun ei lasketa klassisen mallin mukaan aluetta yksilön takana, sillä luonnossa parveileva yksilö harvoin pystyy havainnoimaan takanaan tapahtuvaa liikettä (kuva 5). [12.]



Kuva 5. Yksilön lähiseutu klassisen boid-mallin mukaan [13].

Yksi ensimmäisistä peleistä, joka käytti boid-parveilumallia, oli Half-Life (Valve, Sierra Studios, 1998). Pelin lopussa pelaaja näki taustalla parveilevia, lentäviä olentoja, joihin oli viitattu pelin lähdetiedostoissa nimellä ”Boid” (kuva 6).



Kuva 6. Xen-boideja Half-Lifessa [14].

Half-Lifen jälkeen pelien tullessa monimutkaisemmiksi ja laajemmiksi ja tietokoneiden nopeuden noustessa alkoivat parveilualgoritmitkin yleistyä. Nykyään niiden avulla ohjataan ja simuloidaan yleisellä tasolla erilaisten joukkojen, kuten ihmisten tai eläinten liikehdintää, mutta ei välttämättä tarkalla yksilötasolla, vaan lähinnä suuntaa-antavasti. Esimerkiksi pelissä Assassin’s Creed (Ubisoft, 2007) on yritetty luoda elävämpää

toritunnelmaa liikuttelemalla ihmismassoja samansuuntaisesti, mutta väistäen ja reagoiden pelaajaan ja muihin hahmoihin. [10; 15.]

Boid-mallin suorituskyky

Boid-malli on erittäin tarkka algoritmi parveilukäyttäytymisen kuvaukseen, mutta se vaatii myös suorituskyvyllisesti paljon resursseja. Koska jokaisen parven yksilön täytyy laskea etäisyys jokaiseen muuhun parven yksilöön määritelläkseen etäisyyden ja oman lähipiirinsä, on tekoälyn suoritusaika vähintään neliöön verrannollinen yksilöiden määrään. Reynoldsin mukaan parvella on siis asymptoottinen monimutkaisuus $O(n^2)$, jossa n on parven koko. Algoritmin käyttöä sellaisenaan vältetäänkin usein mobiiliympäristössä. [12; 13; 16.]

Alkuperäiseen boid-malliin on julkaisun jälkeen kehitelty parannuksia ja optimointitapoja, joista tehokkaimpia ovat

- tekoälyn päivitystaajuuden laskeminen
- tiedostorakenne, joka pitää yksilön lähipiirin helpommin käsiteltävässä muodossa tarvitsematta tehdä läheisyyshakua aina päivityksen yhteydessä. [17.]

Käyttämällä esimerkiksi kolmiulotteista tiedostorakennetta, jossa jokaista lintua kuvaa taulukossa kolme arvoa: paikka, nopeus ja tyyppi, on onnistuttu simuloimaan jopa yli 8000 yksilön parvea mobiililaitteella ylläpitäen noin 60 FPS:n (*Frames Per Second*) päivitysnopeus. Samalla laitteella testattu klassinen boid-malli ilman tiedostorakenteen muuttamista toimi vain noin 300 yksilön parveen asti samalla nopeudella. Simulaatiossa käytetty mobiililaite on julkaistu vuonna 2011, joten voidaan olettaa, että nykyisillä mobiililaitteilla mahdollinen simuloitava parvikoko samalla algoritmilla olisi vielä suurempi. Vertailun vuoksi mainittakoon, että testissä käytettiin ARM Cortex A9 -kaksoisydinprosessoria, jonka korkein kellotaajuus on 1 GHz, kun esimerkiksi Samsungin juuri julkaisemassa Galaxy S9 -laitteen prosessorissa Exynos 9810:ssa on neljä ydintä, joiden suoritusnopeus on 2,9 GHz, ja lisäksi neljä ydintä, joiden nopeus on 1,9 GHz. Tämän lisäksi nopeuteen vaikuttavat tietenkin testausympäristön arkkitehtuuri, käytetty ohjelmointirajapinta sekä esimerkiksi väli- ja keskusmuistin määrä. Luonnollisesti simulaation tehokkuus kasvaa siirryttäessä PC-ympäristöön. [16; 18.]

Uudistettu malli suurille parville

Huolimatta parannuksista klassiseen malliin sen tuoma suorituskykyrasite oli liikaa suurimpiin parveilumallia tarvitseviin projekteihin, ja parven ohjaaminen ja kohdistaminen saattoi olla työlästä, riippuen implementaatiosta. Tällaista käytöstä mallintaviin sovellutuksiin luotiin uusi malli, jota kuvaillaan yleisesti englanninkielisellä sanalla *swarming* (suom. (hyönteisten) parveilu). Yleisesti ottaen tietotekniikassa käytettyä parveilua kuvataan englanniksi termillä *flocking*, joka tarkoittaa suomeksi samaa kuin *swarming*. Koska *flock*-termillä tarkoitetaan siivekkäiden eläimien, pääasiassa lintujen parvea ja *swarm*-termillä tarkoitetaan hyönteisistä koostuvaa parvea, voidaan olettaa, että swarm-parvi on yksilökooltaan suurempi.

Swarm-mallissa yksilöt jaetaan kahteen ryhmään niiden sijainnin mukaan. Jokainen yksilö sijaitsee joko ulommalla alueella tai sisemmällä alueella, riippuen niiden sijainnista ja etäisyydestä kohteesta katsottuna. Swarm-mallia käytetään yleensä monilukuisten, nopeiden ja pienten yksilöiden navigointiin kohdetta kohti. Esimerkiksi pelissä Hunt: Showdown (Crytek, 2018) eräs vihollistyyppi voi lähettää parven myrkyllisiä hyönteisiä pelaajan kimppuun. Parvi käyttäytyy kuten swarm-mallissa ottaen lähes suoraviivaisen reitin vihollisesta pelaajaan, lopulta ympäröiden tämän.

Jos yksilö on ulommalla alueella, sen suuntaa ohjataan hieman lähemmäksi sisempää aluetta käyttäen pieniä nopeus- ja suuntimavariaatioita. Kauempana olevien yksilöiden törmäyksistä ei tarvitse huolehtia, sillä niiden etäisyys kohteesta ja samansuuntainen liike yleensä takaavat luonnollisen näköisen etenemisen ilman päällekkäisyyttä. Kun yksilö päätyy sisemmälle alueelle, sen liikehdintää ohjaavat kaksi algoritmia: nopeuden riippuvuus suunnasta ja suunnan riippuvuus nopeudesta. Käyttäen näitä kahta algoritmia yksilö saa sisemmän alueen keskiötä kiertävän liikeradan ja lopulta päätyy parveilemaan kohdepisteen ympärille. Swarm-parveiluun lisätään yleensä yksinkertainen törmäyksenestoalgoritmi ympäristössä navigointia varten.

Vaikka swarm-mallilla pystytäänkin teoreettisesti kontrolloimaan tuhansia yksilöitä kerrallaan, on silläkin selkeitä puutteita verrattuna klassiseen malliin: yksilöiden liike ei ole yhtä luonnollista ja törmäyksien havaitseminen on alkeellista. Peleissä on myös käytetty partikkeliefektejä luomaan yksinkertaisia ja suurikokoisia parvia, joiden ei välttämättä tarvitse kommunikoida ja navigoida keskenään parvena, vaan ne liikkuvat kaavamaisesti. Niiden luonti on helppoa, ja ne ajavat asian esimerkiksi taustaefektinä.

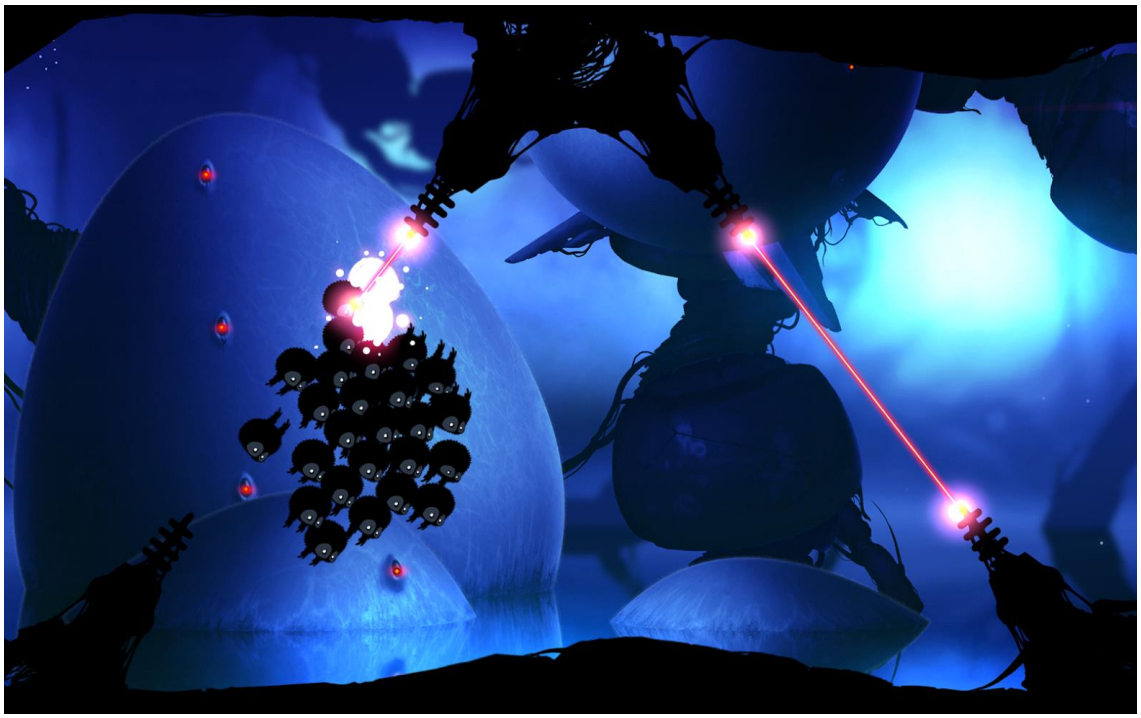
Insinöörityöprojektissa on käytetty lintujen tekoälynä klassista boid-mallia ja törmäysten havainnointia. Partikkeliefektejä on hyödynnetty luomaan demopuiston linnuille lentäviä hyönteisiä ravinnoksi.

Kuva 7. Parvi swarm-hyönteisiä pelistä Hunt: Showdown (Crytek, 2018) [19].

3.3 Parveilun käyttö pääasiallisena pelimekaniikkana

Vaikka parveilua on käytetty paljon osana pelin ympäristöä tai tietyn hahmon mekaniikkaa, pääasiallisena pelimekaniikkana sitä ei ole hyödynnetty suuren luokan peleissä. Esimerkiksi Nintendo GameCubelle vuonna 2001 julkaistu Pikmin (Nintendo) hyödyntää boid-mallin parveilusääntöjä pelihahmojen liikuttamiseen pelaajan käskyjen mukaisesti, mutta se jää silti pieneen osaan itse pelin kannalta. Mobiilipelien nousun myötä parveilupelit ovat alkaneet yleistyä, sillä mobiilialustalle pelien kehitys voi olla erittäin minimalistista ja perustua yhteen hyvään ideaan. Mobiilipelien inspiroimat tietokonepelit ovat myös lisääntyneet Valven Steam-sovellusportaalin tarjoaman kauppapaikan siivittämänä. Pelin sisältöä ei välttämättä tarvitse olla paljoakaan, kunhan peli on miellyttävä pelata ja toimii omassa kategoriassaan. Usein nämä pelit ovat ilmaisia ladata ja ne perustuvat värikkäisiin grafiikoihin, joiden avulla houkutellaan pelaajia lataamaan ja kokeilemaan peliä.

Esimerkkejä parveilua päätoimisena pelimekaniikkanaan käyttävistä peleistä ovat muun muassa mobiilialustalle luotu Swarming (Acalamity, 2016), Windows PC:lle tehty Swarm Arena (Dedication Games, 2010) sekä paljon mainetta niittänyt suomalainen mobiilipeli Badland ja sen jatko-osa Badland 2 (Frogmind Games, 2013 ja 2015). Badlandissa pelaaja kontrolloi pientä mustaa olentoa napauttamalla ruutua ja saaden sen lentämään hieman ylöspäin joka napautuksella. Näin hahmo on tarkoitus saada sivuttaissuunnassa liikkuvan kentän alusta loppuun. Pelin parvimekaniikka tulee esille, kun pelaaja löytää kentältä lisää pieniä olentoja, jotka liittyvät pelaajan ohjaamaan parveen. Vaikka parvi ei selkeästikään käytä kaikkia boid-mallin periaatteita, eikä ohjaile itseään, se pysyy silti koossa ja muuttaa tarvittaessa muotoaan tason sekä pelaajan syötteen mukaisesti (kuva 8). Alkuperäistä Badland-peliä on ladattu pelkästään Android-laitteille jo lähes 1,5 miljoonaa kertaa maaliskuuhun 2018 mennessä [20].



Kuva 8. Pelikuvaa ja parvidynamiikkaa pelistä Badland [20].

4 Tekoälyn ohjelmointi insinöörityössä

4.1 Lintujen liikkuminen

Lintujen ohjelmointi oli luonnollista aloittaa yksittäisen linnun liikkumiseen vaikuttavista funktioista ja komponenteista. Ensimmäisenä osana lintuobjektiin kuuluu Rigidbody-komponentti, joka hoitaa fysiikkalaskennan ja toteuttaa voimien muutokset, joiden avulla lintuja liikutellaan. Rigidbodyn lisäksi objektiin tulee lisätä Collider-komponentti, joka puolestaan havainnoi törmäyksiä ja kerää dataa törmäyspositioista ja kohdatuista objekteista. Alkuun objektiin voi lisätä vaikka kuutio-objektin testausta varten, mutta koska lintulajien vertailu oli tehty jo, lisäsin objektiin suoraan punarinnan 3D-mallin ja animaattorin.

Lintujen liikkumista ja hallinnointia varten tehtiin C#-ohjelmaskripti *BirdMovement*. Skriptistä löytyvät funktiot linnun kohteeseen lentämiselle ja passiivisille toiminnoille, kuten ilmassa pysymiselle ja linnun rotaatiosta ja nopeudesta huolehtimiselle. Skripti sisältää myös lintukohtaisia muuttujia, kuten onko lintu navigoimassa vai parveilemassa, mikä on linnun ensisijainen lentokorkeus sekä muita linnun liikkumiseen vaikuttavia arvoja. Esimerkkikoodissa 1 esimerkiksi nostetta luova funktio, joka laskee nostevoiman demossa käytetyn painovoimavakion perusteella. Nosteeseen lisätään satunnainen arvo, jotta lintu ei leijuisi paikallaan samassa paikassa. Näitä ennen lasketaan linnun korkeusero, ja jos lintu lentää tavoitekorkeuden lähellä, ei luoda lisää nostetta, vaan linnun lento saa kelluvan yleisvaikutelman. Lopuksi tulokseksi saatu voima lähetetään linnun Rigidbody-komponentin prosessoitavaksi (esimerkkikoodi 1). Funktiota kutsutaan jokaisen fysiikkapäivityksen yhteydessä.

```
//Generate random lift force near the amount of reverse gravity to get a
"floaty" feel to the flight
public void GenerateLift()
{
    float distanceFromPref = preferredHeight - transform.position.y;
    if (Mathf.Abs(distanceFromPref) < heightBounceModifier)
        return;
    Vector3 lift = -gravity;
    lift.y += Random.Range(-liftOffset, liftOffset);
    lift *= distanceFromPref * verticalSway;
    rb.AddForce(lift, ForceMode.Force);
}
```

Esimerkkikoodi 1. Linnulle vaihtelevan nosteen luova funktio *GenerateLift().*

Liikkumisarvojen ja oikeiden voimaskaalojen etsiminen aloitettiin ohjelmoimalla linnuille yksinkertainen funktiojoukko, jonka tarkoituksena on saada lintu lentämään kohdekoordinaatteihin. Kun oli löydetty linnuille mieluiset lento- ja kääntymisnopeudet, alkoi parveilutekoälyn työstäminen.

4.2 Tekoälyalgoritmin luonti

Projektissa päätettiin tehdä tekoälystä klassinen boid-pohjainen parveilualgoritmi, sillä se on yhä laadukkain kohtalaisten ja pienten yksilömäärien parveilua mallintava tekoäly.

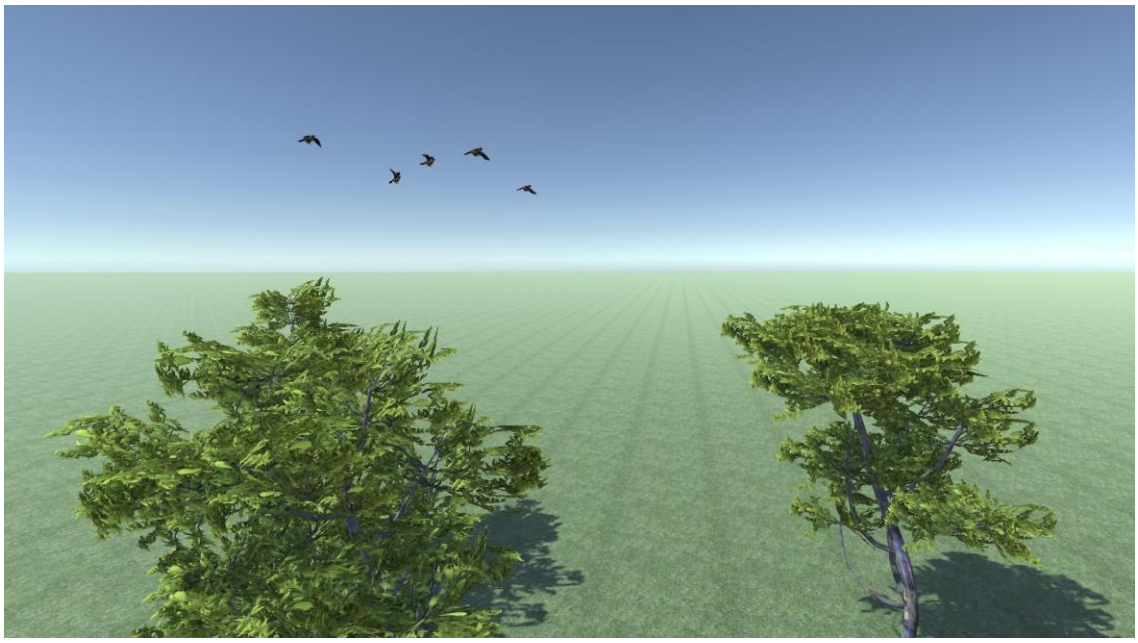
Tekoälyä varten luotiin uusi skripti *BoidFlockingAI*, joka sijoitettiin Unityn demoskeneen tyhjään objektiin. Tarkoituksena on, että jokainen yksilö kysyy tältä skriptiltä uudet voimavektorit fysiikkapäivityksen yhteydessä, jotta jokainen lintu ei tarvitse omaa tekoälyskriptiä suoritettavakseen. Skriptin sisältö on yksinkertainen: siihen kuuluu lintujen luonti demon alussa, lintujen lisääminen ja poistaminen sekä funktiot, joita linnut käyttävät uusien voimavektorien luomiseen. Yksinkertaistettuna tekoälyn toiminta kulkee seuraavasti:

- Lintu suorittaa tekoälyskriptin funktion *GetUpdate(Transform b)*, jossa *b* on linnun Transform-komponentti (kontrolloi objektin fyysisiä ulottuvuuksia).
- *GetUpdate* käy kaikki ympäristön linnut läpi ja lisää niiden sijainnit listaan.
- *GetUpdate* kutsuu nyt kolmea funktiota, joista jokainen luo linnulle uuden voimavektorin. Näitä funktioita ovat
  - *DoSeparation.* Funktio varmistaa, että yksilöt eivät lennä liian lähelle toisiaan laskemalla lähipiirin yksilöiden etäisyydestä kääntäen verrannollisen voimavektorin.
  - *DoAlignment.* Funktio etsii lähipiirin keskimääräisen suuntiman ja ohjaa yksilöä samaan suuntaan.
  - *DoCohesion.* Funktion laskee lähipiirin keskimääräisen sijainnin ja ohjaa yksilöä sen suuntaan.
- Lintuobjektissa oleva *BirdMovement*-skripti kääntää linnun rotaation nopeuden suuntaiseksi ja lisää lintua eteenpäin liikuttavan voiman sekä nosteen.
- Linnun Rigidbody-komponentti yhdistää päivityksen aikana luodut voimavektorit yhdeksi ja kohdistaa lintuun halutun voiman.

BoidFlockingAI-skripti on kokonaisuudessaan liitteenä 1. Skripti on kirjoitettu suoraan tätä projektia varten projektin tekijän toimesta ottaen mallia boid-mallin yleisistä säännöistä ja pohtimalla niille toimiva toteutustapa [13]. *BoidFlockingAI*-skriptin käyttöönottoa varten tarvitaan Unity-projekti, johon lisätään tyhjä objekti, johon skripti kiinnitetään, sekä jonkinlainen objekti kuvaamaan parven yksilöä. Yksilöobjekti sisältää Rigidbody-komponentin ja annetaan skriptille argumenttina. Skripti luo niitä halutun määrän *SpawnBoids*-funktiolla. Huomioitavaa on, että *SpawnBoids* luo objektit ilmaan satunnaisiin paikkoihin, koska se on tehty projektiin nimenomaisesti lintujen luontia varten. Näiden lisäksi tarvitaan enää yksinkertainen skripti, joka liitetään yksilöobjektiin. Skriptin tarkoituksena on kutsua päivitysfunktiossa *BoidFlockingAI*:n funktiota *GetUpdate*, joka muuttaa yksilön suuntaa ja nopeutta muiden yksilöiden tilaan perustuen. Suositeltavaa olisi myös tehdä samaan skriptiin funktio, jolla yksilöt voivat liikkua itsenäisesti. Jos yksilöt ovat esimerkiksi lintuja, kannattaa tehdä nosteen luomiselle yksi funktio, joka olisi yksinkertaisimmillaan esimerkiksi rivi *Rigidbody.AddForce(Vector3.up * -9.81f)* ja eteenpäin liikkuminen rivillä *Rigidbody.AddForce(transform.forward * speed).* Edellä olevat esimerkkirivit ovat pseudokoodia, eivätkä toimi sellaisenaan, vaan vaativat pientä muokkausta skriptistä riippuen.

Jokaiselle tekoälyn tekemälle laskennalle (separation, alignment, cohesion) on määritelty oma kerroin, jolla voimavektori skaalataan laskutoimituksen lopuksi. Näitä kertoimia pystyy säätämään demon pysäytysvalikosta suorituksen aikana, ja ne vaikuttavat merkittävästi parven käyttäytymiseen. Muita reaaliajassa säädettäviä ominaisuuksia ovat muun muassa parven koko ja lähinaapurien etäisyys. Esimerkiksi säätämällä alignment-kertoimen arvoa pienemmäksi lintujen käytös muuttuu selkeästi hajanaisemmaksi ja parveilu vaikuttaa kaoottiselta. Jos arvon asettaa suureksi, tulee parvesta yhteen suuntaan liikkuva pitkänomainen nauha.

Aluksi arvot olivat vakioita ja samat kaikille yksilöille. Tästä johtui esimerkiksi parven hyvin pallomainen muoto ja suoraviivainen lentokäytös (kuva 9). Ongelman ratkaisuksi lisättiin funktioihin satunnaisuutta ja lisäksi jokaiselle linnulle arvotaan linnun luonnin yhteydessä oma ”persoonallisuuskerroin”, joka vaikuttaa kaikkien voimavektorien suuruuteen. Näin lintujen parveilukäyttäytymisestä tuli hyvin orgaanisen tuntuista ja parvien muodot vaihtelevat reilusti, kuten luonnossakin (kuva 10).

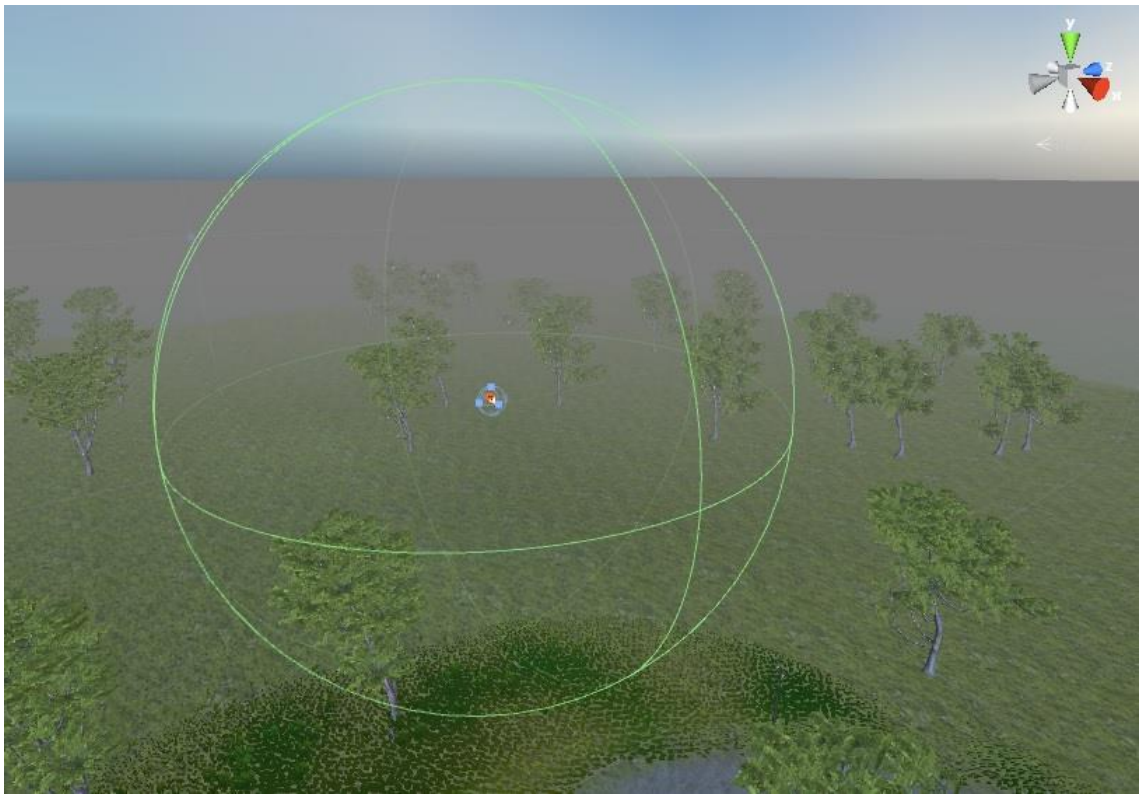


Kuva 9. Projektin alkuvaiheessa linnut lentävät samassa tasossa, koska vaihtelua ei ollut vielä satunnaistettu.

Kuva 10. Parven käytös on paljon vaihtelevampaa persoonallisuuskertoimien lisäämisen ja satunnaistamisen jälkeen.

4.3 Törmäysten havaitseminen

Tärkeä osa demon yleistä toimintaa silmällä pitäen on demoalueen rajojen tunnistus, joka tuntui järkevältä tehdä käyttäen Unityn Collider-komponentteja törmäyksien tunnistukseen. Demoalueen koko on 400 x 400 metriä, ja sen reunoilla on näkymättömiä seiniä, joita linnut väistävät tullessaan kosketuksiin niiden kanssa. Seiniä on myös maan päällä ja korkealla ilmassa niin, että linnut saavat lentää vapaasti niiden ympäröimässä tilassa. Lintujen Collider-komponentti on pallon muotoinen ja reilusti lintua suurempi kappale (kuva 11). Näin linnun tullessa kosketuksiin esteiden kanssa sillä on reilusti aikaa ja tilaa väistää kyseistä estettä. Lintujen välinen etäisyys syntyy tekoälyn *separation*-algoritmin tuloksena, joten sitä ei tarvitse ottaa huomioon. Lintujen Colliderit eivät siis välitä toisistaan.

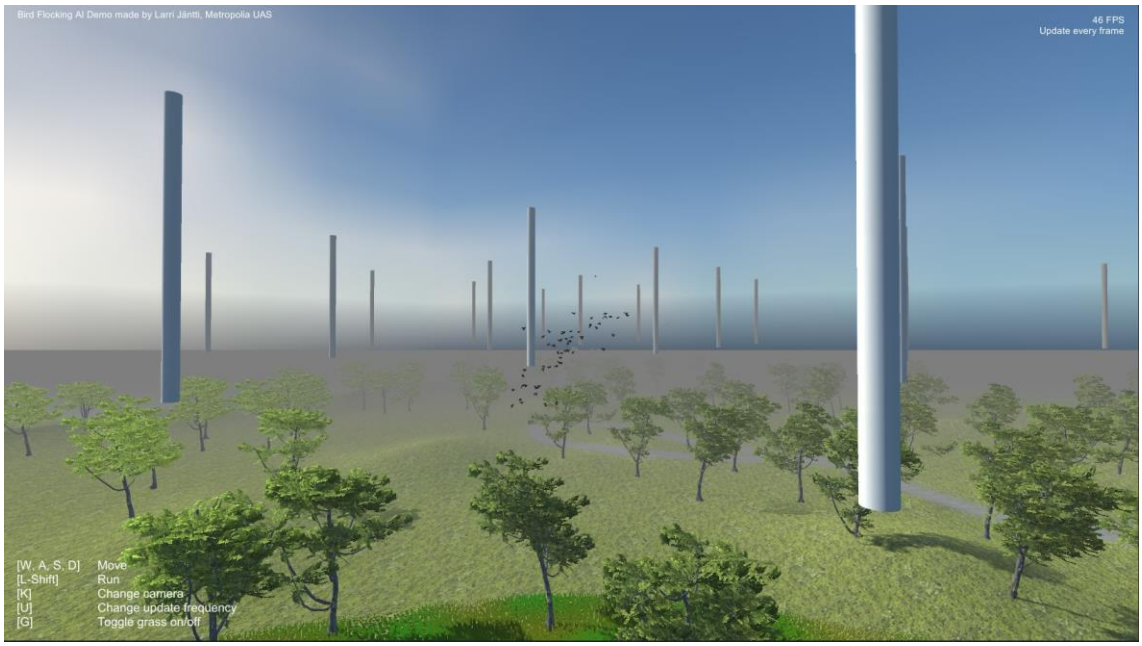


Kuva 11. Lintuobjektin Collider-komponentti (vihreillä viivoilla havainnollistettu pallo) on halkaisijaltaan noin kymmenen metriä.

Törmäysten laskennallinen prosessointi tehdään *BirdMovement*-skriptissä, jossa sitä varten on kaksi funktiota: *OnTriggerEnter(Collider col)* ja *OnTriggerExit(Collider col)*. *OnTriggerEnter* on Unityn sisäänrakennettu funktio, jota kutsutaan automaattisesti, kun jokin toinen Colliderilla varustettu objekti tulee linnun Colliderin sisään. Samalla tavalla

OnTriggerExitiä kutsutaan aina, kun toinen objekti poistuu linnun Colliderin alueelta. Parametrina oleva *col* on törmäävän objektin Collider-komponentti ja sisältää muun muassa tiedon lähimmästä pisteestä lintuobjektista katsottuna. Tätä pistettä voi käyttää vektorilaskennassa, ja linnun suunnan voi näin määrittää suoraan poispäin törmäyksestä. Kun objekti (esimerkiksi demoalueen raja) tulee linnun Collider-alueen sisälle, alkaa linnun ohjausskripti kohdistaa lintuun vastakkaissuuntaista voimaa niin kauan, kunnes objekti poistuu linnun Colliderin alueelta.

Puistoalueella on lisäksi näkymättömiä pylväitä, jotka pyörivät hitaasti puiston keskipisteen ympärillä (kuva 12). Pylväät asetettiin samaan fysiikkatasoon alueen rajojen kanssa, joten linnut joutuvat väistämään niitä eivätkä lennä vain alueen päästä toiseen, mitä voi tapahtua ilman ympäristövaikutuksia. Näin lintujen lentoon tulee aika ajoin odottamatonta turbulenssia, joka voisi luonnossa johtua esimerkiksi toisen lajin linnun havaitsemisesta tai tuulenpuuskasta.



Kuva 12. Alueen ympäri pyöriviä pylväsobjekteja, jotka on asetettu kuvausajaksi valkoiseksi läpinäkyvän sijaan.

4.4 Nälkä

Vaikka lintuparven lento tuntui jo tässä vaiheessa sulavalta ja tarkalta, haluttiin linnuille luoda lisää ärsykkeitä, jotta yksilöt pystyisivät myös erottumaan parvesta ja

muodostamaan uusia parvia. Tarkoitusta varten tehtiin maan tasalle koko puiston mittainen partikkelisysteemi (*Particle System*), jonka tarkoituksena on luoda puiston päälle lentäviä hyönteisiä. Hyönteisiä kuvaavat pienet pallot, joille partikkelisysteemi luo automaattisesti lyhyen ”hännän” perustuen partikkelin reittiin. Säätämällä partikkelien gravitaatiokerrointa negatiiviseksi partikkelit alkoivat leijua ylöspäin maasta. ”Hyönteisten” liikettä saa satunnaistettua säätämällä partikkelien liikkumista kohinan (engl. *noise*) mukaan. Unity antaa vaihtoehdon luoda partikkeleille automaattisen kohinamääritelmän, joka on harmaasävyinen neliön muotoinen 2D-grafiikka, joka satunnaistaa partikkelien liikettä ajan funktiona.

BirdMovement-skriptiin lisättiin muuttuja nälälle, ja sille tehtiin prosessointi *Update*-funktioon. Jos linnun nälkä nousee rajan yli (60 sekuntia), lintu etsii lähimmän hyönteisen ja käy syömässä sen. Linnun nälkä laskee takaisin nollaan, ja se alkaa parveilla normaalisti. Nälkäarvo satunnaistetaan lintua luotaessa jokaiselle linnulle erikseen, muuten kaikki linnut syöksyvät saalistamaan samanaikaisesti, ja vaikka se näyttääkin upealta, ei se tuntunut luonnolliselta vaihtoehdolta. Hyönteisten avulla saatiin lintujen käytökseen hyvää vaihtelevuutta ja linnuille tunnuksenomaisia syöksyjä alkoi syntyä itsekseen. Hyönteispartikkelien määrä myös skaalautuu lintujen määrän mukaan, joten sillä ei pitäisi olla vaikutusta parveiluun lintujen määrää muutettaessa.

4.5 Laskeutuminen ja lentoonlähtö

Lintujen mahdollisuus laskeutua ja lähteä lentoon päätettiin ohjelmoida ja lisätä projektiin vasta viimeiseksi, kun demo tuntui jo muuten laadukkaalta ja valmiilta. Ennen projektin aloittamista suunniteltiin tämän ominaisuuden lisäämistä, mutta työn edetessä ajateltiin sen vievän turhan paljon aikaa ja sisältävän liian monta ongelmaa ratkaistavaksi, joten sitä lykättiin pidemmälle ja lopuksi se jäi melkein tekemättä. Vasta tätä raporttia kirjoittaessa päätettiin, että ominaisuus halutaan osaksi demoa.

Perusajatus laskeutumiskäytöksessä on yksinkertainen: jokaisella linnulla on pieni mahdollisuus (0,15 %) aloittaa laskeutumisrutiini jokainen sekunti. Lintu laskeutuu myös, jos sen välittömässä läheisyydessä (1,5 metrin säteellä) oleva lajitoveri aloittaa laskeutumisen. Alueella on kaksitoista laskeutumisaluetta, jotka asettelin sinne käsin. Alueet ovat suhteellisen avoimia ja puiden välissä. Kun lintu aloittaa laskeutumisrutiinin, se ensin etsii lähimmän alueista ja arpoo alueen ympäriltä kahdeksan metrin säteeltä

laskeutumispaikan. Koska laskeutumispisteen pitää olla maaston päällä, tulee laskuissa ottaa huomioon maaston korkeus laskeutumispisteen kohdalla. Sen saa haettua suoraan *Terrain*-komponentista funktiolla *SampleHeight(Vector3 worldPosition)*, jossa *worldPosition* on linnun laskeutumispiste, jonka korkeuskoordinaatti on 0. Hakeutuminen laskeutumispaikalle toimii samalla tavalla kuin ravinnonhaku muutamaa poikkeusta lukuun ottamatta. Lintu siis hylkää parveilun ja alkaa lentää laskeutumispistettään kohti. Linnun Collider-komponentti on deaktivoitava laskeutumisen ajaksi, ettei lintu ponnahda takaisin ilmaan törmätessään läpinäkyvään seinään, joka indikoi lentotilan pohjaa. Kun lintu on päässyt kolmen metrin korkeuteen, sen Rigidbodyn *drag*- eli ilmanvastusarvoa lisätään huomattavasti ja animaatio muutetaan nopeammaksi räpyttelyksi. Näin linnun liike hidastuu ja se näyttää olevan laskeutumassa. Päästessään puolen metrin päähän laskeutumispisteestä linnun fysiikat otetaan pois käytöstä ja se siirretään maan pinnalle. Tarkka laskeutuminen pelin sisäisiä fysiikoita käyttäen olisi turhan monimutkaista haluttua käytöstä varten. Maan pinnalle päästyä linnun animaatio muuttuu ja lintu pyörähtää satunnaisen asteluvun y-akselin ympärillä, ettei koko laskeutunut parvi katso samaan suuntaan (kuva 13).

Kuva 13. Lintujen animaatio ja rotaatio muuttuvat maan päällä.

Maassa ollessaan linnut seuraavat pelaajahahmon läheisyyttä, ja jos pelaaja kävelee tarpeeksi lähelle, linnut lähtevät lentoon ja aloittavat parveilun uudelleen. Linnut eivät kuitenkaan reagoi lentokameran läheisyyteen, vaan sitä käyttämällä pystyy tarkkailemaan lintujen käytöstä vapaasti. Linnuilla on myös pieni satunnaismahdollisuus lähteä

lentoon omasta tahdostaan. Kuten laskeutumiskäytöksessä, jos yksi linnuista lähtee lentoon, seuraavat muut saman laskeutumisalueen linnut perässä. Jos pelaaja kävelee liian lähelle laskeutumassa olevaa lintua, lintu nousee takaisin lentoon kesken laskeutumisen. Myös jos laskeutumiseen on kulunut odotettua enemmän aikaa, lintu nousee takaisin lentoon olettaen, että joko suuntaus tai ympäristö on vaikuttanut pisteeseen navigointiin tai lintu on jäänyt jumiin. Lintujen lentoonlähtösuunta on puiston origon yläpuolella noin 200 metrin korkeudessa, ja linnut lentävät sitä kohti, kunnes pääsevät tavoitekorkeuteensa. Tämä estää muun muassa ei-toivottuja interaktioita sallitun lentoalueen reunojen kanssa.

Koska myös linnun ympärillä olevat yksilöt vaistoavat laskeutumisen ja laskeutuvat samalle alueelle, on mahdollista, että jopa kokonainen esimerkiksi sadan linnun parvi laskeutuisi yhtäaikaisesti. Vaikka tämä onkin erittäin harvinaista, huomasin silti pienien parvien laskeutumisessa ja lentoonlähdössä epämiellyttävää suoraviivaisuutta, kun koko laskeutumisalueen parvi toimii samanaikaisesti. Siispä lisäsin lentoonlähtöön ja parvilaskeutumiseen satunnaistetun viiveen jokaiselle yksilölle. Näin jotkut yksilöt lähtevät lentoon välittömästi ja toiset vasta esimerkiksi sekunnin tai puolentoista viiveellä. Tämä toi käyttäytymiseen toivottua vaihtelua.

5 Sovelluksen rakenne, testaus ja tulokset

5.1 Tavoite

Insinöörityöprojektia suunnitellessa ajateltiin heti trendiksi nousseita rentoja elämyspelejä. Näissä peleissä ei välttämättä ole tavoitetta tai mielenkiintoisia pelimekaniikkoja, vaan tarkoituksena on luoda pelaajalle lyhyt kokonaisvaltainen hyväntuulinen pelikokemus. Päätettiin siis yrittää luoda mahdollisimman autenttinen ja nautittava ympäristödemo, jossa lintuparvet tulisivat olemaan pääosassa. Tässä muodossa myös parveilutekoälyä on helpompi esitellä ja demonstroida, kun jo ympäristö herättää pelaajan mielenkiinnon. Tekijä oli aikaisemmin jo tutustunut Unityn graafisiin mahdollisuuksiin, joten projektin kaunistamiseen ei kulunut merkittävää aikaa verrattuna itse tekoälyn ja muun käyttäytymisen luomiseen ja säätämiseen.

5.2 Kamerat

Yksi ensimmäisistä työvaiheista oli luoda demoon vapaasti ohjattava lentokamera, jolla ei ole liikunnallisia rajoitteita. Kameraa ohjataan W-, A-, S- ja D -näppäimillä, ja korkeutta voi säätää välilyönnillä ja vasemmalla kontrollilla. Kameran rotaatiota pystyy muuttamaan hiirtä liikuttamalla. Tämä kamera on tärkeä osa tekoälyn käyttäytymisen seuraamista, kun voimavektorien kertoimille yrittää löytää hyviä arvoja tai vaikka seurata lintuparven käyttäytymistä pelialueen rajaa lähestyttäessä. Kameraa liikuttaa yksi ohjelmaskripti, jossa ei ole juurikaan muuta toiminnallisuutta. Kun peruskamera oli saatu toimimaan, alettiin heti kehittää lintujen liikuttamiskomponentteja ja tekoälyosuutta.

Kun myöhemmin saatiin parveilun perusmekaniikka toimimaan, haluttiin luoda ensimmäisen persoonan liikuteltava hahmo, jotta pelaaja voisi kokea kävelevänsä puistossa. Hahmon perustaksi luotiin kaksi metriä korkea kapselin muotoinen 3D-objekti. Pelaajahahmo sisältää lisäksi kapselin muotoisen Colliderin, joka varmistaa, että pelaaja ei putoa maan läpi tai eksy pelialueen ulkopuolelle. Kapselin yläosaan tehtiin uusi kameraobjekti, joka mallintaa hahmon päätä. Tähän kameraan alettiin työstää uutta liikkumiskoodia. Ensimmäisen persoonan liikkuminen osoittautui hieman hankalammaksi, koska hahmon ”pää” oli erillinen objekti, jonka piti kääntyä ylä- ja alasuunnassa (x-akselin rotaatio) itsenäisesti, vaikuttamatta itse hahmon rotaatioon. Tämän lisäksi x-akselin rotaatiota piti rajoittaa niin, että pelaaja ei pysty katsomaan taakseen kääntämällä kameraa tarpeeksi paljon ylös- tai alaspäin. Unity tarjoaa paljon erilaisia tapoja liikuttaa ja kääntää kappaleita. Näistä osa toimii fysiikan rajoitteiden mukaisesti ja osa vain siirtää kappaletta välittämättä ympäristöstä. Paras tulos saatiin käyttämällä Rigidbody-komponentin funktioita *MoveRotation* ja *MovePosition*. Koska Rigidbody on kappaleen fyysisen käyttäytymisen perusta, päätös erilaisten ratkaisujen välillä oli helppo.

Vaikka ensimmäisen persoonan kamera toimi hienosti, siihen haluttiin vielä lisää autenttisuutta. Päätettiin lisätä kameraa liikuttavaan koodiin vielä funktio, joka liikuttaa kameraa ylös ja alas (y-akselin suuntaisesti) hahmon kävellessä. Tämän liikkeen tarkoituksena on kuvata kävelemisestä johtuvaa aaltoilevaa ruumiin liikettä. Idean pystyy helposti toteuttamaan käyttämällä sinikäyrää kameran y-sijainnin muuttamiseen hahmon liikkuessa (esimerkkikoodi 2).

```
public void DoHeadBob()
{
    float running = sprinting ? sprintingModifier : 1.0f;
    float headBobValue = Mathf.Abs(Mathf.Sin(headBobCounter *
          headBobLengthModifier * running)) * headBobHeightModifier;
    transform.localPosition = originalPosition +
          new Vector3(0, headBobValue, 0);
}
```

Esimerkkikoodi 2. Funktio liikuttaa kameraa sinikäyrän, halutun askelkorkeuden ja pelaajan nopeuden mukaan.

Koska alue oli iso, huomattiin, että pelaajahahmo ei pysynyt lintujen perässä, joten lisättiin mahdollisuus juosta pitämällä vasenta vaihtonäppäintä pohjassa. Juokseminen muuttaa pelaajan nopeuden kaksinkertaiseksi ja on olennainen osa monia pelejä. Ensimmäisen persoonan kamerassa on myös muita ominaisuuksia, kuten sivuttaisliikkumiseen vaikuttava nopeusrangaistus.

Valmiissa demossa molemmat kamerat ovat käytettävissä ja niiden välillä voi siirtyä painamalla K-painiketta. Oletuksena pelaaja aloittaa demon ensimmäisen persoonan kamerassa. Molemmat kamerat ovat erillisiä objekteja, joten pelaajahahmoa esittävään kapseliin liitettiin alueen rajoissakin käytetty grafiikkavarjostin, joka tekee kappaleesta läpinäkyvän. Näin pelaaja ei näe kapselia, vaikka vaihtaisi toiseen kameraan ja katsoisi sitä kohti. Ensimmäisen persoonan kameran aloituspiste on lähellä puiston keskikohtaa, ja lentokamera sijaitsee noin kolmenkymmenen metrin korkeudessa sen yläpuolella.

5.3 Ympäristö

Ulkonäkö ja visuaalisuus

Ensimmäisenä pelinäkymää luodessa on yleensä hyvä tehdä taso-objekti kuvaamaan maanpintaa tai lattiaa. Niin tässäkin projektissa. Objektien perusväritys on vaaleanharmaa, joten siihen etsittiin Internetistä ensin jonkinlainen ruohoa muistuttava, väliaikainen 2D-tekstuuri, jotta ympäristön syvyys erottuisi paremmin. Kun näkymään oli tehty ensimmäinen liikuteltava kamera, selattiin Unityn Asset Store -kauppapaikkaa etsien miellyttävän näköisiä lehtipuumalleja. Päädyttiin käyttämään ”Realistic Tree 9” -pakettia tekijältä Rakshi Games. Demonäkymään lisättiin muutama puu, että saatiin paremmin selvää ympäristön skaalasta alettaessa tuoda lintumalleja projektiin. Lintujen 3D-malli, animaatiot ja äänet ovat paketista ”Living Birds” käyttäjältä dinopunch. Projektin tässä vaiheessa käytettiin seuraavat päivät tekoälyn ja lintujen liikkumisen tekemiseen.

Taivasgrafiikaksi Unityn oman *Skyboxin* tilalle ladattiin ”Wispy Skybox” -paketti, jonka oli Asset Storeen ladannut Mundus Limited. Vielä lisäksi etsittiin puistoon kuuluvia 3D-malleja valaisimista ja penkeistä, ja ne löytyivät paketista ”Park Props Pack” käyttäjältä Dimonati, ja aluskasvillisuus saatiin paketista ”Grass And Flowers Pack 1” käyttäjältä Vladislav Pochezhertsev. Kaikki käytetyt paketit ovat Unityn virallisen Asset Storen ilmaisjakelussa, ja niiden nimet, tekijät ja lähteet on sisällytetty liitteeseen 2.

Vaikka graafinen ulkoasu alkoi muuten miellyttää silmää, oli maanpinta vielä täysin tasainen ja teksturoitu yhdellä tekstuurilla. Unity sisältää monipuolisen työkalun Terrain Editor (maastomuokkain), jolla pystyy luomaan maastonmuotoja ja istuttamaan puita tai aluskasvillisuutta helposti. Ensin luotiin 400 x 400 metrin kokoinen Terrain-alue demonäkymän nollakorkeuteen ja alettiin nostattaa pieniä korkeuseroja pinnasta. Puiston poikki kulkeva kahdeksikon muotoinen alue jätettiin nollatasoon, ja siitä päätettiin tehdä sorapolku, joka kulkee puiston läpi. Maastomuokkaimessa onnistui helposti maalata polulle tarkoitettu osio soratekstuurilla, ja loput maastosta peitettiin uudella ruohotekstuurilla, joka sisältyi käytettyyn kasvillisuuspakettiin Grass And Flowers Pack 1. Kun muodot oli saatu toimiviksi, istutettiin maastoon erilaisia ruoho- ja aluskasvillisuusobjekteja ja lisää puita. Puut ja ruoho tulivat vielä paremmin eloon, kun näkymään lisättiin tuulialue (*Wind Zone*), joka luo demoon realistista, hieman puuskittaista tuulta. Viimeisenä visuaalisena parannuksena tehtiin taivaalle valonlähde ja ladattiin Unityn oma linssiheijastuskokoelma, jotta saatiin vakuuttavan näköinen Aurinko. Valaistusasetuksista löytyy myös vaihtoehto sumun lisäämiselle, ja sen todettiin rajoittavan ja hämärtävän näkyvyyttä säästeliäästi käytettynä. Projektin lopullinen graafinen ulkoasu on luonnollisen yksinkertainen olematta kuitenkaan liian realistinen (kuva 14).

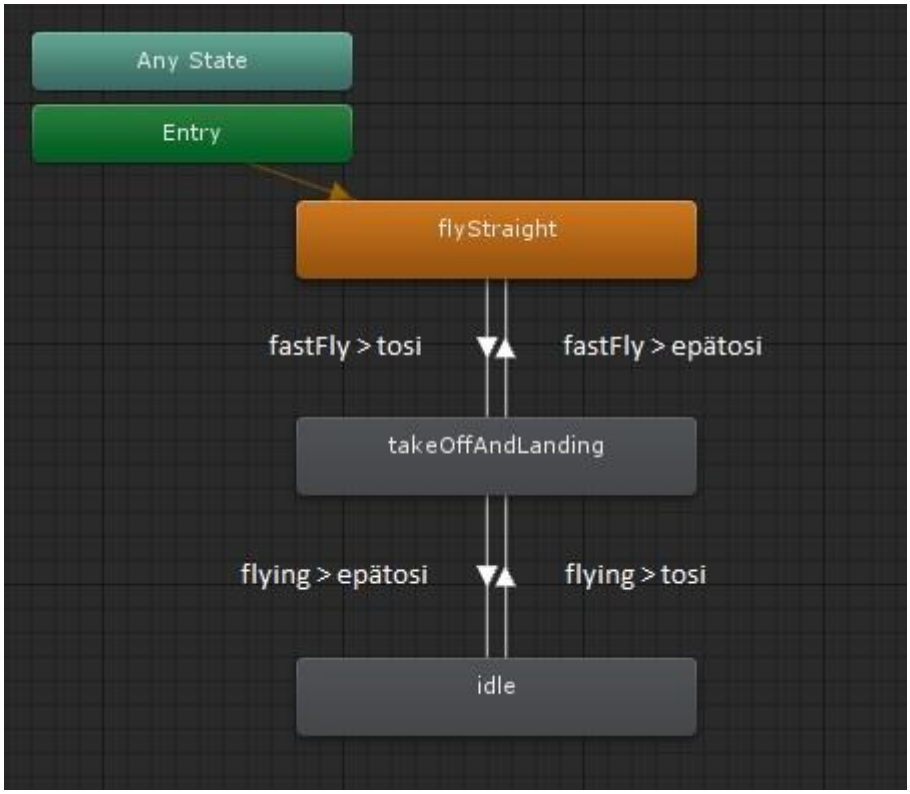
Kuva 14. Insinöörityössä tehdyn sovelluksen viimeistelty ulkoasu.

Projektin loppupuolella saadun palautteen perusteella lisättiin vielä puistonpenkkien päälle kameraobjektit. Nyt kun pelaaja kävelee tarpeeksi lähelle penkkiä ja katsoo sitä kohti, ilmestyy näytölle ohjeteksti istuutua painamalla E-näppäintä. Kun pelaaja istuutuu, linnut eivät enää havaitse pelaajaa, vaan saattavat laskeutua penkin lähistölle. Tämän ominaisuuden innoittamana lisättiin jokaisen penkin lähelle uusi lintujen laskeutumisalue ja deaktivoitiin muutama vanha alue puiston laidoilta. Kun pelaaja nousee ylös penkiltä, toimivat vanhat mekaniikat oletetusti ja lähistölle mahdollisesti laskeutuneet linnut pyrähtävät lentoon. Teknisesti istuutuminen tapahtuu deaktivoimalla ensimmäisen persoonan kamera pelaajahahmosta ja aktivoimalla kohti katsotun penkin päällä oleva kamera. Unity vaihtaa kameranäkymää automaattisesti, koska penkin kamera on ainoa aktiivinen kamera. Kun pelaaja painaa E-näppäintä uudelleen, siirtyy kamerafokus takaisin pelaajahahmon kameraan ja penkin yllä oleva kamera suljetaan.

Animaatiot

Kaikki animaatiot sisältyivät Living Birds -pakettiin. Animaatioista käytettiin maassa seisovaa, ”huolestunutta” lintua ja oletusarvoista lentoanimaatiota. Kolmanneksi animaatiotilaksi lisättiin sama lentoanimaatio kaksinkertaiseksi nopeutettuna. Tämän oli tarkoitus kuvata laskeutumisen loppuvaihetta ja lentoonlähtöä, ja se toimikin mukavasti tähän tarkoitukseen. Animaatioita ohjaa Unityn Mecanim-animaattori. Kuten mainittua, animaattorissa on kolme tasoa ja siirtymät niiden välillä. Lentävästä linnusta on siirtymä

nopeaan lentoon (laskeutuminen), josta on siirtymät joko maassa olevaan lintuun tai takaisin lentävään. Maassa olevasta linnusta on siirtymä takaisin nopeaan lentoon (lentoonlähtö). Animaatiota ohjataan kahden totuusarvomuuttujan, *flying* ja *fastFly,* avulla kuvan 15 mukaisesti. Esimerkiksi kun animaattorin tila on ”*fastFly* = epätosi, *flying* = tosi”, lintu on lennossa. Kun lintuun liitetty ohjelmakoodi muuttaa animaattorin muuttujan *fastFly* arvoon tosi, muuttuu linnun animaatio nopeaan lentoon. Kun laskeutuminen on suoritettu, linnun koodi muuttaa animaattorin muuttujan *flying* arvoon epätosi, joka asettaa linnun seisonta-animaatioon. Lentoonlähtösekvenssi taas on sama toiseen suuntaan, eli ensin *flying* asetetaan arvoon tosi ja kun lintu on saavuttanut tavoitekorkeuden, asetetaan *fastFly* arvoon epätosi.



Kuva 15. Linnun kolmitilainen animaatio-ohjain, johon lisätty selitteet tilojen muutoksien laukaisimista.

Äänet

Puisto ja lintusimulaatio ilman ääniä ei tuo kovin uskottavaa tunnelmaa. Ladattu lintupaketti sisältää lintujen 3D-mallien ja animaatioiden lisäksi myös äänileikkeitä kyseisille lintulajeille. Lintuobjekteihin lisättiin *Audio Source* (äänilähde) -komponentti, ja äänien toistamista varten tehtiin uusi skripti. Alkuperäisenä ajatuksena oli ollut saada linnun laulu

laukeamaan toisten lintujen äänestä, mutta se osa jätettiin tekemättä, koska pelkkä satunnaistoisto toimii erinomaisesti. Jokaisella linnulla on pieni mahdollisuus (0,5 %) toistaa äänileike joka sekunti, jollei leike ole jo soimassa. Tämä tuntui sopivalta arvolta monenkokoisille parville. Lintujen lähettämä ääni simuloitiin 3D-tilaan, ja se vaimenee etäisyyden kasvaessa, eli pelaaja ei kuule toisella puolella puistoa olevan linnun laulua kovinkaan voimakkaana.

Lintujen äänttelyn lisäksi puistosta puuttuivat pelaajan askelten äänet. Pelaajan kävellessä hahmon pää liikkuu askelten mukana, joten samaan funktioon oli helppo lisätä äänileikkeen toisto aina, kun askelkorkeus kävi askelkorkeuden pohjalukemassa. Äänileike toistuu näin myös aina pelaajan pysähtyessä ja kaksi kertaa nopeammin pelaajan juostessa. Pelaajan askelia varten lisättiin hahmon jalkojen kohdalle tyhjä peliobjekti, johon liitettiin uusi äänilähdekomponentti. Myös pelaajan askeleet on konfiguroitu 3D-ääniksi, vaikkakin äänten kuulija (kamera) on aina samassa sijainnissa äänilähteeseen nähden. Askeläänet sisältyivät Asset Storen pakettiin Footsteps (Snow and Grass) käyttäjältä MGWSoundDesign. Äänimaailmaa täydennettiin lisäämällä vielä ympäristöön sopiva tuuliraita, joka kuulostaa samalta jokaisessa puiston kohdassa ja toistuu automaattisesti.

5.4 Sovelluksen suorituskyky

Boid-mallin suorituskykyrajoite huomattiin projektissa, kun ohjelmoitiin mahdollisuus lisätä ja poistaa lintuja parvesta. Jo ylitettäessä neljänkymmenen linnun raja parvikoossa alkoi pelin päivitystaajuus hidastua tavoitellusta 60 FPS:stä. Ajateltiin, että koska parven tekoälyn suorituskyky on neliöön verrannollinen ja se tiedettiin hitaaksi, ei sitä pystyisi tehostamaan ottamatta käyttöön kolmiulotteisia tiedostorakenteita, jotka ovat monimutkaisempia ohjelmoida kuin itse tekoäly. Myöhemmin tarkasteltaessa koodia ja tehtäessä optimointimuutoksia löydettiin syypää huonolle suorituskyvylle. Testattaessa eri algoritmeja, ei ollut yhtenäistetty naapurihakua, vaan jokainen lintu haki naapurilistan uudelleen jokaisen algoritmin osan (alignment, separation, cohesion) laskufunktiossa. Funktiorakennetta muutettiin niin, että samaa naapurilistaa käytetään jokaisessa laskufunktiossa, ja parven maksimikoko nousikin nelinkertaiseksi.

Sovelluksen lopullinen suorituskyky PC-kokoonpanolla (AMD Ryzen 7 1700, kahdeksan ydintä 3,7 GHz sekä Nvidia GTX 1070) on reilusti odotettua parempi. 170 linnun parvi toimii 60 FPS:n päivitystaajuudella myös sovelluksen lopullisilla ympäristögrafiikoilla,

joihin kuuluu muun muassa reilu aluskasvillisuus sekä tuulialueet ja korkealaatuiset valaistus- ja varjoasetukset. Kun otettiin mukaan laskeutumista ja lentoonlähtöä laskevat funktiot, suorituskyky alkoi laskea hieman päivitysfunktiossa tehtävien satunnaislaskutoimitusten vuoksi. Kaikki todennäköisyyslaskelmat muutettiin toimimaan sekuntipohjaisesti, joten nyt esimerkiksi linnunlaulun todennäköisyyden tulisi olla sama sekä 15 FPS:n että 60 FPS:n nopeudella ja muutos säästää myös suorituskykyä. Unityn *Profiler*-työkalun mukaan prosessorin suoritusajasta noin 70 % menee pelkästään lintujen liikuttamiseen tehdyn *BirdMovement*-skriptin päivitysfunktion suoritukseen. Sitä kautta suoritetaan kaikki tärkeimmät lintua koskevat toiminnot, kuten parveilu, saalistus ja laskeutuminen sekä lentoonlähtö. Grafiikoiden piirtäminen vie suurimman osan lopusta suorituskyvystä.

Tekoälyä optimoitiin antamalla käyttäjälle vaihtoehto muuttaa lintujen tekoälyn päivitystaajuutta. Se määrittää, haetaanko linnuille uusia voimavektoreita joka päivityssyklin yhteydessä vai kenties harvemmin. Jos käyttäjä haluaa simuloida isompia parvia, voi päivitystaajuutta pienentää kuitenkaan vaikuttamatta itse algoritmin toimintaan. Parveilusimulaatio ei ole yhtä tarkkaa, mutta suurille parville tai esimerkiksi pelin taustaelementiksi se on hyvä vaihtoehto, jollei tahdo siirtyä käyttämään esimerkiksi swarm-tekoälyvariaatiota. Päivitystaajuuden pienentäminen vaatii tekoälyn käyttäytymisen kannalta voimakertoimien suurentamista, mutta parvikäyttäytymisessä ei silti esiinny huomattavia eroja. Parvikoon kasvaessa lisääntyvät tietenkin myös muut suorituskykyä rajoittavat ominaisuudet, kuten lintujen animointi, äänikomponenttien suoritus, varjojen määrä ja lintujen yksilökohtaiset funktiot.

Aluskasvillisuuden saa pois käytöstä painamalla näppäimistön painiketta G. Jos tietokoneessa ei ole erillistä grafiikkasuoritinta, aluskasvillisuuden piilottaminen saattaa nostaa suorituskykyä huomattavasti. Sovellusta käynnistettäessä pystyy myös valitsemaan muun muassa valaistukseen ja varjoihin sekä reunanpehmennykseen vaikuttavan pelin yleisen graafisen laadun asetuksen sekä piirtoresoluution.

5.5 Kehityksenaikainen testaus

Kehitys tehtiin täysin omatoimisesti, ja niin myös kaikki tekoälyn sekä demon testaaminen, kehityksenaikainen arviointi ja parantelu sekä lisäominaisuuksien tarve on pääasiassa henkilökohtaista. Tekoälyn voimavektorien kertoimien arvojen löytäminen oli

erittäin hankalaa aika-ajoin, ja se oli itse asiassa suurin syy sille, että pysäytysvalikkoon lisättiin mahdollisuus säätää näitä arvoja ajonaikaisesti. Siinä missä arvojen suuruusluokan löytäminen oli yksinkertaista, oli yllätys, kuinka pienikin muutos arvojen suhteissa sai välillä aikaan suuria muutoksia koko parveilun muodossa ja käyttäytymisessä. Referenssiksi annettakoon, että parveilumuuttujien oletusarvot ovat tällä hetkellä seuraavat:

- lintuyksilöiden määrä: 70
- lähiympäristön etsintäetäisyys: 30 metriä
- erotusalgoritmin kerroin: 0,1
- suuntausalgoritmin kerroin: 0,2
- yhtenäisyysalgoritmin kerroin: 1,0.

On toki huomioitava, että vaikka arvoja muuttamalla parveilun käyttäytyminen muuttuu, se ei tarkoita, että nämä asetetut oletusarvot luovat lähinnä luonnollista olevan parveilumallin. Vaikka mallia otettiinkin oikeiden lintujen parveilusta luonnossa, lopuksi arvoja säädettiin omien mieltymysten mukaisesti. Toisaalta projektin hienoin osa onkin ehkä juuri se, että parveilukäytöstä pystyy muokkaamaan oman mieltymyksen tai mielenkiinnon mukaisesti.

Kehityksessä ilmenneitä ongelmia

Lopuksi voidaan todeta, että projektin ongelmien määrä oli erittäin vähäinen. Suurin osa ominaisuuksista toimi joko suoraan ohjelmoinnin jälkeen tai pienellä arvojen säätämisellä. Tämä tietenkin johtuu pääasiassa siitä, että demosovelluksen laajuus ei ole kovin huomattava ja yhteensä koodirivejä projektissa on vain noin 1 600. Kehityksessä ilmeni kuitenkin muutamia ongelmia, joita piti pohtia usean tunnin ajan tai jotka vaativat erikoisen ratkaisun.

Yhtenä ongelmana mainittiinkin jo aiemmin boid-mallin suorituskyky, jonka luultiin rajoittavan parvikokoa huomattavasti. Optimointeja tehdessä havaittiin kuitenkin, että jokaista yksilöä kohden tehtiin kaksi ylimääräistä naapurihakua for-silmukkaa käyttäen. Virheen korjauksen jälkeen parvikoko oli itse asiassa suurempi kuin oli odotettu projektia aloittaessa.

Eräs mielenkiintoisemmista ongelmista, joka ratkaistiin vasta tätä raporttia kirjoittaessa, oli ensimmäisen persoonan kameran satunnainen sivuttaisliike. Siihen etsittiin projektin

aikana vastauksia ja tyydyttiin erääseen foorumikommenttiin, joka kertoi analogisen peliohjaimen saattavan aiheuttaa syötehäiriöitä, jos sellainen oli kiinni tietokoneessa. Koska sellainen ohjain sattui olemaan kiinni tietokoneessa, sivuutettiin ongelma hetkeksi. Kuitenkin kun ohjain hieman myöhemmin irrotettiin, huomattiin että pieni sivuttaisliike jatkui yhä ja sitä koetettiin korjata ohjelmallisesti, kunnes todettiin, että ongelma ei johtunut kameraa liikuttavasta skriptistäkään. Lopullisena ratkaisuna rajoitettiin pelaajahahmon Rigidbody-komponenttia niin, että se ei pysty kääntymään ulkopuolisten voimien seurauksena (*Constraints* > *Freeze Rotation* -vaihtoehto). Tämä auttoi, ja oletetaan, että hahmon kääntyminen saattoi johtua tuulen vaikutuksesta tai maaston muodoista, sillä pelaajahahmon Collider on kapselin muotoinen, eli siis pyöreäpohjainen.

Toinen askarruttava ongelma oli, kun projektin loppuvaiheessa lisättiin puiston ylle Aurinko ja siihen liitettiin Unityn linssiheijastusefekti. Efekti kuitenkin katosi välillä näkyvistä kokonaan. Ongelma tuntui tapahtuvan sitä enemmän, mitä kauempana valonlähteestä kamera oli. Lyhyen pohdinnan jälkeen muistettiin, että lintuobjektien Collider-komponentti ulottuu viisi metriä linnun ympärille (s. 15, kuva 11). Asiaa testattiin ja todettiin, että tosiaankin ylilentävien lintujen Colliderit piilottivat linssiheijastusefektin. Asia oli helposti korjattu, kun *Lens Flare* -komponentin *Ignore Layers* -listaan lisättiin taso, jolla linnut olivat. *Ignore Layers* määrittelee linssiheijastukselle tasot, joiden läpi se näkyy. Näitä voi käyttää esimerkiksi ikkunalasin simuloimiseen peleissä (fyysinen objekti, josta kuitenkin näkee läpi).

Projektista saatu palaute

Lähes valmis demo annettiin testattavaksi sekä Hisomen edustajalle että ulkopuoliselle testaajalle. Hisomen edustaja ehdotti muun muassa parviliikkeen yhtenäisyyden lisäämistä ja parven johtajan ottamista yhdeksi parveiluparametriksi. Tätä metodia on esitelty myös joissakin luetuissa lähteissä, kuten Bourin ja Seemannin teoksessa AI for Game Developers [21]. Tekoälyyn koetettiin lisätä johtajaparametri, mutta sillä ei ollut mitään positiivisia vaikutuksia parvikäyttäytymiseen. Lintuparvet ovat luonnostaan melko tasapainoisia, ja yleensä johtajaa käytetäänkin tavoitepohjaiseen parveiluun, kuten esimerkiksi strategiapeleissä joukkojen liikuttamiseen kahden pisteen välillä. Demon lintuparvi on kuitenkin vapaa lentämään koko ympäristön alueella, ja koska johtaja yhä saa suuntansa muulta parvelta, ei johtajan asettaminen ollut johdonmukaista. Jos johtajan otti pois parveilutekoälyn piiristä, muut linnut kyllä pyrkivät seuraamaan sitä, mutta itse johtajan liike oli poukkoilevan äkkinäistä ja suoraviivaista. Johtajan asettaminen olisi voinut

toimia, jos lisäksi olisi tehty myös pelkästään johtajan liikettä ohjaava ylimääräinen skripti. Palautetta tuli myös pysäytysvalikon säätimistä, sillä termejä ei ollut selitetty missään. Asiaan vihkiytymätön ei siis tiedä, mitä mikäkin säädin tekee, ja osittain tämän inspiroimana tehtiin pieni infonäkymä pysäytysvalikon sivuun käyttäen yleisiä termejä ja alkuperäisen mallin havainnekuvia, jotka auttoivat itseäni ymmärtämään aihetta paremmin [13].

Toinen ulkopuolinen testaaja, jolla on vain niukasti mielenkiintoa ja ennakkotietoa tietotekniikan saralta, antoi myös kuitenkin merkittävää palautetta. Hänen mukaansa demon ulkoasu on onnistunut ja vaikuttaa puistolta. Hän olisi toivonut lisää lähinnä graafiseen ulkoasuun ja puiston esineistöön liittyviä asioita, kuten linnuille juoma-allas tai lampi, taivaalle pilviä ja vaihtuva päivänaika (koska olihan puistossa katulamppuja). Näistä oli jo ajateltu esimerkiksi pienen kalalammen lisäystä, missä voisi demonstroida toisenlaista, esimerkiksi swarm-pohjaista tekoälyä pienellä kalaparvella, mutta projekti päätettiin rajata pelkästään lintuihin ja boid-mallin hiomiseen. Sen lisäksi tekijällä ei ole yhtään henkilökohtaista kokemusta veden luonnista Unityssa, joten asia jäi vain varhaiselle harkinta-asteelle.

Saadun palautteen pohjalta projektiin tehtiin kuitenkin hieman lisäsisältöä, kuten istuttavat penkit, ja myös näytön alanurkassa näkyviä ohjetekstejä paranneltiin. Esimerkiksi Esc-näppäimestä aukeavasta asetusvalikosta ei ollut mitään mainintaa ohjeteksteissä, vaikka se on olennainen osa projektia. Näiden lisäksi ladattiin Unityn Asset Storesta uusi paketti ”Simple Dynamic Clouds” käyttäjältä Butterfly World, ja se lisättiin nopeasti projektiin. Paketti sisälsi ennalta sekä ylä- että alapilviä, mutta lintujen todettiin katoavan hieman taustaan, jos molemmat pilviryhmät olivat aktiivisina. Niinpä puiston ylle jätettiin pelkät pehmeät alapilvet ja lisättiin vaihtoehto näyttää tai piilottaa pilvet painamalla näppäintä C. Pilvet generoidaan reaaliajassa satunnaisen 2D-tekstuurin avulla, ja tekijän mukaan niiden vaikutus suorituskykyyn on minimaalinen. Testaaja oli muutoksiin tyytyväinen.

5.6 Tulokset, analyysi ja tulevaisuus

Projektin valmis demo sisältää seuraavat ominaisuudet:

- täysin toimiva ja ”oikeaoppinen” boid-mallin parveilutekoäly

- yksilökohtainen ympäristön esteiden havainnointi
- laskeutuminen ja lentoonlähtö perustuen satunnaisuuteen ja pelaajan läheisyyteen sekä parven muiden lintujen käytökseen
- yksilökohtainen nälkämittari ja saalistusmahdollisuus
- ympäristön graafinen ulkoasu ja elementtejä kuten tuulialue sekä dynaaminen aluskasvillisuus ja valaistusefektit
- kattava äänimaailma
- kaksi liikkuvaa pelaajakameraa ja kolme istuttavaa penkkiä (kameraa)
- pysäytysvalikko, jossa voi säätää parveilukäyttäytymistä reaaliajassa, sekä infotaulu säätöjen vaikutuksesta.

Projektin tavoitteisiin nähden demon sisältö on kattava, eikä mitään yksittäisiä projektiin suunniteltuja osia jäänyt puuttumaan. Parveilutekoäly toimii kuten sen on tarkoitus, ja arvojen muuttaminen pysäytysvalikosta muuttaa parvikäyttäytymistä odotetusti. Lisäsisällön määrään ja laatuun voi olla tyytyväinen, eikä itse metodeissa tai niiden toteutuksessa jäänyt mitään epäselvyyksiä. Vaikka ympäristöllisiä ”lisäpaketteja” onkin lisätty paljon, ne ovat kaikki projektissa ohjaamassa huomiota itse aiheeseen, eli lintujen dynamiikkaan ja käyttäytymiseen. On hämmästyttävää, miten vähällä koodimäärällä voi saada aikaan luonnollisen näköistä parvikäyttäytymistä. Toisaalta on mielenkiintoista myös ajatella, että luonnossa parvikäyttäytyminen seuraa pääasiassa näitä samoja yksinkertaisia sääntöjä.

Tulevaisuudessa ei koko projektille luultavasti löydy paljon käyttöä sellaisenaan, mutta se antoi kuitenkin paljon uutta kokemusta ja tietoa monimuotoisesta tekoälyimplementoinnista pelimaailmassa. Demosta ei aiota kehittää kaupallista versiota, mutta se saatetaan laittaa Internetiin ladattavaksi, jos kiinnostusta löytyy. Melko varmasti parveilumallia tullaan käyttämään ainakin taustaefektinä tulevissa projekteissa. Se luo omanlaisensa immersion pelin tunnelmaan ja vähentää suoraviivaisuutta lisäten ainakin näennäistä satunnaisuutta kokemukseen.

Jos jatkaisin sovelluksen kehitystä, lisäisin luultavasti lintujen yksilökäytökseen liittyviä muutoksia, kuten laskeutumismahdollisuuden puiden oksille sekä janomittarin, ja tekisin puistoon juoma-altaita linnuille. Pelaajalle voisi myös esimerkiksi antaa mahdollisuuden heittää maahan ruokaa, jotta linnut laskeutuisivat pelaajan lähelle ruokailemaan. Pelialuetta voisi myös suurentaa hieman, muttei kuitenkaan niin paljon, että lintujen paikallistaminen menisi liian vaikeaksi. Kalalampi oli yksi lempiehdotuksistani, mutta jäi

toteuttamatta ajanpuutteen takia. Parveilun puolesta ja kokemuksellisella tasolla kuitenkin olen sitä mieltä, että projekti on jo maalissa.

6 Yhteenveto

Insinöörityön tavoitteena oli luoda pelidemoon luonnolliselta vaikuttava ympäristöelämys, jonka pääosassa on lintujen boid-pohjainen parveilutekoäly. Demoprojektista oli tarkoitus tehdä kokemuspohjainen ja havainnollistava kokonaisuus, joka esittelee parveilutekoälyn toimintaa avoimessa ympäristössä ja antaa käyttäjälle mahdollisuuden muokata parven käyttäytymistä ajonaikaisesti. Parveilutekoälyn laajentamiseksi aiottiin tehdä törmäysten havainnointia, laskeutumis- ja lentoonlähtömekaniikka sekä jonkinlainen nälkämittari ja ruokailumetodi. Projekti tehtiin omasta mielenkiinnosta aihepiiriä kohtaan, ja sen tulokset on tilannut pelialan startupyritys Hisome Oy. Kaikki projektin osat tulivat valmiiksi määräaikaan mennessä, karsimatta tuloksen laadusta.

Projektin lopputulokseen ja onnistumiseen voi olla erittäin tyytyväinen. Samanaikaisesti on havaittavissa hieman pettymystä, kuinka nopeasti parveilutekoälyn voi yksinkertaisimmillaan tehdä: siihen riittää minimissään yksi satarivinen skripti ja jonkinlainen kappale edustamaan parven yksilöä. Olikin hyvä päätös laajentaa klassista parveilutekoälyä muun muassa törmäysten havaitsemisella, nälkämittarilla ja saalistuksella, lintujen persoonallisuusarvoilla ja ennen kaikkea laskeutumistoiminnallisuudella ja graafisella ympäristöllä. Demo ei kuitenkaan kuvaannollisesti räjähtänyt käsiin, vaan alkusuunnitelmien tavoite oli aina kirkas ja projekti kulki nopeasti sitä kohti. Olisi mielenkiintoista työskennellä vastaavanlaisten pelin sisäisten tekoälyjen parissa tulevaisuudessa, mikä olikin osasyynä aiheen valinnalle.

Projektityöskentelyä olisi voinut tehostaa esimerkiksi käyttämällä päivän kirjallisen ja yksityiskohtaisen suunnitelman tekemiseen, mutta koska projekti tehtiin kokonaisuudessaan yksilötyönä, pysyi paketti hyvin koossa loppuun asti. Projektiin paneutuminen oli ajoittain hajanaista töiden takia, mutta demoa pyrittiin tekemään yhtenä tai kahtena päivänä viikossa pitkän aikaa. Lopuksi demo viimeisteltiin tätä raporttia kirjoitettaessa viikon vapaalla töistä. Projektityöstä tulikin jossain määrin jopa suunniteltua laajempi, vaikka kaikkia ajateltuja ominaisuuksia ei lähdettykään toteuttamaan. Jokaiseen projektiin saa loputtomasti lisäsisältöä, jollei sitä pysty itse rajaamaan.

Parveilutekoälyn luominen on ehdottomasti vaivan arvoista, jos sitä esimerkiksi peliin harkitsee tarvitsevansa. Perustoiminnallisuus on nopeasti ohjelmoitu (aloittaa voi vaikka liitteenä 1 olevasta skriptistä), ja käyttäytymisarvoja säätämällä parveilumuotoon ja -käytökseen saa paljonkin vaihtelevuutta, jopa ajonaikaisesti. Kaiken kaikkiaan, kappaleiden parveilu takaa mielenkiintoisen ja näennäisen satunnaisen ympäristömuuttujan, jota moni pelaaja ja käyttäjä osaa varmasti arvostaa. Vähemmän näkyviä, mutta tärkeitä implementaatioita voisi harkita esimerkiksi hävittäjäparven tai -laivueen liikuttamiseen tai strategiapelin joukkojen hallintaan.

Lähteet

1 Swarup, Prakhar. 2012. Artificial Intelligence. International Journal of Computing and Corporate Research, Vol. 2 (4).

2 Mohit, Saiwan ym. 2014. Flocking Behaviour Simulation: Explanation and Enhancements in Boid Algorithm. International Journal of Computer Science and Information Technologies, Vol. 5 (4).

3 Farley, Elizabeth A. ym. 2008. Characterizing complex mixed-species bird flocks using an objective method for determining species participation. Dt. Ornithologen-Gesellschaft.

4 Turdus migratorius. The IUCN Red List of Threatened Species 2016. Verkkoaineisto. International Union for Conservation of Nature. <http://www.iucnredlist.org/details/103889499/0>. Luettu 19.3.2018.

5 Haku ”robin flock”. Verkkoaineisto. Youtube. <https://www.youtube.com/results?search_query=robin+flock>. Luettu 26.3.2018.

6 Södersved, Jan (päätoim.). 2007. Luonnossa: Linnut 2. Helsinki: Weilin+Göös.

7 Hemelrijk, Charlotte K. & Hildenbrandt, Hanno. 2011. Some Causes of the Variable Shape of Flocks of Birds. Verkkoaineisto. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3150374/>. Päivitetty 4.8.2011. Luettu 19.3.2018.

8 Rabin, Steve. 2002. AI Game Programming Wisdom. Charles River Media.

9 Walther, Anders. 2006. AI for real-time strategy games. Master’s thesis. IT-University of Copenhagen. Saatavilla: <http://www.itu.dk/image/edu/theses/pdf/AndersWalther06.pdf>.

10 Hurk, S. 2009. A Multi-Layered Flocking System For Crowd Simulation. Verkkoaineisto. <https://www.cs.auckland.ac.nz/research/gameai/dissertations/Hurk_BSc_09.pdf>. Luettu 26.3.2018.

11 Bertz, Matt. 2011. The Technology Behind The Elder Scrolls V: Skyrim. Verkkojulkaisu. <http://www.gameinformer.com/games/the_elder_scrolls_v_skyrim/b/xbox360/archive/2011/01/17/the-technology-behind-elder-scrolls-v-skyrim.aspx>. Päivitetty 17.1.2011. Luettu 26.3.2018.

12 Reynolds, Craig. 1986. Flocks, herds and schools: A distributed behavioral model. SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques. S. 7–15.

13 Reynolds, Craig. 2001. Boids. Verkkoaineisto <http://www.red3d.com/cwr/boids/>. Päivitetty 6.8.2001. Luettu 19.3.2018.

14 Half-Life Wiki. Verkkoaineisto. <http://half-life.wikia.com/wiki/Boid>. Luettu 8.4.2018.

15 Berg, J. ym. 2008. Interactive navigation of multiple agents in crowded environments. Verkkoaineisto. <http://rll.berkeley.edu/~sachin/papers/Berg-I3D2008.pdf>. Luettu 16.3.2018.

16 Joselli, Mark ym. 2012. A Flocking Boids Simulation and Optimization Structure for Mobile Multicore Architectures. Verkkoaineisto. <https://pdfs.semanticscholar.org/4183/faeef708a56b988742b5572fce9174caec7b.pdf>. Luettu 26.3.2018.

17 Reynolds, Craig. 2000. Interaction with Groups of Autonomous Characters. Verkkoaineisto. <http://www.red3d.com/cwr/papers/2000/pip.pdf>. Luettu 20.3.2018.

18 How Exynos 9810 sets a new standard for mobile processing power. 2018. Verkkoaineisto. Samsung. <http://www.samsung.com/semiconductor/minisite/exynos/newsroom/blog/how-exynos-9810-sets-a-new-standard-for-mobile-processing-power/>. Päivitetty 6.2.2018. Luettu 26.3. 2018.

19 Brown, Fraser. 2018. Premature Evaluation – Hunt: Showdown. Verkkojulkaisu. Rock, Paper, Shotgun. <https://www.rockpapershotgun.com/2018/03/06/hunt-showdown-early-access-review/>. Päivitetty 6.3.2018. Luettu 8.4.2018.

20 Badland. Google Play Store. <https://play.google.com/store/apps/details?id=com.frogmind.badland>. Luettu 22.3.2018.

21 Bour, David M. & Seemann, Glenn. AI for Game Developers. Verkkoaineisto. <https://www.cse.unr.edu/~sushil/class/381/notes/AIGDch04.pdf>. Luettu 23.3.2018.

22 Hunt: Showdown. 2018. Crytek. <https://www.huntshowdown.com/>. Luettu 26.3.2018.

BoidFlockingAI.cs

```
using System.Collections.Generic;
using UnityEngine;

public class BoidFlockingAI : MonoBehaviour {

    public GameObject boid;
    public GameObject boidParent;
    public int amountOfBoids = 20;

    private List<Transform> boids;
    private List<float> individualForceModifiers;

    [Header("Neighbor distance")]
    public float neighborDistance = 30f;

    [Header("Separation")]
    public float separationStrength = 0.15f;

    [Header("Alignment")]
    public float alignmentStrength = 0.2f;

    [Header("Cohesion")]
    public float cohesionStrength = 0.8f;

    [Header("Turbulence")]
    public float turbulenceStrengthMin = 1f;
    public float turbulenceStrengthMax = 3f;

    void Start ()
    {
        SpawnBoids(amountOfBoids);
        //Create separate personalities for each boid
        individualForceModifiers = new List<float>();
        for (int i = 0; i < amountOfBoids; i++)
        {
            individualForceModifiers.Add(Random.Range(turbulenceStrengthMin, tur-
bulenceStrengthMax));
        }
    }

    public void SpawnBoids(int amount)
    {
        if (boids != null)
            foreach (Transform t in boids)
                Destroy(t.gameObject);

        boids = new List<Transform>();
        for (int i = 0; i < amount; i++)
        {
            GameObject b = GameObject.Instantiate(boid, new Vector3(Ran-
dom.Range(-50, 50), Random.Range(30, 35), Random.Range(-50, 50)), Quater-
nion.identity, boidParent.transform);
            boids.Add(b.transform);
        }
    }
```

```
    public void GetUpdate(Transform b)
    {

        List<Transform> neighbors = new List<Transform>();
        foreach (Transform t in boids)
            if (t != b)
                if ((b.position - t.position).magnitude <= neighborDistance &&
!IsNeighborBehind(b, t))
                    neighbors.Add(t);

        if (neighbors.Count == 0)
            return;

        DoSeparation(b, neighbors);
        DoAlignment(b, neighbors);
        DoCohesion(b, neighbors);
    }

    public void DoSeparation(Transform b, List<Transform> neighbors)
    {
        Vector3 separationForce = Vector3.zero;
        foreach (Transform n in neighbors)
        {
            separationForce += (b.position - n.position) * 1 / (b.position -
n.position).magnitude;
        }
        separationForce *= separationStrength * Random.Range(individualForceMo-
difiers[b.GetSiblingIndex()], individualForceModifiers[b.GetSiblingIndex()]);
        b.GetComponent<Rigidbody>().AddForce(separationForce, ForceMode.Force);
    }

    public void DoAlignment(Transform b, List<Transform> neighbors)
    {

        Vector3 avgVelocity = Vector3.zero;
        foreach (Transform n in neighbors)
        {
            avgVelocity += n.GetComponent<Rigidbody>().velocity;
        }
        avgVelocity /= neighbors.Count;
        Vector3 alignmentForce = avgVelocity * alignmentStrength * Ran-
dom.Range(individualForceModifiers[b.GetSiblingIndex()], individualForceMo-
difiers[b.GetSiblingIndex()]);
        b.GetComponent<Rigidbody>().AddForce(alignmentForce, ForceMode.Force);
    }

    public void DoCohesion(Transform b, List<Transform> neighbors)
    {
        Vector3 avgPosition = Vector3.zero;
        foreach (Transform n in neighbors)
        {
            avgPosition += n.position;
        }
        avgPosition /= neighbors.Count;
        Vector3 cohesionForce = (avgPosition - b.position) * cohesionStrength *
Random.Range(individualForceModifiers[b.GetSiblingIndex()], individualForceMo-
difiers[b.GetSiblingIndex()]);
        b.GetComponent<Rigidbody>().AddForce(cohesionForce, ForceMode.Force);
    }
```

```
    public bool IsNeighborBehind(Transform t1, Transform t2)
    {
        Vector3 targetDir = transform.forward;
        float angle = Vector3.Angle(transform.forward, t1.position - t2.posi-
tion);
        //Don't count the neighbor if it is beyond 150 deg. (60 deg sector behind
the boid)
        if (angle > 150.0f)
            return true;
        return false;
    }

    public void AddBoid()
    {
        amountOfBoids++;
        GameObject b = Instantiate(boid, new Vector3(Random.Range(-50, 50), Ran-
dom.Range(50, 70), Random.Range(-50, 50)), Quaternion.identity, boidParent.trans-
form);
        individualForceModifiers.Add(Random.Range(turbulenceStrengthMin, turbu-
lenceStrengthMax));
        boids.Add(b.transform);
    }

    public void RemoveBoid()
    {
        if (boids.Count == 0)
            return;
        amountOfBoids--;
        Destroy(boids[amountOfBoids].gameObject);
        individualForceModifiers.RemoveAt(amountOfBoids);
        boids.RemoveAt(amountOfBoids);
    }
}
```

Ulkopuoliset paketit ja materiaali

Puut: Realistic Tree 9 [Rainbow Tree] (Rakshi Games), <https://assetstore.unity.com/packages/3d/vegetation/trees/realistic-tree-9-rainbow-tree-54622>.

Linnun 3D-malli, animaatio ja äänet: Living Birds (dinopunch), <https://assetstore.unity.com/packages/3d/characters/animals/living-birds-15649>.

Skybox: Wispy Skybox (Mundus Limited), <https://assetstore.unity.com/packages/2d/textures-materials/sky/wispy-skybox-21737>.

Askelten äänet: Footsteps (Snow and Grass) (MGWSoundDesign), <https://assetstore.unity.com/packages/audio/sound-fx/footstep-snow-and-grass-90678>.

Puistonpenkit ja valaisimet: Park Props Pack (Dimonati), <https://assetstore.unity.com/packages/3d/props/exterior/park-props-pack-49221>.

Aluskasvillisuus ja maastotekstuuri: Grass And Flowers Pack 1 (Vladislav Pochezhertsev), <https://assetstore.unity.com/packages/2d/textures-materials/grass-and-flowers-pack-1-17100>.

Tuulen äänileike: Light Breeze Field Sound FX (nathanolson), <https://www.youtube.com/watch?v=490z2QH6v5o>.

Dynaamiset pilvet: Simple Dynamic Clouds (Butterfly World), <https://assetstore.unity.com/packages/tools/particles-effects/bfw-simple-dynamic-clouds-85665>.