Mikko Partonen

Pesäpallopelin jatkokehitys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

8.5.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Mikko Partonen Pesäpallopelin jatkokehitys 39 sivua 8.5.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro Lehtori Antti Laiho

Tämän insinöörityön aihe on pesäpallopelin jatkokehitys, eli tarkennettuna sisäkentän tekoälyn toteuttaminen ja ulkokentän muokkaaminen ihmiselle pelattavaksi. Pelissä pystyi jo valmiiksi pelaamaan sisäkenttää ulkokentän tekoälyä vastaan. Työn tavoite oli tehdä myös ulkokenttä ihmiselle pelattavaksi ja tehdä tekoäly sisäkenttää pyörittämään.

Työn alussa selvitettiin olemassa olevia tekoälymenetelmiä erilaisissa peleissä. Näitä käytettiin pesäpallopelin tekoälyä suunniteltaessa ja toteuttaessa. Työssä päädyttiin toteuttamaan tekoäly äärellisenä tilakoneena, samoin kuin jo ennestään oleva ulkokentän tekoäly oli toteutettu. Ulkokentän muokkaaminen ihmiselle pelattavaksi sisälsi sisäkentän tekoälyn lisäksi ulkokenttäpelaajien hiirellä käskyttämisen ja kameran liikkeet.

Työn tuloksena pesäpallopelissä ihminen voi pelata myös ulkokenttää sisäkentän lisäksi. Ulkokenttäpelaajia voi käskyttää seuraamaan palloa tai valitsemaan kelle kopattu pallo heitetään. Koppaajat palaavat lähtöpisteilleen itsestään. Sisäkentän tekoäly sai tehtyä useita juoksuja suoritetuissa testipeleissä ihmistä vastaan. Kun tekoälyt laitettiin ottelemaan toisiaan vastaan, ulkokenttä sai selvästi enemmän paloja aikaiseksi kuin sisäkenttä juoksuja. Ihmisen on ainakin aluksi vaikeampaa hallita ulkokenttää, joten ihminen saa vähemmän paloja ulkokentän tekoälyyn verrattuna. Tavoiteltu haasteellisuus saavutettiin.

Jatkossa pesäpallopeliin voi kehittää pelimuodon, jossa ihminen ohjaa vaihdellen sisä- ja ulkokenttää, ja tehdä kunnolliset grafiikat. Pelin tekoälyjä voi myös parannella monimutkaisempia tekoälymenetelmiä käyttäen.

Avainsanat	pesäpallo, tekoäly, tietokonepeli, Unity, C#, tilakone

Author(s) Title Number of Pages Date	Mikko Partonen Further Development of Finnish Baseball Game 39 pages 8 May 2014
Degree	Bachelor of Engineering
Degree Program	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Miikka Mäki-Uuro, Senior Lecturer Antti Laiho, Lecturer

The aim of the study was to implement artificial intelligence to infield and edit outfield to be playable to human players in the Finnish baseball game. It was already possible in the game to play infield against outfield artificial intelligence. The goal of the study was to also make outfield playable to humans and make artificial intelligence to control infield players.

At the beginning of the study, existing artificial intelligence methods in different games were researched. This information was used in designing and implementing artificial intelligence in the baseball game. The artificial intelligence was chosen to be implemented as a finite state machine, as the artificial intelligence for outfield had been made. Modifying outfield for human play included, in addition to infield artificial intelligence, controlling outfield players with mouse and camera movements.

The outcome of the study is that in the baseball game, humans can now play outfield in addition to infield. Outfield players can be ordered to follow the ball or select to whom the ball will be thrown. Catchers return to their default locations automatically. Infield AI made several runs in test plays. When both AIs were put against each other, outfield AI scored much more burns than infield AI made runs. It is at least at the beginning harder for humans to control outfield, so human scores fewer burns compared to outfield AI. The intended level of difficulty was achieved.

In the future a game mode can be implemented in the baseball game, in which human playing switches between infield and outfield play, and proper graphics can be made. The artificial intelligences of the game can also be upgraded using more advanced artificial intelligence methods.

Keywords	baseball, artificial intelligence, computer game, Unity, C#, state machine

Sisällys

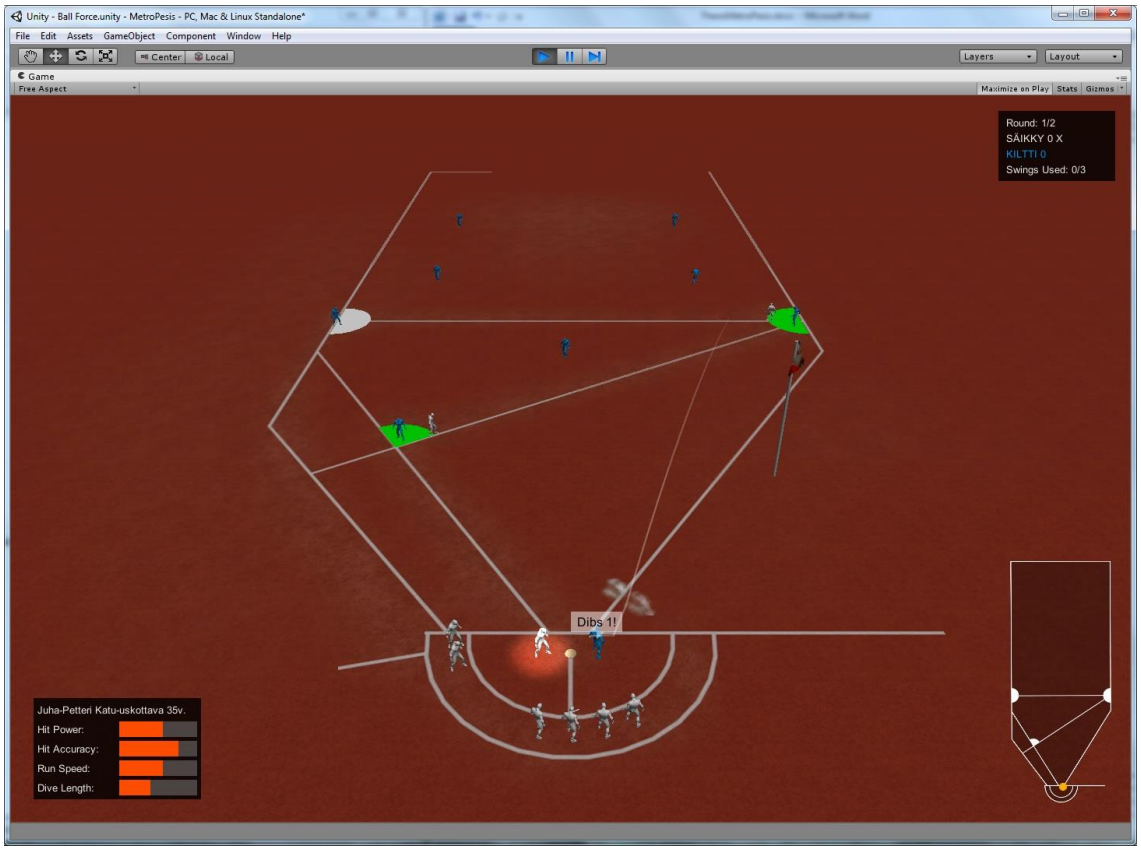
1 Johdanto

Aiheena oli pesäpallopelin jatkokehitys. Peliä alettiin kehittää ohjelmistotuotantoprojektina, jossa olin osana ryhmää sitä tekemässä. Pelille annettiin nimeksi MetroPesis. Pelin sisäkenttäpelaaminen saatiin tehtyä, joskin jäljelle jäi erilaisia virheitä. Ulkokentälle tehtiin tekoäly sitä pyörittämään. Tarkoitus oli tehdä ulkokenttä ihmiselle pelattavaksi ja kirjoittaa tekoäly sisäkenttää pelaamaan. Valitsin tämän aiheen, koska pelin kehittäminen ohjelmistotuotantoprojektissa oli mielenkiintoista.

Työ jakautuu kolmeen osaan, jotka ovat tekoälyjen taustatutkimus, sisäkentän tekoälyn toteutus ja ulkokentän pelimekaniikan toteutus. Työn alussa tutkittiin, millaisia tekoälytekniikoita on olemassa ja miten tekoälyjä on toteutettu erilaisissa peleissä. Tämän pohjalta päätettiin, miten sisäkentän tekoäly toteutetaan. Tekoälymenetelmien valitsemisen jälkeen tehtiin tekoäly sisäkentälle. Tavoitteena oli tehdä tekoälystä niin haasteellinen, että sisäkenttä saa tehtyä useita juoksuja yhden kierroksen (jokainen lyöjä juoksee ainakin kerran) aikana. Sisäkentän tekoälyn tekemisen jälkeen muokattiin ulkokenttä pelattavaksi ihmispelaajalle.

Pesäpallopeli on tehty Unity-pelimoottoria käyttäen. Kaikki ohjelmakoodit on kirjoitettu C#-ohjelmointikielellä. Toinen vaihtoehto Unityn kanssa käytettäväksi olisi JavaScript. Pelaajilla ei vielä ole tekstuureja, vaan pelkästään mallit ja osa animaatioista. Ohjelmistotuotantoprojektissa oli yhteensä kuusi henkilöä mukana syksyllä 2013.

Pesäpallo on joukkueurheilupeli jossa sisäkenttäpelaajien tavoitteena on juosta kentällä olevat pesät läpi. Ulkokenttäpelaajien tavoitteena on estää sisäkenttäpelaajien juoksut polttamalla tai haavoittamalla juoksijoita. Palon tai haavan sattuessa juoksija joutuu palamaan kotipesälle, eikä sitä lasketa juoksuksi. Kuvassa 1 näkyy ohjelmistotuotantoprojektissa kehitetty sisäkentän pelaaminen. Kahdella pesällä on juoksija, ja lukkarille heitettiin pallo seuraavaa lyöntiä varten. Suurin osa pesäpallon säännöistä toteutettiin. Ohjelmistotuotantoprojektissa tehtiin peliin kaksi pelimuotoa, joista ensimmäinen on loputon sisäkentän pelaaminen ja toinen kahden pelaajan peli. Kahden pelaajan pelissä pelataan myös loputtomasti sisäkenttää, mutta joukkue vaihtuu pesäpallon sääntöjen mukaisesti sisä- ja ulkokentän välillä.



Kuva 1. Ohjelmistotuotantoprojektissa saatiin pesäpallon sisäkenttä pelattavaksi.

Luku kaksi käsittelee tekoälytekniikoita. Luku kolme käsittelee tekoälyjä peleissä ja miten tekoälytekniikoita sovelletaan niissä. Luku neljä käsittelee sisäkentän toteutusta suunnittelusta loppuun saakka. Ensin siinä käsitellään, mitä vaatimuksia olen asettanut pesäpallolle ja mitä ominaisuuksia pelissä voisi myös olla. Tämän jälkeen tarkastellaan erilaisia vaihtoehtoja, ja millaiset säikeet toteutetaan. Sen jälkeen selostetaan, miten tekoäly päätettiin tehdä. Luvun lopussa kerrotaan, miten tekoäly käytännössä toteutettiin C#-koodina. Luku viisi käsittelee ulkokentän pelimekaniikan toteutusta. Ensin siinä kerrotaan pelaajien ohjaamisesta ja lopuksi kameran toteuttamisesta. Luku kuusi kertoo pelin testauksesta ja tulosten arvioinnista. Lopuksi luku seitsemän on yhteenveto koko työstä.

2 Tekoälytekniikoita

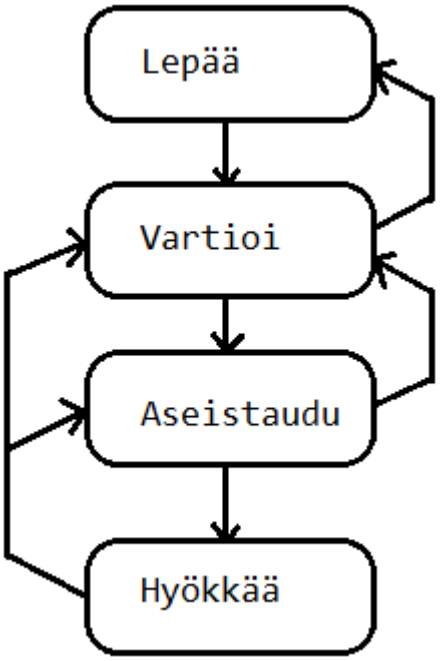
Pelitekoälyjen kehittely alkoi tietokoneiden alkuajoilta 1960-luvulta. 1970-luvulla julkaistiin useita pelikonsoleita. Pitkän aikaa videopelien tekoälyt olivat tosi yksinkertaista, koska pelit olivat suhteellisen yksinkertaisia ja yleensä vastapelaajana oli toinen ihmi-

nen. Peruspeleissä oli yksinkertainen hakutaulukko, jossa oli sopiva toiminta nykyiselle tilanteelle. Tosiaikainen pelaaminen on ollut suuri taakka tekoälyille videopeleissä tietokoneiden laskentatehosta johtuen. 1990-luvulla pelien tekoälyt kehittyivät valtavasti. Pääosin tietokonepelit kehittyivät konsolipelien sijasta, koska henkilökohtaisissa tietokoneissa oli paljon enemmän laskentatehoa. [1, s. 3–5.]

Koska pesäpallopelissä ei ollut mitään polunetsintää, eikä sille toistaiseksi ole ollut erityistä tarvetta, rajattiin polunetsintä pois taustatutkimuksesta. Pelikentällä ainoat kulkemista haittaavat esteet ovat toiset pelaajat. Pelissä oli jo koodia, joka pistää hahmot väistämään toisiaan törmäyksen sattuessa. Kunnollisen väistämisen toteuttaminen jäi kuitenkin kesken, joten kytkettiin pelaajien toisiinsa törmäykset kokonaan pois päältä niin, että pelaajat kävelevät toistensa läpi törmätessään. Tulevaisuudessa peliin voisi kehitellä väistämistekoälyn, joka huolehtisi siitä, ettei toisiin pelaajiin edes törmätä.

2.1 Äärelliset tilakoneet

Äärelliset tilakoneet ovat yksinkertaisimpia, tehokkaimpia ja eniten käytettyjä menetelmiä tekoälyn ohjelmoinnissa. Esimerkiksi ritari voi olla aseistautumassa, vartioimassa, hyökkäämässä tai lepäämässä taistelun jälkeen kuvan 2 mukaisesti. Kyläläinen voi olla hakkaamassa puuta, rakentamassa taloa tai suojautumassa hyökkäyksiltä. Tiloista riippuen peliobjektit vastaavat erikseen äärelliseen määrään ulkoisia ärsykkeitä. Tämä menetelmä mahdollistaa peliobjektien käyttäytymisen helpon jakamisen pienempiin osiin joita on helppo testata ja laajentaa. Jokainen tila sisältää koodia objektin alustamiseen ja epäalustamiseen, koodia suoritettavaksi jokaisella kehyksellä esimerkiksi animaatioita varten, ja koodia käsittelemään ja tulkitsemaan viestejä ympäristöstä. [2, s. 8–9.]



Kuva 2. Kaaviossa näkyy ritarin tilat ja niiden siirtymät toisiinsa.

Ennen oliosuuntautunutta ohjelmointia tilakoneet toteutettiin C-kielellä makroina. Nykyään tilakoneita toteutetaan suunnittelumalleilla. Voidaan määritellä yleinen tilaluokka, joka peritään kaikissa tiloja hyödyntävissä olioissa. Tässä luokassa voi olla funktiot tilaan saapumiseen, tilasta poistumiseen, tilassa olemiseen ja viestin vastaanotolle. [2, s. 9.]

Äärellisiä tilakoneita käytetään kaikkialla. Yksinkertaisesti niiden idea on käydä läpi erilaisia vaihtoehtoja ja suorittaa niitä käyttäen jos-lausekkeita erilaisten olosuhteiden ja vaatimusten tarkistukseen. [3, s. 26.]

Äärellinen tilakone on yhdessä useammasta ennalta määrätyssä tilassa. Se voi myös määrittää ehdot tilasta toiseen siirtymiselle. Äärelliset tilakoneet ovat olleet käytössä tietokonepeliohjelmoinnin alkuajoilta, mutta ovat edelleen hyödyllisiä ja yleisesti käytössä. [3, s. 228.]

Tila on vallassa, kunnes tilanvaihtoehdot täyttyvät ja tilakone siirtyy toiseen tilaan. Mikäli tiloja on paljon, on helpompaa ohjelmoida modulaarinen tilakone. Siinä tila ei tiedä muista tiloista eikä tilakoneesta. Tällaisella tilalla on viisi eri toimintoa: sisääntulo, poistuminen, pysyminen, alustus ja tilanvaihdontarkistus. Tilakoneluokka sisältää kaikki siihen liittyvät tilat. [4, s. 261–262, 267.]

2.2 Sumea logiikka

Pelinkehittäjät käyttävät usein sumeaa logiikkaa sumeissa tilakoneissa, jotta tulokset olisivat vähemmän ennustettavissa ja vähentäisivät taakkaa käydä valtavaa määrää jos-lausekkeita läpi. Sumea logiikka esittää ongelmia tietokoneelle samaan tapaan kuin ihminen ratkaisee niitä. Ihmiset tosi usein analysoivat tilanteita ja ratkaisevat ongelmia epätäsmällisesti. Kaikkia tosiasioita ei ole saatavilla, tai ne saattavat olla epäselviä. Esimerkiksi määriteltäessä onko joku ihminen pitkä vai tosi pitkä, ihmisellä ei ole täsmällistä pituuslukemaa rajan määrittämiseksi. Sumean logiikan tarkoitus on tehdä sama asia tietokoneella. Eli täsmällisten raja-arvojen sijasta käytetään epämääräisiä vyöhykkeitä. Erilaiset arvovyöhykkeet menevät jonkin verran toistensa päälle, kuten tässä tapauksessa pitkä ja tosi pitkä ihminen. [3, s. 26, 257–258.]

Sumeassa logiikassa jokin voi olla ehdottomasti totta tai epätotta tai sitten jotain siltä väliltä. Tämän määrittelemisessä käytetään omia funktioita. Funktion on tarkoitus tuottaa totuus esimerkiksi asteikolla nollasta yhteen liukulukuna ilmoitettuna. Tiettyyn raja-arvoon asti funktio tuottaa arvon 0, jonka jälkeen lukema kasvaa suoraan tai käyränä toiseen raja-arvoon, minkä jälkeen funktio tuottaa arvon 1. Funktion tulos ilmaisee, kuinka totta sisään annettu arvo on. Jotkut sumean logiikan harjoittajat suosittelevat seitsemän sumean joukon määrittämistä jokaisen sisääntuloarvon määrittelemiseksi. Eli totuusarvojen määrittelevä funktio kertoo seitsemällä liukuluvulla, kuinka totta funktiolle annettu arvo on. Näiden sekavien joukkojen pitäisi mennä toistensa päälle neljänneksen. [3, s. 258, 264, 267.]

Sumean logiikan tilakone voi olla useassa tilassa yhtä aikaa. Jokaisella tilalla on taso, joka määrittelee, kuinka suuressa käytössä tila on. Sumeat tilakoneet ovat hyödyllisiä vain tapauksissa joissa voi olla monta tilaa kerralla. [4, s. 303–304, 306.]

2.3 Viesteihin perustuvat järjestelmät

Viestijärjestelmää käytetään tilakoneita enemmän. Niitä voisi kutsua myös tapahtumajärjestelmiksi. Sen sijaan että pelin osat kyselevät jatkuvasti pelin muiden osien tietoja, tapahtuman sattuessa kyseinen pelin osa lähettää viestejä muille osille. Pelin osien välinen viestintä voidaan keskittää yhteen paikkaan, niin että pelin osien ei tarvitse

päästä suoraan käsiksi tarvitsemiinsa luokkiin, vaan viestijärjestelmän kautta. [4, s. 335–336.]

2.4 Tekoälykieli

Yksi keino toteuttaa tekoäly on käyttää tai luoda jokin ohjelmointikieli tekoälyä varten. Tekoälyä siis ohjataan varsinaisen peliohjelmakoodin ulkopuolelta tiedostoista. Tiedostot tulkitaan joko lennosta tai ennen pelin alkamista käännetään konekieleksi. Tällöin pelin tekoälyn toimintaa on helppo muokata myös pelaajien toimesta. On hyvä huomioida, kenen on tarkoitus lukea tekoälytiedostoja tekoälykielen monimutkaisuuden määrittelemiseksi. On yleistä ja hyödyllistä määritellä kaikki tekoälyvastustajien perusarvot jonkinlaisella tiedostolla. Tämä tekee helpoksi tekoälyvastustajien muokkaamisen pelin kehittämisen aikana. Jos kaikki tärkeät tiedot ovat kovakoodattuna ohjelmaan, jokaisen pienenkin muutoksen tekeminen vaatii uudelleenkääntämisen. Tiedostojen lukemisessa on tärkeää huomioida virheidentarkistus varsinkin, jos pelaajille annetaan mahdollisuus muokata tiedostoja. Koskaan ei voi luottaa siihen, että tiedostot on kirjoitettu oikein. [3, s. 205–207, 212; 4, s. 363.]

2.5 Geneettiset algoritmit

Geneettisiä algoritmeja voi verrata eliöiden DNA:n muuttumiseen ajan kuluessa. Parhaiten ympäristöönsä sopeutuvat eliöt selviävät, ja heikot kuolevat pois.

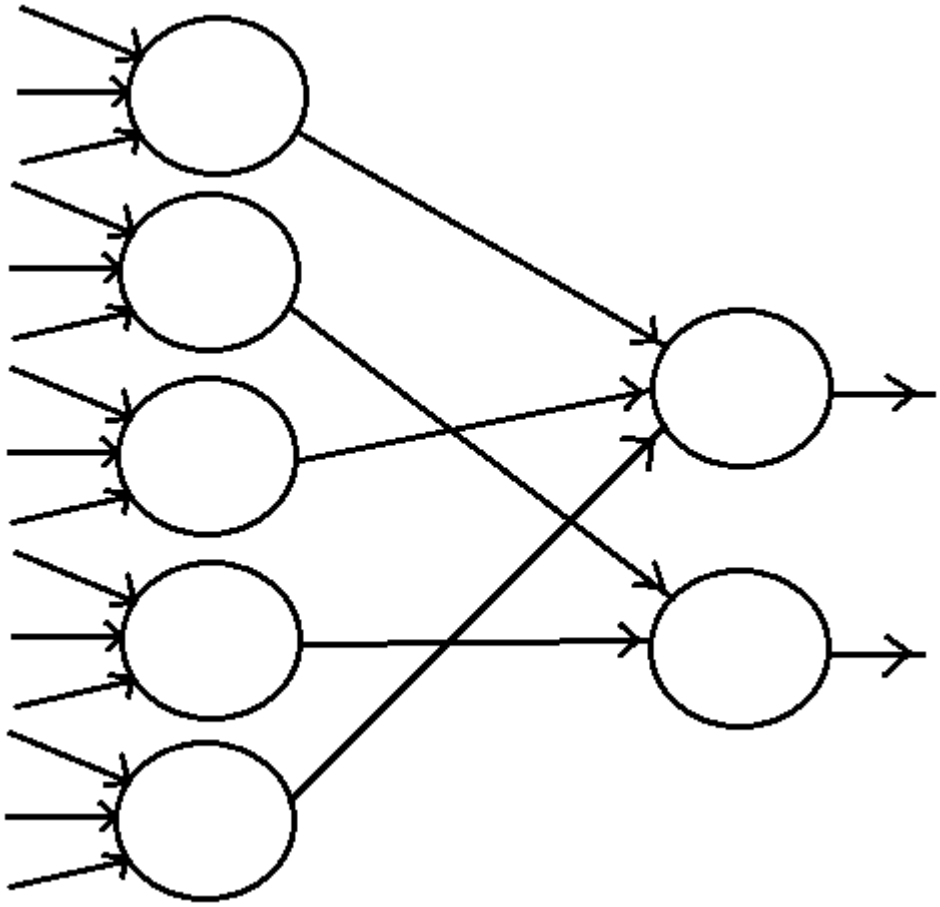
Geneettisten algoritmien toiminta matkii eliöiden DNA:n muuttumista ajan kuluessa. Ensin selvitetään keino ongelman mahdollisen ratkaisun koodaamiseksi digitaaliseen kromosomiin. Sitten luodaan alkupopulaatio satunnaisia kromosomeja ja kehitetään niitä ajan kanssa kasvattaen sopeutuvimpia yksilöitä ja lisäten mutaatioita sinne tänne. Onnen käydessä usean sukupolven jälkeen geneettinen algoritmi antaa ratkaisun. Geneettiset algoritmit eivät takaa ratkaisua tai parasta ratkaisua, mutta oikein käytettyinä pystyy yleensä kirjoittamaan hyvin suoriutuvan geneettisen algoritmin. Parasta geneettisissä algoritmeissa on, ettei tarvitse tietää ratkaisua ongelmaan, vaan ainoastaan miten koodata ratkaisu niin, että geneettinen algoritmi voi sitä hyödyntää. [5, s. 98–99.]

Tavallisesti kromosomi koodataan sarjana binääribittejä. Alussa luodaan kromosomipopulaatio, jonka jokaisen kromosomin bitit on asetettu satunnaisesti. Kromosomin pituus on yleensä vakio koko populaatiolle. Tärkeää on, että kromosomit koodataan niin, että niistä voidaan muodostaa ratkaisu ongelmalle. Jokainen kromosomi esittää mahdollista ratkaisua. [5, s. 99.]

Ensin jokaista kromosomia testataan ja niille annetaan sopivuusarvo. Sitten valitaan kaksi kromosomia nykyisestä populaatiosta. Kromosomien bitit vaihdetaan keskenään satunnaisesta kohdasta alkaen eteenpäin, mikäli satunnaislukugeneraattori tuottaa valittua vaihtumisnopeutta pienemmän arvon. Tyypillisesti 70 %:n todennäköisyys valitaan bittien keskenään vaihtamiselle. Seuraavaksi muutetaan satunnaisesti bittejä määritellyn mutaationopeuden mukaisesti. Hyvä arvo on tuhannesosan todennäköisyys. Valitaan taas satunnaisesti kaksi kromosomia koko populaatiosta niiden sopivuusarvojen perusteella, vaihdetaan niiden bittejä keskenään satunnaisesta kohdasta eteenpäin ja luodaan niistä uudet mutatoituneet kromosomit, kunnes kokonainen uusi populaatio on luotu. Hyvä sääntö populaation koolle on olla kaksi kertaa kromosomin (bitti) pituus. [5, s. 99, 112.]

2.6 Neuroverkot

Neuroverkot matkivat aivojen rakennetta. Tekoneuroverkot muistuttavat aivoja siinä suhteessa, että ne koostuvat tekohermoista. Tekohermojen määrä vaihtelee kymmenistä moniin tuhansiin riippuen käyttötarpeesta. Tekohermoon tulee useita liukulukuja syötteenä, jotka kerrotaan niille asetetuilla etumerkillisillä painokertoimilla. Tekohermossa syötteet lasketaan yhteen. Jos tulos on tiettyä rajaa korkeampi, tekohermo tulostaa ykkösen, muutoin nollan. Kuva 3 havainnollistaa neuroverkon rakennetta. [5, s. 238–239.]



Kuva 3. Kaavio esittää neuroverkkoa. Vasemmassa reunassa on neuroverkkoon tulevat syötteet ja oikeassa reunassa on kaksi neuroverkosta tulevaa totuusarvoa.

Kuten biologiset hermot, tekohermot liittyvät myös toisiin tekohermoihin. Tavallisesti tekohermot laitetaan kerroksiin, niin että kerros tulostaa syötteitä seuraavalle. Yleensä kaksi kerrosta tekohermoja riittää. Tekohermot ovat hyviä hahmontunnistamisessa. Neuroverkon luomisen jälkeen se pitää opettaa tunnistamaan tietty kuvio. Yksi keino tekoneuroverkon opettamiseen on antaa sille satunnaiset syötekertoimet ja muokata niitä sen mukaan, mitä erilaiset kuviot tuottavat lopputulokseksi, kunnes tekoneuroverkko oppii tunnistamaan halutun kuvion. Helppo tapa luoda hyviä neuroverkkoja on kehittää niitä geneettisillä algoritmeilla. [5, s. 242–243, 245.]

3 Tekoäly peleissä

3.1 Ammuntapelit

Ensimmäisen ja kolmannen henkilön näkökulman ammuntapelit ovat olleet rahallisesti menestyneimpiä videopelien alkuajoilta asti. Ammuntapeleissä pelaajat yleensä liikkuvat jalkaisin, ja kamera on kiinteästi sidottuna ohjattavaan pelihahmoon. Viholliset ovat yleensä pelaajan kaltaisia hahmoja, joita on suhteellisen vähän ja joita tekoäly ohjaa. Tärkeimmät asiat tekoälyn kannalta ovat liikkuminen, ammunta, päätöksenteko, havaintokyky, polunetsintä ja taktinen tekoäly. [6, s. 814.]

Ammuntapeleissä tekoäly on yleensä toteutettu kerroksellisena rakenteena. Pohjalla olevat kerrokset huolehtivat perustehtävistä, kuten parhaan reitin löytämisestä kohteeseen tai sopivien animaatioiden pyörittämisestä oikealla nopeudella. Korkeammat tasot huolehtivat taktisesta päättelystä ja tekoälyllä ohjailtavan objektin käyttäytymisestä, kuten pitäisikö objektin vartioida aluetta, ryhtyä kamppailuun vai juosta karttaa ympäri vastustajia etsien. Kun korkeampi tekoälyn kerros on päättänyt sopivan tehtävän valitsemisesta tilanteeseen, se antaa alempien kerroksien tehtäväksi valita parhaat keinot tehtävän täyttämiseen. Jos esimerkiksi ylempi kerros on päättänyt, että on aika hyökätä, alemmat kerrokset päättävät, hiivitäänkö, viritetäänkö ansa vai juostaanko päin näköä. [2, s. 2.]

Monet pelit, jotka käyttävät ohjelmointikieliä pelihahmon ohjaamiseen, käyttävät samoja ohjaimia tekoälylle ja ihmispelaajille. Tällöin pelihahmon ohjaaja on helppo vaihtaa ihmisestä tekoälyyn tai toisinpäin, koska molemmat ohjaavat pelihahmoa samalla tavalla. Päätöksenteko on yleensä tehty äärellisillä tilakoneilla ja enenevässä määrin käyttäytymispuilla. Tosi yleinen tapa päätöksentekoon on kehittää tietokoneen ohjaamille pelihahmoille ohjelmointikieli. Tällöin pelin kehittäjien lisäksi muut voivat muokata tekoälyä helposti. [6, s. 815–816.]

3.2 Ajopelit

Ajaminen on erikoistunein tekoälytehtävä kehittäjille. Toisin kuin muissa peleissä kaikki oleelliset tekoälytehtävät keskittyvät liikkumiseen. Yksinkertaisin tapa toteuttaa autojen liike on tehdä kartoille ajolinjoja tekoälyä varten, joita tekoälyn ohjaamat autot seuraa-

vat tarkasti. Törmäysten sattuessa käytetään yksinkertaista auton ohjauskoodia auton saamiseksi takaisin ajolinjalle. Nykyisin sen sijaan laitetaan tekoäly ajamaan autoa samaan tapaan kuin ihmispelaaja sitä ajaa. Silti ajolinjojen käyttö on yleistä. Tekoäly koettaa seurata ajolinjaa, muttei kulje sitä pitkin kuin juna. Toisenlainen tapa on jäniksenjahtaaminen, jossa tekoäly ohjaa autoa edessäpäin olevaa kohdetta kohti, jota siirretään jatkuvasti. Tämä mahdollistaa luonnollisemmat liikkeet mutkissa. [6, s. 820–822.]

3.3 Tosiaikaiset strategiapelit

Tosiaikaisissa strategiapeleissä voidaan erotella useita eri tekoälyn osia ja kerrosmaista rakennetta. Yksi perusosa on tehokas polunetsintäjärjestelmä. Tekoälyn korkeammilla tasoilla on osia jotka vastaavat taloudesta, kehityksestä ja kartan analysoinnista. Matalammilla tekoälyn tasoilla ohjataan pienempien joukkojen toimintaa ja alimmillaan vain yksikön toimintaa. [2, s. 3.]

Tosiaikaisten strategiapelien tärkeimmät asiat tekoälyn kannalta ovat polunetsintä, ryhmäliikunta, taktinen ja strateginen tekoäly ja päätöksenteko. Päätöksenteko tapahtuu monella tasolla. Yksittäiset pelihahmot tekevät itsenäisiä päätöksiä, sotilaiden joukot yhteisiä päätöksiä ja koko armeija omia. Ylemmän tason tekoälyn päätökset annetaan alemmille tasoille toteutettaviksi. Taloudellinen ja sotilaallinen tekoäly voivat olla enimmäkseen itsenäisiä. Yleensä suurin osa päätöksenteosta koostuu yksinkertaisista tilakoneista, päätöspuista ja sääntöihin perustuvista järjestelmistä. [6, s. 823–824, 826–827; 4, s. 105–106.]

3.4 Urheilupelit

Useimmissa urheilupeleissä käytetään laajamittaista huijausta. Esimerkiksi kilpa-ajopeleissä tekoälyä varten kartan geometriasta vain ajorataan kuuluvat polygonit otetaan huomioon. Kaksi käyrää merkataan karttaan, ensimmäinen merkitsemään parasta ajoreittiä ja toinen kilpailijoiden ohittamista merkitsevä reitti. Tämäntyyppisissä peleissä tärkeimpiä tekoälyn ominaisuuksia ovat tiellä olevien esteiden havaitseminen ja kiertäminen sekä tiivis yhteistyö fysiikkamoduulin kanssa. Fysiikkamoduuli voi kertoa tietoa esimerkiksi renkaiden pitämisestä ja siitä, pitäisikö tekoälyn reagoida siihen jotenkin

pitääkseen ajoneuvon ohjauksessa. Samantapaista huijaamista on muissakin urheilupeleissä. Monessa tapauksessa tietokoneen ohjaaman pelaajan koko käyttäytyminen on päätetty jo ennen kuin pelaajan vuoro alkaa. [2, s. 4–5.]

Joukkueisiin perustuvissa urheilupeleissä yksittäisten pelihahmojen on tärkeää reagoida tilanteisiin huomioiden joukkueen muut jäsenet. Urheilupeleissä on usein monitasoinen tekoäly. Korkean tason tekoäly tekee strategisia päätöksiä. Alemmalla tasolla voi olla pelihahmoja sopivassa laumassa liikuttava tekoäly. Alimmalla tasolla on yksittäisiä pelihahmoja ohjaava tekoäly. [6, s. 827.]

Monissa urheilupeleissä on jonkinlaisia liikkuvia palloja tai vastaavia, joiden liikerataa tekoälyn on ennakoitava. Monimutkaisten pallojen liikkeiden peleissä voi olla tarvetta ennustaa pallon liikerataa ajamalla fysiikkamoottoria. Tekoäly siis pelaa peliä eteenpäin ihmisiltä näkymättömissä. [6, s. 827.]

Urheilupelejä ostavat ihmiset, jotka ovat joko pelanneet oikeassa elämässä paljon tai muuten vain tietävät pelin perusteellisesti. Pelinkehittäjillä on iso työ saada peli ja tekoäly toimimaan niin, että pelaaminen vaikuttaa aidolta. Ilman joukkuetason tekoälyä tekoälyjoukkueen pelaaminen voi vaikuttaa satunnaiselta tai päämäärättömältä. Pelaajatason tekoäly koskee vain yksittäistä pelaajaa. Mikäli jokin toimenpide vaatii useamman pelaajan yhteistyötä, tehtävä annetaan joukkuetason tekoälylle. On tärkeää saada tekoälypelaajien taitotaso vastaamaan vaihtelevantasoisia ihmispelaajia. Äärelliset ja sumeat tilakoneet ovat laajasti käytettyjä urheilupeleissä. [4, s. 172–174, 177.]

3.5 Roolipelit

Koska roolipelien läpipelaaminen kestää paljon kauemmin kuin muiden pelityyppien, ihmisen on helpompi huomata tekoälyssä olevia virheitä ja saman toistoa. Tästä syystä roolipelien tekoälyihin panostetaan enemmän. Käytetyimmät tekoälymenetelmät ovat ohjelmointikielen käyttö, äärelliset tilakoneet ja viestintäjärjestelmät. [4, s. 69–91.]

4 Sisäkentän tekoälyn toteutus

4.1 Vaatimukset

Jokaisen sisäkenttäpelaajan vaiheisiin kuuluu jonottaminen, lyöminen ja juokseminen. Tekoäly voisi myös päättää ennen joukkueen vaihtumista ja pelin alussa, missä järjestyksessä sisäkentän pelaajat laitetaan lyömään. Se jätettiin satunnaiseksi, niin kuin se oli jo pelissä toteutettuna.

Jonotustilassa tarvitsee vain tarkkailla, milloin pelaajan lyöntivuoro alkaa. Jonottavat pelaajat voisivat pelitilanteesta riippuen sanoa satunnaisia huudahduksia.

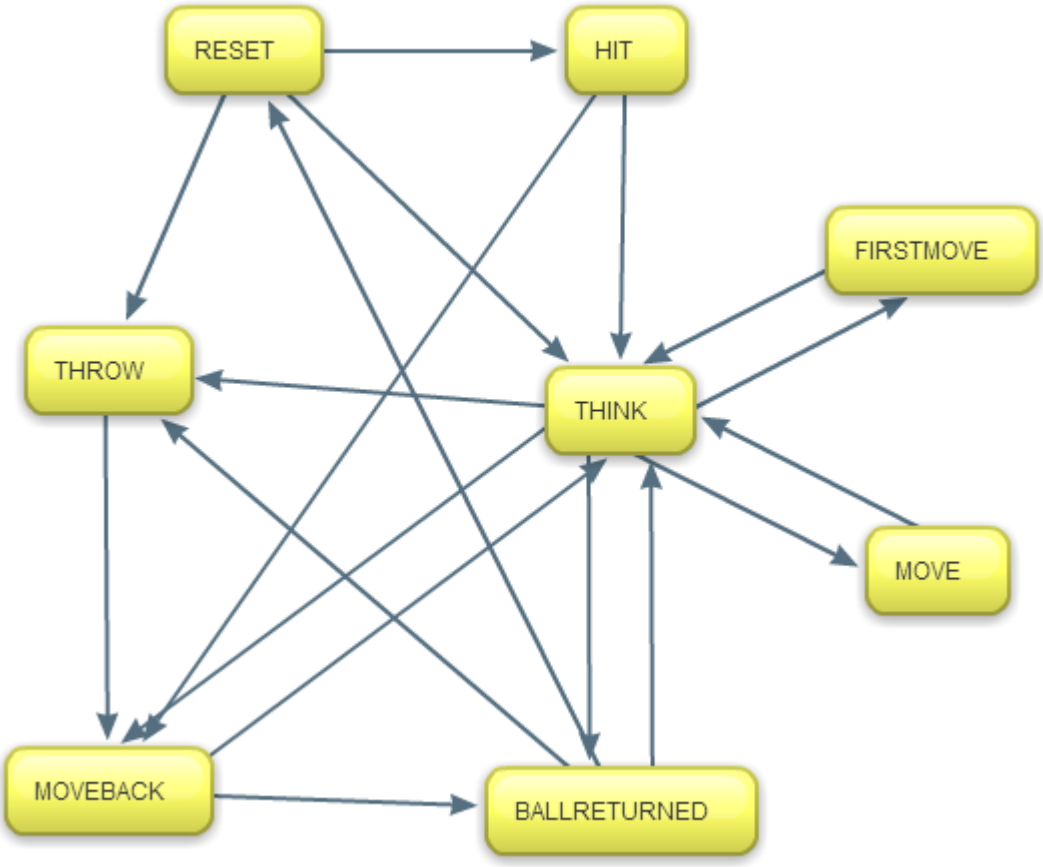
Lyöntivaiheessa pitää määritellä, kuinka kovaa lyödään, minne päin lyödään ja minkälainen lyönti lyödään. Lyöntivaihe tehtiin aluksi ilman mitään kummempaa älykkyyttä, niin että lyöjä yritti satunnaisella tarkkuudella lyödä maan kautta kimpoavan pallon keskelle kenttää. Lyöntiin voisi huomioida tuulen vaikutuksen, ulkokenttäpelaajien koppaustarkkuuden, heittovoiman ja heittotarkkuuden. Ulkokenttäpelaajien tietojen perusteella voisi arvioida minne päin kenttää kannattaa lyödä, jotta ulkokenttäpelaajilta kestäisi mahdollisimman kauan saada pallo takaisin kotipesälle. Vasta lyönnin tapahduttua on tarpeen miettiä, kannattaako lyöjän juosta.

Juoksuvaiheessa olevat pelaajat tarkkailevat, kannattaako juosta seuraavalle pesälle. Tässä pitää ottaa huomioon pallon etäisyys seuraavasta pesästä. Tekoälyä saisi hienosteltua analysoimalla lähistön ulkokenttäpelaajien tietoja. On tilanteita, jolloin on pakko juosta, ja niitä pitää tarkkailla juoksuvaiheessa. Juoksuvaiheessa olevan pelaajan ei tarvitse tarkkailla pelikentän tilannetta seuraavaa pesää pidemmälle. Heti kun juoksija pääsee seuraavalle pesälle, juoksemisen jatkaminen sitä seuraavalle pesälle alkaa. Siinä vaiheessa kun juoksija on juoksemassa, voi käydä niin, että kannattaakin palata takaisin lähtöpesälle. Pallo esimerkiksi saapuu yllättävän nopeasti seuraavaa pesää kohti. Tällaisessa tilanteessa pitäisi arvioida mahdollisuuksia juosta edelliselle pesälle.

Pelaajan lähestyessä pesää loppuvaiheessa pitää alkaa miettimään, kannattaako sukeltaa seuraavaa pesää kohti, jotta pääsisi hieman nopeammin. Tässä tarvitsee ottaa huomioon vain pallon nopeus ja se, kuinka lähellä pallo on pesävahtia.

4.2 Vaihtoehtojen tarkastelua

Pesäpallopelissä oli ulkokentän tekoäly toteutettu äärellisenä tilakoneena kuvan 4 mukaisesti. Tilat olivat string-tyyppisiä, joten ne muutettiin luetelluksi tyypiksi, jotta ulkokentän tekoälyä korjaillessa ja kehitellessä tulisi vähemmän virheitä. Päätettiin tehdä sisäkentän tekoäly jonkinlaisena tilakoneena myös. Yksittäisellä sisäkentän pelaajalla on tiloina jonotus, lyönti, juokseminen ja pesällä odottaminen.



Kuva 4. Kaavio esittää ohjelmistotuotantoprojektissa tehdyn ulkokentän tekoälyn tiloja siirtymisineen.

Sumea logiikka vaikutti mielenkiintoiselta tavalta saada tekoälyn toiminta ihmismäisemmäksi. Sen käyttämisen sijaan tyydyttiin vain satunnaislukugeneraattorilla luotaviin rajavyöhykkeisiin. Esimerkiksi lyöjä voi jäädä lyömään, jos hänen lyöntivoimansa on enemmän kuin satunnaislukugeneraattorilla tuotettu raja väliltä 70 ja 80. Tämä raja arvotaan uudestaan jokaisen lyönnin jälkeen. Koska sumea logiikka sopii tilanteisiin joissa voidaan olla useissa tiloissa yhtä aikaa, se ei sovellu pesäpallon yksinkertaisiin

tiloihin, kuten jonottaminen, lyönti, juoksu ja pesällä oleminen. Vain yhdessä tällaisista tiloista on järkevää olla kerrallaan.

Pesäpallopelin pelaajien arvot arvotaan satunnaisesti joukkueiden luonnin yhteydessä. Koska joukkueiden pelaajien arvojen on tarkoitus olla satunnaisia, ei ollut hirveästi järkeä laittaa arvoja luettavaksi erillisestä tekstitiedostosta.

Pesäpallon eri osat ovat sotkeutuneet toisiinsa aika tiiviisti. Jonkinlaisen viestijärjestelmän toteuttaminen voisi helpottaa pelin jatkokehitystä, koska eri luokat keskustelisivat keskitetysti toistensa kanssa.

Geneettistä tekoälyalgoritmia houkutti käyttää pesäpallossa. Voisi esimerkiksi tehdä niin, että digitaalinen kromosomi sisältää lyöntivoiman ja lyöntikulmat. Toisaalta tekoäly olisi silloin erittäin tyhmä pelin alkaessa, jolloin olisi syytä kokeilemalla ensin selvittää hyvät digitaaliset kromosomit ja pelin edetessä tekoäly kehittyisi jatkuvasti paremmaksi lyöjäksi. Lyöntikromosomit voisivat saada sopivuusarvon sen perusteella, montako onnistunutta kahden pesän välistä juoksua tapahtuu lyönnin jälkeen ennen seuraavaa lyöntiä. Geneettisen tekoälyn käyttäminen vaikutti aika työläälle toteuttaa, joten se jätettiin pesäpallopeliä myöhemmin jatkokehittäville pohdittavaksi.

Pesäpallossa voisi käyttää neuroverkkoa tunnistamaan, minkälaisessa pelaajien ja pallon sijoittumisessa kentällä on hyvä juosta. Työmääräarvio suhteessa tulokseen oli melko suuri neuroverkon toteuttamiseksi.

Ei koettu tarpeelliseksi käyttää tai kehittää mitään ohjelmointikieltä tekoälyä varten. Pesäpallo on niin yksinkertainen peli, että suurin hyöty erillisen ohjelmointikielen käytöstä tekoälyssä olisi mahdollistaa ulkopuolisille tekoälyn muokkaaminen.

Ajateltiin tehdä joko yksi säie kaikkien sisäkentän pelaajien liikuttamiseen tai identtinen tekoälysäie jokaiselle sisäkenttäpelaajalle. Tarkasteltiin myös jonkinlaisen yhdistelmän mahdollisuutta, jossa olisi erikseen tekoälysäie koko sisäkentälle ja sen lisäksi jokaista sisäkenttäpelaajaa ohjaisi oma säikeensä.

Yhden säikeen vaihtoehdossa yksittäisillä pelaajilla ei ole omaa päätäntävaltaa vaan sisäkentän tekoälysäie käskee koko joukkuetta, kuten ulkokenttä on tehty toimimaan. Säie pitäisi kirjaa, missä kukin sisäkentän pelaaja on ja pallon etenemisen seuraami-

sen perusteella käskisi pelaajia. Tämäntyyppisessä ratkaisussa säie miettisi jatkuvasti, missä pallo on, ja tekisi sen perustella päätöksiä lyönnin ja juoksemisten suhteen.

Saman tekoälyn pyörittäminen jokaisella pelaajalla erikseen vaikutti luonnollisemmalta. Tässä vaihtoehdossa tekoäly, joka pyörittää jokaista sisäkenttäpelaajaa, on samanlainen. Tämä vastaisi parhaiten todellista pelitilannetta. Jonottamisvaiheessa pelaajan tekoäly voisi tarkistaa tilannetta vaikka sekunnin välein. Kun pelaaja pääsee lyömään, alkaisi aktiivinen lyönnin mietiskely.

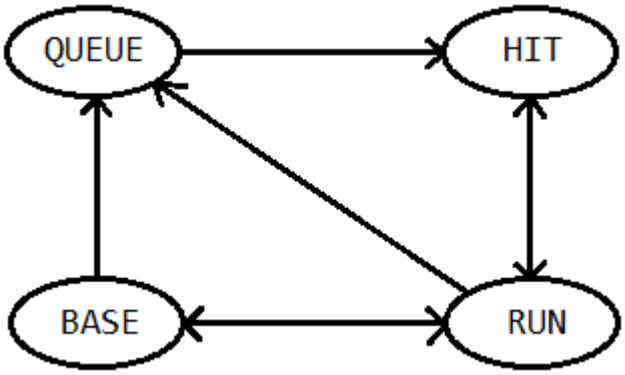
Yhdistelmävaihtoehdon etu on siinä, että kaikkia komentava tekoälysäie voi käskyttää yksittäisiä pelaajia, eikä jokaisen pelaajan tarvitse käydä läpi samanlaista analysointia pelitilanteesta. Esimerkiksi pallon seurannan voisi keskittää pääsäikeelle, jolloin yksittäisille säikeille jäisi jäljelle päättää vaikka juoksuista. Tässä tapauksessa pallon ollessa lukkarilla vain juoksemassa olevien pelaajien tarvitsisi miettiä, ehtisivätkö juosta seuraavalle pesälle, vaikka lukkari heittäisikin pallon sinne. Lukkarin ei kuitenkaan pitäisi olla niin huono heittämään palloa pesille, että kenenkään kannattaisi juosta pallon ollessa lukkarilla. Pääsäie voisi huolehtia koko lyöntivaiheesta yksin. Pelaajille, jotka eivät ole jonottamassa, voitaisiin tarpeen mukaan antaa omat säikeensä miettimään, kannattaako seuraavalle pesälle juosta, ja jos kannattaa, kannattaako juoksun aikana sukeltaa tai peruuttaakin edelliselle pesälle. Jos pallo menisi tarpeeksi kauas pesistä, voisi pääsäie pistää jokaisen kentällä olevan sisäkenttäpelaajan juoksemaan.

4.3 Johtopäätöksiä

Tässä vaiheessa oli vielä hankala sanoa, mikä vaihtoehdoista olisi paras. Päätettiin lähteä toteuttamaan tekoälyä niin, että jokaista sisäkenttäpalaajaa ohjaa oma säikeensä. Sama logiikka pyörii kaikilla pelaajilla. Tarpeen vaatiessa ajateltiin tehdä pääsäie, jolla ja jonka kautta voisi käskyttää muita pelaajia.

Päätettiin toteuttaa sisäkentän tekoäly ensin niin, että olemassa oleva ulkokentän tekoäly pelaa sitä vastaan. Lähdettiin toteuttamaan sisäkentän tekoälyä tilakoneena, joka pyörii säikeenä jokaisella sisäkentän pelaajalla. Mietiskelyssä, miten saataisiin tekoälystä mahdollisimman hyvä, tultiin siihen johtopäätökseen, että geneettiset algoritmit voisivat olla hyvä keino harjoittaa tekoälystä hyvä. Geneettisiä algoritmeja käytettäisiin pääasiassa ennen pelin julkaisua tuottamaan hyvä tekoäly. Tavoitteena oli kuitenkin

saada sisäkentän tekoälystä riittävän haasteellinen tai muuten mielenkiintoinen ilman geneettisiä algoritmeja. Kuvassa 5 näkyvät sisäkentän tekoälyjen tilat, ja mistä tilasta pääsee mihinkin tilaan.



Kuva 5. Kaavio esittää sisäkentän tekoälyn tiloja siirtymisineen.

Ihmispelaaja ohjaa sisäkenttää hiirellä ja näppäimistöllä. Muokattiin funktioita niin, että sisäkentän tekoäly voi ohjata pelin kulkua samaan tapaan kuin ihminen. Jo olemassa olevat sisäkentän ihmispelaajan käyttämät funktiot voisi ajatella jonkinlaisena tekoälyn osana. Rakennettiin sisäkentän tekoäly muuttamatta merkittävästi ennestään olevaa koodia. Siis sekä ihmispelaaja että sisäkentän tekoäly käyttävät samoja toimintoja sisäkentän pelaajien liikuttamiseen.

Pesäpallopelissä on etukäteen määritelty sisäkenttäpelaajien kulkureitit juoksemisessa ja pesillä olemisessa. Ulkokenttäpelaajille on etukäteen määritelty paikat kentällä, joihin palaavat pallon perässä juostuaan.

Sisäkentän tekoälyä toteuttaessa mietittiin, että voisi olla hyvä idea tehdä joukkuetason tekoäly sisäkentälle yksittäisiä pelaajia ohjaavien tekoälyjen lisäksi. Tällöin voisi olla helpompaa hallita kentällä jo olevien pelaajien juoksemisia. Silloin ei välttämättä olisi tarvetta yksittäisillä juoksijoilla olla omaa säiettä pyörimässä juoksuvaiheessa, jos joukkuetason tekoäly päättää milloin juostaan. Koettiin kuitenkin luontevammaksi juoksupäätöksien olevan jokaisella pelaajalla itsellään.

Saatiin syksyllä hiottua pallon putoamispaikan laskenta riittävän tarkaksi, ettei se vaatinut erityistä hiomista enää. Ulkokentän ihmispelaajalle muokkaamisen jälkeen aiottiin miettiä vielä uudestaan, miten voisi kehittää sisäkentän tekoälyä paremmaksi.

4.4 Sisäkentän tekoälyn C#-koodi

Seuraavaksi käydään läpi sisäkentän tekoälyyn liittyvää koodia selittäen samalla mitä missäkin vaiheessa on. Seuraavana on esitetty koodinpätkät tekoälytilojen luetellusta tyypistä ja koodinpätkä, jossa pelaajan tekoälytila asetetaan jonotukseen palamisen tai haavoittumisen yhteydessä. Tekoälytilan muuttuja lisättiin Playerluokkaan, jotta pelaajien tekoälytilaa voisi muokata myös tekoälysäikeen ulkopuolelta. Tämä on hyödyllistä silloin, kun pelaaja palaa tai haavoittuu. On yksinkertaisempaa asettaa pelaajan tekoälytila jonotukseen suoraan yhdistetyssä palamis- ja haavoittumiskoodissa kuin tarkkailla palamista ja haavoittumista tekoälysäikeessä.

```
public enum runnerState
{
  QUEUE,
  HIT,
  RUN,
  BASE
};

// move player back to home when burned / wounded
private void movePlayerBackToHome (Player player)
{
  player.aiState = AI.runnerState.QUEUE;
  ...
}
```

Peli-istunnon tyyppi on asetettu ennen koodien ajoa/kääntämistä Unityssä. Jos on valittu ulkokenttäpelaaminen, käynnistetään jokaiselle sisäkentän pelaajalle sisäkentän tekoäly. Muussa tapauksessa käynnistetään vain ulkokentän tekoäly, kuten seuraavasta koodinpätkästä näkyy.

```
if (Settings.instance.gameMode == Settings.gameModes.outField) {
  foreach (Player p in inField) {
    StartCoroutine(InFieldPlayerAI(p));
  }
} else {
  StartCoroutine(OutFieldAI(status));
```

```
}
```

Varsinainen sisäkentän tekoäly näyttää seuraavanlaiselta. Kyseessä on säie, jolle annetaan syötteenä pelaaja, jota säie ohjailee. Säie sisältää joitakin yleisiä muuttujia. Mikäli säikeessä ei olisi ollenkaan viiveitä, se jäisi jumittamaan koko pelin. Muuttujat ”delay” ja ”wait” säätelevät tekoälyn nukkumisia tietyissä kohdissa. Muuttujan ”execute” tarkoitus on mahdollistaa tekoälyn suorituksen keskeyttäminen missä kohtaa tahansa. Sitä ei käytetä vielä, koska vielä ei ole toteutettu sisä- ja ulkokentän pelaamisen vaihtumista. Muuttuja ”visitedField” liittyy juoksujen tekemiseen. Muuttujaa ”runSpeed” käytetään juoksijan enimmäisnopeuden laskemisessa. Ensimmäinen tila, johon pelaaja asettuu, on jonotustila.

```
IEnumerator InFieldPlayerAI (Player player)
{
  float delay = 0.3F, wait = 2F;
  player.aiState = runnerState.QUEUE;
  bool execute = true, visitedField = false;
  float runSpeed = 0F;
  while (execute) {
    switch (player.aiState) {
    case runnerState.QUEUE:
      visitedField = false;
      runSpeed = 0F;
      // Stay in queue state until player becomes a batter
      if (player.battingNumber == playerScript.getBatter().battingNumber) {
        player.aiState = runnerState.HIT;
      } else {
        yield return new WaitForSeconds(wait);
      }
      break;
```

Jonotustilassa säie nukkuu ja tarkistaa tietyin väliajoin, onko pelaajan lyöntivuoro tullut. Alun perin aiottiin laukaista säie vasta lyöjän vaihtuessa, mutta todettiin yksinkertaisemmaksi pitää tekoälysäiettä jatkuvasti päällä jokaisella sisäkentän pelaajalla. Lyöntivuoron tullessa jonottajalle pelaaja asetetaan lyöntitilaan.

4.4.1 Lyömisen toteutus

Seuraavassa koodinpätkässä on tekoälyn lyöntivaiheen alkuosa. Ensin lyöjä odottaa, että lukkarilla on pallo. Säie nukkuu 0,3 sekuntia tarkistusten välissä, kunnes ehto täyttyy ja siirrytään seuraavaan vaiheeseen. Tämän jälkeen lyöjä odottaa, että lukkari on saapunut syöttöpaikalle. Ihmispelaajan tapauksessa hiiren vasenta nappia pitäisi painaa, joten nyt tekoälysäie merkkaa hiiren vasemman napin totuusarvomuuttujan todeksi, vastaten ihmispelaajan toimintaa. Jos pelitilanne on päätynyt tapaukseen, jossa pallo pitää nollata, niin tekoäly huolehtii siitä. Näin voi käydä laittoman tapauksessa.

```
case runnerState.HIT:
  if (!pitcher.hasBall) {
    while (!pitcher.hasBall) {
      yield return new WaitForSeconds(delay);
    }
  }
  if (!GameLogic.isPitching) {
    // Make sure the pitcher is actually in the correct location
    while (Vector3.Distance(pitcher.gameObject.transform.position,
      pitcher.originalCoords) > 0.2F) {
      yield return new WaitForSeconds(delay);
    }
    GameLogic.leftMouseButtonIsDown = true;
    while (!GameLogic.isPitching) {
      yield return new WaitForSeconds(delay);
      if (GameLogic.clickCount == 2) {
        GameLogic.leftMouseButtonIsDown = true;
      }
    }
  }
```

Varsinainen lyöntiosuus on melko yksinkertainen. Toteutettiin kolme erilaista lyöntitapausta. Mikäli lyöjän lyöntivoima on tarpeeksi suuri, lyödään tavallinen lyönti. Tavallinen lyönti tarkoittaa mahdollisimman kovaa lyöntiä lähes suoraan eteenpäin niin, että pallo kimpoaa maan kautta eteenpäin. Pallo lyödään vähän sivuun keskellä olevan kopparin välttämiseksi ja maan kautta haavan välttämiseksi. On kolmen sadasosan todennäköisyys, että lyödään haava tarkoituksella. Tässä lyönnissä lyödään korkeassa kulmassa yläviistoon, niin että kopparit saavat lähes varmasti kopin. Ajateltiin pitää

tarkoitukselliset haavalyönnit huvinvuoksi pelissä. Lyöjän lyöntivoiman ollessa heikko ja jos haavaa ei tapahtunut, lyödään näpy, koska heikon lyöntivoiman takia pallo on liian helposti kopattavissa keskikentällä.

```
float ballHitHeight;
int goal = Random.Range(0, 100);
if (player.hitPower > Random.Range(60, 65)) {
  // Normal hit
  ballHitHeight = 1.25F;
  hitPower = Random.Range(95F, 100F);
  if (GameLogic.swingCounter == 1) {
    ballForce.angleH = -5F;
  } else {
    ballForce.angleH = 5F;
  }
} else if (goal <= 3) {
  // Wound on purpose
  ballHitHeight = 1.55F;
  // Try to prevent out-of-bounds with too strong batters
  if (player.hitPower > Random.Range(75F, 85F)) {
    hitPower = Random.Range(70F, 80F);
  } else {
    hitPower = 100F;
  }
  ballForce.angleH = 0F;
} else {
  // Snap hit
  ballHitHeight = 2.3F;
  hitPower = Random.Range(50F, 70F);
  ballForce.angleH = Random.Range(5F, 10F);
}
```

Tekoäly tarkkailee putoavan pallon korkeutta, ja sopivassa kohdassa asettaa taas ”hiiren vasemman napin” painetuksi. Kuvassa 6 näkyy sisäkentän näkökulmasta lyöntivaihe. Tällöin pelin päälogiikka saa lyönnin aikaiseksi. Pallon pystykulma määräytyy siitä, millä korkeudella pallo oli lyötäessä.

Kuva 6. Kuvassa näkyy lyöntitapahtuma sisäkentän näkökulmasta. Kapea pitkä palkki on lyöntivoima ja keskellä oleva asteikko on pystykulma, joka riippuu pallon korkeudesta lyöntihetkellä.

Tavallinen lyönti on mahdollisimman kovaa maan kautta kimpoava, jotta pallo menee kauas eikä tule haavaa. Lyönnin jälkeen siirrytään juoksuvaiheeseen. Tosin tällä hetkellä juokseminen päätetään jo ennen syöttämisen alkamista.

```
// Make the pitcher throw the ball
while (GameLogic.clickCount < 1) {
  if (Random.Range(0, 9) < 9) {
    allShouldRun = true;
  }
  GameLogic.leftMouseButtonIsDown = true;
  yield return new WaitForSeconds(delay);
}
while (ballForce.getAltitude() > ballHitHeight) {
  yield return new WaitForSeconds(0.03F);
}
GameLogic.leftMouseButtonIsDown = true;
```

```
if (player.hitPower < Random.Range(75F, 85F) || GameLogic.isLastSwing ||
  (playerMovement.checkIfBaseIsEmpty(Player.playerLocation.firstBase) &&
  playerMovement.checkIfBaseIsEmpty(Player.playerLocation.secondBase) &&
  playerMovement.checkIfBaseIsEmpty(Player.playerLocation.thirdBase))) {
  player.aiState = runnerState.RUN;
}
break;
```

Sisäkentän juoksijat lähtevät 90 %:n todennäköisyydellä juoksemaan heti lukkarin heitettyä pallon ilmaan syöttövaiheessa. Lyöjä jää lyömään, mikäli lyöntivoimansa on tarpeeksi suuri. Lyöjä kuitenkin juoksee joka tapauksessa, jos pesillä ei ole ketään tai kyseessä on viimeinen lyönti. Tässä olisi tarvetta lyöjien järjestämiseen jonoon niin, että kovat lyöjät saavat nopeat juoksijat tekemään juoksuja.

4.4.2 Juoksemisen toteutus

Jos lyönti onnistuu hyvin, niin juoksutkin menevät putkeen, mutta usein etenkin heikkolyöntivoimaisilla lyöjillä lyönti jää melko lyhyeksi tai sitten pallo saadaan kiinni liian nopeasti, jolloin ei oikein ehdi juosta ja kaikki juoksijat palavat. Juoksuvaiheessa tarkistetaan, missä pelaaja sijaitsee kentällä. Jokaisella pesällä on liipaisimet, jotka asettavat pelaajan nykyisen pesän muuttujan oikeaksi. Taaksepäin juokseminen huonon tilanteen sattuessa jäi toteuttamatta toistaiseksi. Niin olisi hyvä tehdä, mikäli pallo on liian lähellä seuraavaa pesää ja taaksepäin on mahdollista juosta. Siinä pitäisi tarkkailla muitakin juoksijoita, jottei samalle pesälle yritä juosta useampi pelaaja.

```
case runnerState.RUN:
  switch (player.currentBase) {
  case Player.playerLocation.field: // Player is not on a base
    visitedField = true;
    if (runSpeed < player.gameObject.rigidbody.velocity.magnitude) {
      runSpeed = player.gameObject.rigidbody.velocity.magnitude;
    }
    // Cancel running if hit does not look good
    // Dive when near next base
    if (player.isRunning && !player.isDiving && !player.isRising) {
      switch (player.targetLocation) {
      case Player.playerLocation.firstBase:
```

```
            if (5 > Vector3.Distance(player.gameObject.transform.position,
              PlayerMovement.firstBase)) {
              playerMovement.playerDive(player);
            }
            break;
          case Player.playerLocation.secondBase:
            if (5 > Vector3.Distance(player.gameObject.transform.position,
              PlayerMovement.secondBase)) {
              playerMovement.playerDive(player);
            }
            break;
          case Player.playerLocation.thirdBase:
            if (5 > Vector3.Distance(player.gameObject.transform.position,
              PlayerMovement.thirdBase)) {
              playerMovement.playerDive(player);
            }
            break;
          case Player.playerLocation.homeBaseWaiting:
            if (5 > Vector3.Distance(player.gameObject.transform.position,
              PlayerMovement.homeBasePath[1])) {
              playerMovement.playerDive(player);
            }
            break;
          }
        }
        break;
```

Sukeltamisen toteuttaminen jäi syksyllä kesken, koska siinä oli paljon erilaisia virheitä. Lopussa sukeltaminen muutettiin yksinkertaiseksi nopeutukseksi, jota pystyy käyttämään juuri ennen pesälle saapumista. Toteutettiin juoksuvaiheeseen sukeltaminen aina oltaessa lähellä pesää. Heti kun juokseva pelaaja saavuttaa pesän, tekoälytila vaihtuu pesätilaksi. Kotipesä on vähän erilainen tapaus, koska siellä ollaan myös jonotus- ja lyöntitiloissa. Siinä tarkistetaan, onko pelaaja lyöjä ja onko lyöjän juokseminen sallittu syötön jälkeen. Jos ensimmäinen pesä on tyhjä, lyöjä asetetaan juoksemaan. Jos ensimmäisellä pesällä on joku, lyöjä palaa takaisin lyöntivaiheeseen, eikä lähdekään juoksemaan. Viimeisen lyönnin jälkeen lyöjä ja kentällä olevat juoksijat laitetaan kaikki juoksemaan. Lyöjän voisi jättää palamaan itsekseen, jos kentällä on useita pelaajia, jotka eivät ehtisi seuraavalle pesälle, jolloin vain lyöjä palaisi usean

juoksijan sijasta. Monimutkaisemman juoksemispäätöksen toteutus jäi myöhemmäksi. Jos pelaaja kävi jo kentällä, mutta palasi jostain syystä takaisin kotipesälle, lyöjä asetetaan jonotustilaan.

```
case Player.playerLocation.firstBase:
case Player.playerLocation.secondBase:
case Player.playerLocation.thirdBase:
  player.aiState = runnerState.BASE;
  break;
case Player.playerLocation.homeBaseWaiting:
  if (player.battingNumber == playerScript.getBatter().battingNumber
    && GameLogic.allowRunAfterPitch) {
    if (playerMovement.checkIfBaseIsEmpty(Player.playerLocation.firstBase)) {
      batterShouldRun = true;
    } else if (!GameLogic.isLastSwing) {
      player.aiState = runnerState.HIT;
    } else {
      allShouldRun = true;
      batterShouldRun = true;
    }
  } else if (visitedField) {
    // Player arrived at home base
    player.aiState = runnerState.QUEUE;
  } else {
    if (!player.isRunning) {
      player.aiState = runnerState.QUEUE;
    }
  }
        break;
}
yield return new WaitForSeconds(0.2F);
break;
```

Pesätilassa tekoäly laskee, kannattaisiko seuraavalle pesälle juosta ilman muualta tulevaa käskyä. Ensin lasketaan, kauanko juoksijalta kestää seuraavalle pesälle ja sen jälkeen, kauanko pallolla kestää seuraavalle pesälle. Jos juoksijan aika on lyhyempi, juoksija lähtee juoksemaan. Tämän jälkeen on kahden sekunnin odotus, jonka tarkoitus on varmistaa, että juoksija ehtii pois pesältä ennen tekoälyn pyörimisen jatkumista.

```
    case runnerState.BASE:
      // Calculate time to drop point
      float ballTime = 0F, myTime = 0F;
      switch (player.currentBase) {
      case Player.playerLocation.firstBase:
        myTime = playerMovement.calculateRunTime(player,
                   PlayerMovement.secondBase);
        ballTime = calcBallTime(PlayerMovement.secondBase);
        break;
      case Player.playerLocation.secondBase:
        myTime = playerMovement.calculateRunTime(player,
                   PlayerMovement.thirdBase);
        ballTime = calcBallTime(PlayerMovement.thirdBase);
        break;
      case Player.playerLocation.thirdBase:
        //myTime = playerMovement.calculateRunTime(player,
                     PlayerMovement.homeBasePath[0]);
        myTime = playerMovement.calculateRunTime(player,
                   PlayerMovement.homeBasePath[1]);
        ballTime = calcBallTime(PlayerMovement.hitLocation);
        break;
      default:
        break;
      }
      if (myTime < ballTime) {
        playerMovement.movePlayerToNextBase(player);
        yield return new WaitForSeconds(2F);
      }
      yield return new WaitForSeconds(0.3F);
      player.aiState = runnerState.RUN;
      break;
    }
  }
}
```

Seuraavana on koodi, jota käytetään juoksemisen kannattavuuden laskemiseen. Pallon heittonopeudeksi on asetettu kiinteä 33 metriä sekunnissa. Jos pallo on jo kopattu, lasketaan vain heittoaika kohteeseen eli seuraavalle pesälle. Muussa tapauksessa lasketaan pallon nykyisestä sijainnista heittoaika kohteeseen ja kauanko lähimmällä

koppaajalla kestää juosta pallon nykyiseen sijaintiin. Jos pallo ei ole vielä pysähtynyt, käytetään laskussa pallon oletettua pysähtymispaikkaa, jota lasketaan jatkuvasti muualla.

```
private float calcBallTime (Vector3 target)
{
  float time = 0F;
  Vector3 trackPoint;
  if (BallCatch.ballIsCaught) {
    trackPoint = ballForce.gameObject.transform.position;
  } else {
    trackPoint = (ballForce.gameObject.rigidbody.velocity.magnitude < 0.5F) ?
      ballForce.gameObject.transform.position : BallGroundHit.touchCoords[1];
    Player catcher =
      playerScript.getPlayerById(nearestPlayerCord(trackPoint));
    // Run time to actual catch point
    time = playerMovement.calculateRunTime(catcher, trackPoint);
  }
  // Throw time to target. 33 m/s
  time += Vector3.Distance(trackPoint, target) / 33;
  return time;
}
```

Toteutettiin syksyllä juoksemisen niin, ettei pesälle voi juosta, jos siellä on jo joku. Oikeasti pesälle saa juosta, vaikka siellä olisikin joku, mutta koska kahta juoksijaa ei saa olla samalla pesällä, ei mahdollistettu juoksemista varattuun pesään.

5 Ulkokentän pelimekaniikan toteutus

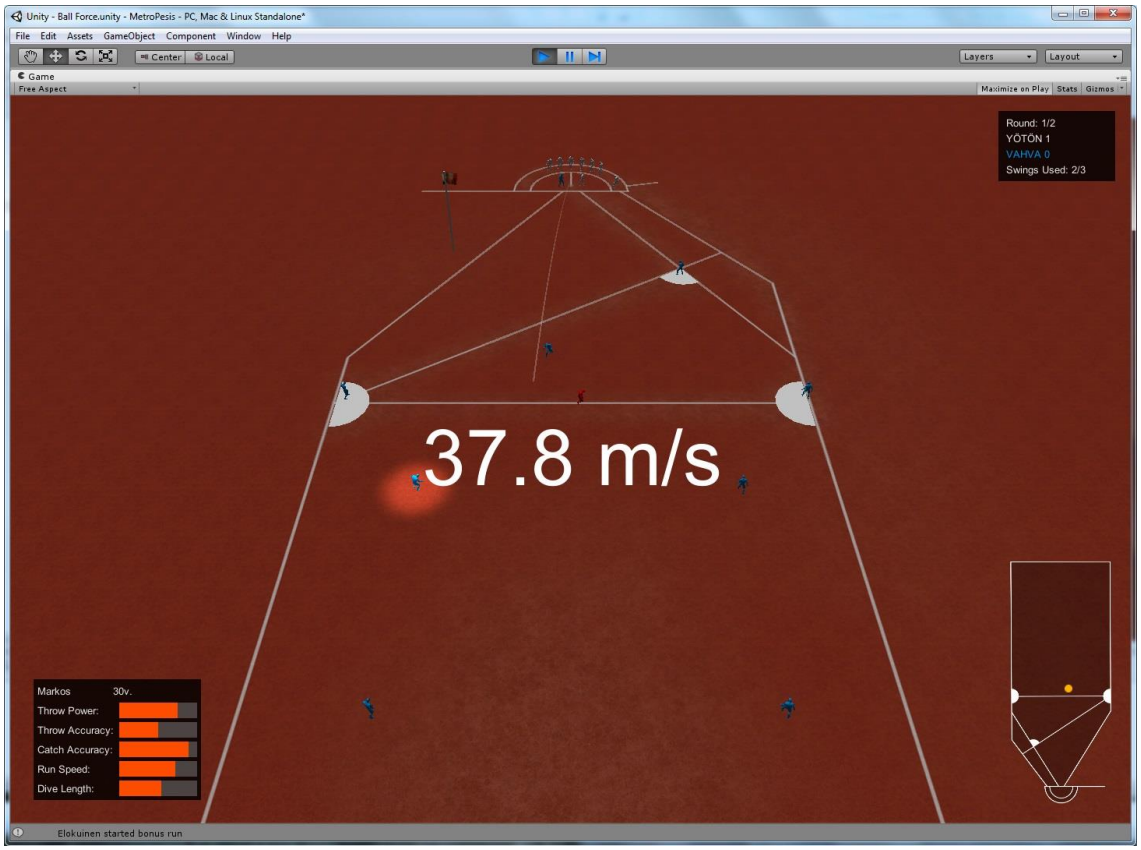
Ensin toteutettiin sisäkentän tekoäly pelaamaan olemassa olevaa ulkokentän tekoälyä vastaan. Tavoitteena oli mahdollisimman pienillä muutoksilla ulkokentän tekoälyyn saada se ja ihmispelaaja käyttämään samoja koodinpätkiä pelaajien liikuttamiseen. Toistaiseksi pelivuoro ei vaihdu ulkokentällä ja sisäkentällä pelaamisen välillä. Ajateltiin, että ulkokentän toteuttamisen jälkeen voisi toteuttaa vuoronvaihtumisen, mutta se jäi tulevaisuuteen. Olemassa oleva kahden pelaajan sisäkentän peli vaihtaa vain sisä-

kentällä ihmisen pelattavaa joukkuetta. Voisi myös tehdä kahden pelaajan ulkokentän pelaamisvaihtoehdon.

5.1 Ohjaaminen

Pesäpallopelissä oli jo ominaisuus, jolla voi valita manuaalisesti, kenelle pallo heitetään ulkokentällä. Muokattiin se kuitenkin uusiksi, koska se oli niin huonosti tehty. Ulkokentän olemassa olevalla tekoälyllä on neljä tapaa ohjata pelaajia. Ulkokenttäpelaaja voi joko seurata pallon putoamispaikkaa, toista putoamispaikkaa, itse palloa tai palata lähtöpisteeseensä. Pallon putoamispaikkojen laskentakaavojen käyttö tuntuu huijaukselta ihmispelaajalle annettavaksi.

Aikaisemmin oli toteutettu pelaajan valinta hiirellä niin, että kotipesästä katsottuna takakenttää kohti enimmäisetäisyys hiirestä pelaajaan kasvaa. Koska ulkokenttäpelaamisessa siirrettiin kamera takakentälle katsomaan kotipesää kohti, kotipesää lähellä olevat pelaajat olivat turhan hankalasti valittavissa hiirellä. Muutettiin valintaetäisyys kasvamaan päinvastaiseen suuntaan. Kuvassa 7 näkyy toisen kopparin valinta lyönnin jälkeen. Pallo on jo ehtinyt ohittaa ensimmäisen koppaajaksi valitun pelaajan.



Kuva 7. Toinen koppari valitaan.

Kaksi käskyä riittää ihmiselle ulkokenttäpelaajien liikuttamiseen; seuraa palloa ja mene hiirellä osoitettavaan paikkaan. Ihmispelaajan kannattaisi tällöin ensin liikuttaa koppareita ihmisen olettamaan putoamispaikkaan, ja jos näyttää siltä, ettei pallo ole koppausetäisyydellä, käskisi kopparia seuraamaan palloa. Lähdettiin toteuttamaan ulkokentän ohjaamista kytkemällä olemassa oleva tekoäly pois ja tekemällä kaksi komentoa ulkokenttäpelaajille: seuraa palloa ja heitä pallo minulle. Todettiin, että ulkokenttäpelaajien manuaalinen ohjailu olisi turhan hankalaa.

Aikaisemmin ulkokentän pelaajia ohjailtiin kahdesta paikkaa. Ulkokentän tekoäly antaa ulkokenttäpelaajille tavoitekoordinaatteja, mihin pelaajien liikkeistä vastaava C#-luokassa oleva sitä varten tehty säie reagoi. Se käy läpi ulkokenttäpelaajia ja liikuttaa niitä niille asetettuja koordinaatteja kohti. Kun siis annettiin ulkokenttäpelaajille kohdekoordinaatteja manuaalisesti tekoälyn ulkopuolelta, koodi hoiti jo valmiiksi pelaajien palaamisen lähtöpisteisiinsä tavoitteensa saavutettuaan. Pallo myös lentää niin nopeasti, että olisi aika hankalaa siirtää manuaalisesti ulkokenttäpelaajia parempiin koppauspaikkoihin. Seuraavaa koodinpätkää käytetään ulkokenttäpelaajien ohjaamiseen.

```
if (Settings.instance.gameMode == Settings.gameModes.outField) {
  if (mouseOverPlayer != null && mouseOverPlayer.teamNumber == 2) {
    // Follow ball
    if (Input.GetMouseButtonDown(0)) {
      moveToCoords[mouseOverPlayer.playerId] = followBall;
    }
    // Go to pointed location
    // Jump to catch flying ball
    // Throw ball to pointed player
    if (Input.GetMouseButtonDown(1)) {
      AiThrow.targetPlayerId = mouseOverPlayer.playerId;
    }
  }
} else { // Controls for infield players
  ...
}
```

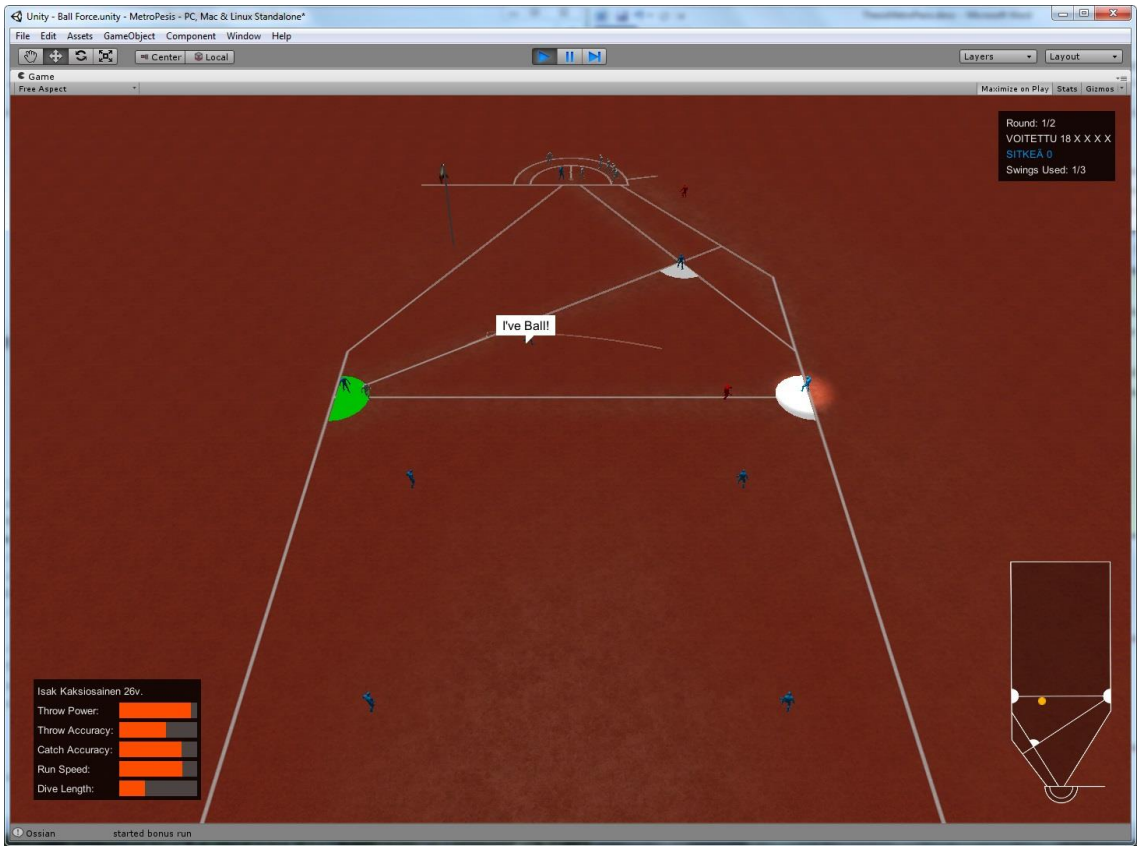
Koodi on osa pelaajien liikuttamisesta vastaavan luokan funktiota, jota kutsutaan aina grafiikan päivittyessä. Se tarkkailee, onko hiiri kenenkään pelaajan päällä ja mitä hiiren- ja näppäimistön nappeja on painettu. Jotta saatiin sisäkentän tekoäly pelaamaan, lisättiin pariin kohtaan koodissa tai-tyyppinen tarkistus, onko tekoäly niin sanotusti painanut tiettyä nappia. Jos pelimuodoksi on valittu ulkokenttäpelaaminen, tarkistetaan, onko hiiri kenenkään ulkokenttäpelaajan päällä. Jos hiiren vasenta nappia on painettu, pelaaja asetetaan seuraamaan palloa. Jos taas hiiren oikeaa nappia on painettu, pallon heitosta vastaavan luokan heiton kohdepelaaja asetetaan hiirellä osoitetuksi pelaajaksi. Aiottiin toteuttaa myös pelaajan manuaalinen ohjaamisen hiirellä osoitettavaan paikkaan ja pelaajan hyppäämisen pallon saamiseksi kiinni ilmasta, mutta ne jäivät peliä seuraavaksi jatkokehittävien toteutettaviksi.

Ulkokenttäpelaajien liikuttamisesta vastaavaa funktiota muokattiin niin, että pallon hakijoista kauimpana oleva pelaaja luopuu pallon hakemisesta. Kun funktio käy pelaajat läpi, se laskee samalla, kuinka monta pelaajaa seuraa palloa ja kuka on kauimpana. Silmukan jälkeen kauimpana oleva pelaaja asetetaan palaamaan lähtöpisteeseen. Seuraavalla pelaajien läpikäynnillä edellisellä kierroksella asetettu kauimpana oleva pelaaja sitten pistetään oikeasti palaamaan lähtöpisteeseensä.

Koska pallonseuranta ei ennakoi, minne pallo menee, palloa seuraamaan asetetut pelaajat kiertävät liikaa. Pallonseurantaa voisi muuttaa niin, että seuraajat ennakoisivat pallon liikkeitä. Ulkokentän tekoälyssä pallonseuranta on luonnollisempaa, koska kopparit seuraavat pallolle jatkuvasti laskettuja putoamispaikkoja ja vasta, kun pallo on maassa, kopparit seuraavat palloa suoraan.

Koppaus tapahtuu itsestään, kunhan pallo on kopparin koppietäisyydellä. Voisi olla käsky myös lähtöpisteeseen palaamiseen, mutta toistaiseksi se tapahtuu automaattisesti pelaajan kopattua pallon tai lakattua seuraamasta palloa. Pelin vaikeustasoa voisi nostaa tekemällä lähtöpisteeseen paluun manuaaliseksi. Tällöin peliä pelaava ihminen voisi myös sijoitella ulkokenttäpelaajia haluamiinsa paikkoihin. Kopparin kopattua pallon, tarvitaan käsky pallon heittämiseksi jollekin toiselle pelaajalle. Jos pallo ei yllä, se jää vain matkan varrelle. Tarvetta ei ole heittää palloa muita kuin toisia koppareita, pesävahteja ja lukkaria kohti. Jos pallo lentää kopparin yläpuolella, mutta sen saisi hyppäämällä kiinni, tätä varten voisi myös olla oma käskynsä. Ulkokentän olemassa oleva tekoäly vain antaa pelaajille määränpäitä ja pelaajia liikuttaa sitä varten erikseen tehty säie.

Toteutettiin pallon heittäminen niin, että hiiren oikealla napilla valitaan kohdepelaaja, jolle heitetään pallo. Aluksi tehtiin se niin, että pallo pitää olla kopattuna, jotta sen voi heittää, mutta peliä testaillessa huomattiin, että on mukavampaa olla mahdollista päättää kohdepelaaja milloin tahansa pelin aikana. Kuvassa 8 näkyy pallon koppaajan valinta. Koppaaja, jolle pallo heitetään, on korostettu valokehällä.



Kuva 8. Kuvassa näkyy pallon heitto ja heiton kohdepelaajaa osoitetaan hiirellä, jolloin se on korostettuna.

Jos pelin vaikeustasoa myöhemmin tarvitsee nostaa, voisi kohdepelaajan valinnan tehdä mahdolliseksi vain pallon ollessa kopattuna. Pallon heittäminen on oma säikeensä, joka käynnistetään pelin alussa. Se tarkkailee, onko pallo kopattuna ja onko kohdepelaaja asetettuna.

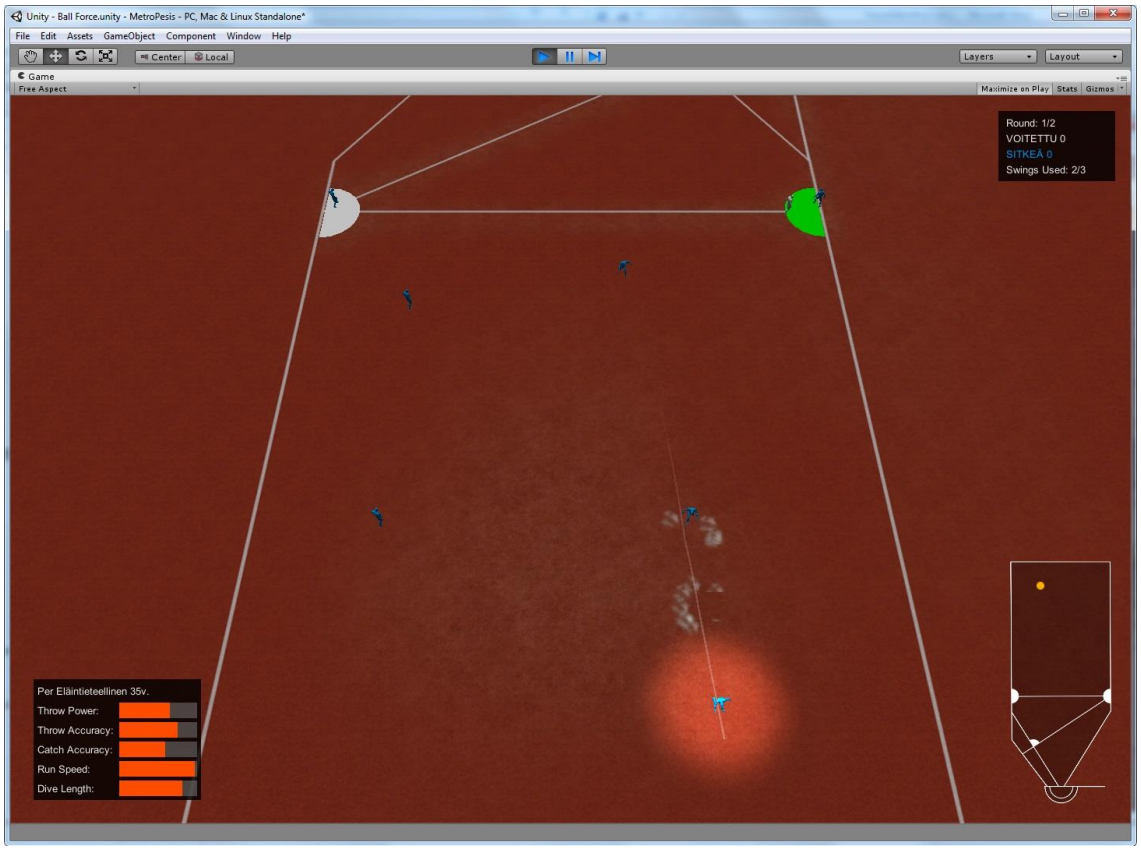
```
IEnumerator BallPasser ()
{
  while (true) {
    if (BallCatch.ballIsCaught && targetPlayerId != -1) {
      // Reset ball followers, if they didn’t register ball being caught
      for (int i=9; i<playerMovement.moveToCoords.Length; i++) {
        if (playerMovement.moveToCoords[i] == PlayerMovement.followBall) {
          playerMovement.moveToCoords[i] = PlayerMovement.giveUpAndReturn;
        }
      }
      throwBall();
      targetPlayerId = -1;
```

```
    }
    yield return new WaitForSeconds(0.5F);
  }
}
```

Jotta muut pallon hakijat eivät jatkaisi pallon seuraamista heiton jälkeen, ennen heittoa käydään läpi jokainen ulkokenttäpelaaja ja asetetaan pallon seuraajat palaamaan takaisin lähtöpisteillensä. Testeissä huomattiin, että joskus pallon seuraajat eivät ehtineet havaita kopin tapahtuneen ja jatkoivat pallon seuraamista heiton jälkeen. Näin pystyi käymään, jos koppari heitti pallon saman tien eteenpäin.

5.2 Kamera

Kun oli toteutettu ulkokentälle toiminnot kopparien ohjaamiseen, eniten muokkaamista vaati kameran sijainti ja käytös pallon liikkuessa. Mikäli kamera oli liian kaukana, kotipesä näkyi liian pienenä kaukana. Kameran ollessa lähellä, takakenttä jäi piiloon. Tästä syystä päädyttiin sijoittamaan kamera takakentälle, niin että kotipesälle näkyy vielä siedettävästi. Asetettiin kamera seuraamaan palloa itsestään. Tehtiin myös kameran kulma muutettavaksi hiiren rullalla. Säädettiin kamera kuitenkin niin, ettei pallo voi kadota kamerasta, jos sitä ei ole kopattuna. Eikä kamera myöskään voi mennä liian paljon eteen tai taaksepäin. Kuva 9 havainnollistaa kameran seuraavan palloa takakentällä.



Kuva 9. Kamera seuraa palloa sen mennessä takakentälle.

Heti alussa ulkokentän ohjaamista toteuttaessa siirrettiin kamera takakentälle. Alla oleva funktio siirtää kameran määritettyyn sijaintiin takakentälle katsomaan etukäteen määritetyssä kulmassa. Harkittiin myös yhdessä vaiheessa kameran siirtämistä kentän sivuun katsomaan toiselta sivulta toiselle niin, että kotipesä olisi näytön vasemmassa laidassa ja takakenttä oikeassa laidassa.

```
public static void MoveToOutFieldView ()
{
  currentView = CameraView.outField;
  iTween.MoveTo(thisCamera, iTween.Hash("position",
    new Vector3(-0.5F, 50F, -40F), "time", 1.5F * bulletTime, "easeType",
    "EaseInOutSine", "oncompletetarget", thisCamera));
  iTween.RotateTo(thisCamera, iTween.Hash("rotation", outFieldRotation,
    "time", 1.25F * bulletTime, "easeType",
    "EaseOutSine", "oncompletetarget", thisCamera));
}
```

Koska pallo voi joutua pitkälle takakentälle ja ulkopuolellekin, tehtiin kamera seuraamaan palloa sen mennessä tarpeeksi ylös tai alas näytöllä. Ensin otetaan kulma kamerasta suoraan eteenpäin. Tämän kulman nollakohdat ovat vaakasuunnassa eteen ja taaksepäin. Keskellä kameran alapuolella on 90 asteen kulma. Koska Unity antaa kulman 0 ja 90 asteen väliltä kummallakin puolella 90 astetta, tarkistetaan, onko kamerasta eteenpäin lähtevän vektorin suunta kentän etu- vai takapuolelle. Näin takakentälle saadaan negatiivinen kulma, jotta kameran automaattinen pyörittely onnistuisi 90 asteen rajan yli. Tämän jälkeen pelikentälle lasketaan koordinaattirajat, joiden sisällä pallon on oltava, jollei se ole kopattuna.

```
if (currentView != CameraView.overview) {
  if (currentView == CameraView.outField) {
    float forwardAngle = thisCamera.transform.eulerAngles.x;
    if (thisCamera.transform.forward.z < 0) {
      forwardAngle = -forwardAngle;
    }
    float lowerAngle = forwardAngle + 20F;
    float upperAngle = forwardAngle - 20F;
    float lowerLimit = thisCamera.transform.position.z +
      thisCamera.transform.position.y /
      Mathf.Tan(lowerAngle * Mathf.Deg2Rad);
    float upperLimit = thisCamera.transform.position.z +
      thisCamera.transform.position.y /
      Mathf.Tan(upperAngle * Mathf.Deg2Rad);

    bool up = Input.GetAxis("Mouse ScrollWheel") > 0;
    bool down = Input.GetAxis("Mouse ScrollWheel") < 0;
    int amount = 150;
    if (!BallCatch.ballIsCaught && !up && !down) {
      amount = 20;
      up = ballForce.transform.position.z > upperLimit;
      down = ballForce.transform.position.z < lowerLimit;
    }
```

Kameraa voi siirtää joka tapauksessa, jos hiiren rullaa käytetään. Koodissa tarkistetaan, onko rullattu jompaankumpaan suuntaan. Jos palloa ei ole kopattu eikä hiiren rullaa rullattu, tarkistetaan, onko pallo sille laskettujen koordinaattirajojen sisällä. Tätä tietoa käytetään kameran automaattiseen kääntämiseen. Kamera palautuu nopeasti

tilaan, jossa koppaamaton pallo näkyy, vaikka yrittäisi rullata hiirellä näkymää muualle. Seuraavassa koodissa annetaan kameran liikuttelukäskyt. Jos näkymä on menossa ylös- tai alaspäin sille asetettujen raja-arvojen yli, kamera liikkuu itsestään takaisinpäin vähäsen.

```
    if (thisCamera.transform.eulerAngles.x > 25) {
      if (up) {
        transform.Rotate(Vector3.left * Time.deltaTime * amount);
      } else if (down) {
        transform.Rotate(Vector3.right * Time.deltaTime * amount);
      }
    } else {
      if (up) {
        transform.Rotate(Vector3.right * Time.deltaTime * 300);
      } else if (down) {
        transform.Rotate(Vector3.left * Time.deltaTime * 300);
      }
    }
  } else { // For infield camera rotation
    ...
    }
  }
}
```

En ole aivan tyytyväinen kameraan ja sen liikkeisiin, koska pallon mennessä pitkälle kotipesä ja joissakin tapauksissa muutkin pesät menevät piiloon, jolloin pallon koppaajia ei voi valita enää pesien läheltä. Toisaalta yleensä jos pallo menee niin pitkälle, ketään ei ehdi enää polttaa muutenkaan.

6 Testaus ja arviointi

Ohjelmakoodia kirjoittaessa ei koettu tarpeelliseksi varsinaisen debuggerin käyttöä. Eteen tulleet ongelmat olivat riittävän yksinkertaisia ratkaistaviksi, jotta pärjättiin koodiin väliaikaisesti kirjoitetuilla konsoliviesteillä. Käytettiin Unityn ilmaista versiota koko pelinkehityksen aikana. Unitystä on olemassa myös kaupallinen versio, jonka tärkeimpänä

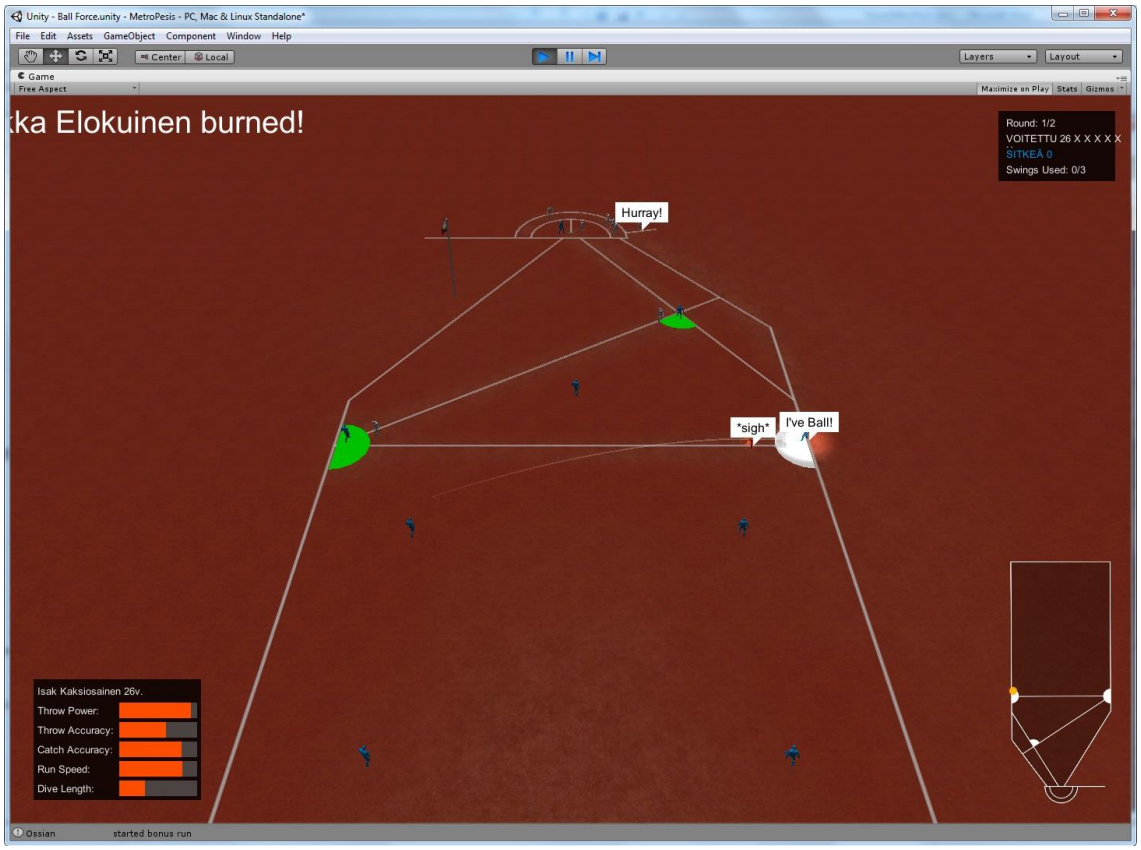
pitämäni lisä on koodin profilointi. Sillä voi jäljittää resursseja vieviä kohtia ja niin vähentää suorittimille tulevaa kuormaa nostaen samalla myös kehysnopeutta. [7.]

6.1 Sisäkentän tekoälyn testaus

Koska toteutettiin sisäkentän tekoäly äärellisenä tilakoneena, sen testaaminen oli helppoa. Toteuttamisen eri vaiheissa lisättiin sopivia debug-tulostuksia koodin eri kohtiin, joiden avulla selvitettiin milloin mitäkin koodissa olevaa ongelmaa. Yksinkertaisuuden vuoksi ja optimointisyistä sisäkentän tekoälyn koko kehityksen aikana annettiin ulkokenttätekoälyn pelata sisäkentän tekoälyä vastaan. Aloitettiin toteutuksen lyönnistä. Ensin pelattiin normaalisti sisäkenttää sen selvittämiseksi, minkälaiset lyönnit ovat hyviä mihinkin tarkoitukseen. Oikeastaan jo syksyllä ohjelmistotuotantoprojektilla kehittäessä pesäpallopeliä havaittiin parhaan lyönnin olevan maan kautta kimpoava kovaa lyöty pallo kentän keskelle ulkokenttäpelaajien väliin. Tässä tapahtuu kuitenkin vaihtokauppa, koska ohjelmoitiin lyönti niin, että mitä kovemmaksi lyöntivoima on asetettu, sitä epätarkempi lyönti on.

6.2 Ulkokentän ohjattavuuden testaus

Toteuttamisen aikana ja sen jälkeen pelasin melko paljon ulkokenttää itse (ulkokentän tekoälyn sijasta). Olen yllättävän tyytyväinen, miten haasteellisen olen saanut sisäkentän tekoälystä, vaikka se ei olekaan kovin monimutkainen. Sisäkentän pelaajat saavat aika helposti tehtyä juoksuja pesien välillä ja kotiin myös. Saan silti jonkin verran paloja tehtyä. Kuvassa 10 näkyy sisäkenttäpelaajan onnistunut polttaminen kolmospesältä.



Kuva 10. Kuvassa näkyy sisäkenttäpelaajan palaminen kolmospesältä ja samaan aikaan toinen sisäkenttäpelaaja ehti tehdä juoksun kotiin.

Joskus haavakin onnistuu, vaikka se on tosi harvinainen lyönti toteutetussa sisäkentän tekoälyssä. Ennen kuin tehtiin ulkokentästä manuaalisesti ohjattava, annettiin tekoälyjen pelata toisiaan vastaan ja selvitettiin sillä jumitustilanteita.

7 Yhteenveto

Työn tuloksena pesäpallopelissä ihminen voi pelata myös ulkokenttää sisäkentän tekoälyä vastaan sisäkentän pelaamisen lisäksi. Tästä eteenpäin seuraava ohjelmistotuotantoporukka tai joku insinöörityön tekijä voi kehittää peliä vielä pidemmälle. Mielestäni tarvitsee laittaa peligrafiikat kuntoon niin voisi jo julkaista jonkinlaisen version isommalle yleisölle. Työn aikana ei esiintynyt suuria ongelmia.

Tekoälyjen taustatutkimuksen perusteella päädyttiin tekemään sisäkentän tekoäly äärellisenä tilakoneena. Tulevaisuudessa peliä mahdollisesti jatkokehittävät voisivat tehdä sekä sisä- että ulkokentän tekoälyistä entistä parempia oppivilla tekoälymenetelmil-

lä, kuten geneettisillä algoritmeilla ja neuroverkoilla. Alun perin aiottiin hioa sisäkentän tekoälyä sen jälkeen, kun oli toteutettu ulkokentän ohjattavuus ihmispelaajalle. Pelatessa huomattiin kuitenkin, että sisäkentän tekoäly on yllättävän haasteellinen yksinkertaisuudestaan huolimatta.

Tehtiin ulkokentälle kaksi koppaajien ohjausvaihtoehtoa: pallon seuraaminen ja kenelle heitetään pallo. Suunniteltiin vielä kaksi muuta ohjausvaihtoehtoa: hyppää ilmaan yli lentävän pallon koppaamiseksi ja mene hiirellä osoitettavaan paikkaan. Näiden toteutus jäi tuleville jatkokehittäjille. Kameran liikkeet, sijainti ja kuvakulmat ovat ihan toimivia, mutta niitä voisivat tulevat jakokehittäjät parannella.

Kun pistää ulko- ja sisäkentän tekoälyt pelaamaan toisiaan vastaan, niin ulkokentän tekoäly saa tehokkaasti poltettua sisäkentän pelaajia, mutta juoksuja kuitenkin tulee joskus. Ihmisen on hankalampaa pelata ulkokenttää kuin tekoälyn, joten palojen ja haavojen tekeminen on selvästi haasteellisempaa kuin tekoälyllä. Omissa testipeleissäni sisäkentän tekoäly sai enemmän juoksuja, kuin sain sitä poltettua. Tavoite useammasta juoksusta sisäkentälle kierroksen aikana toteutui. Onnistuin mielestäni työssä hyvin. Työni aikana opin erilaisista tekoälymenetelmistä eri peleissä ja miten tekoälyjä voi käytännössä toteuttaa.

Lähteet

1 Wexler, James. 2002. Artificial Intelligence in Games. Rochester: University of Rochester.

2 Grzyb, Janusz. 2005. Artificial Intelligence in Games. Software Developer's Journal Magazine.

3 Seeman, Glenn & Bourg, David M. 2004. AI for Game Developers. O'Reilly.

4 Schwab, Brian. 2009. AI Game Engine Programming. Boston: Course Technology.

5 Buckland, Mat. 2002. AI Techniques for Game Programming. Cincinnati: Premier Press.

6 Millington, Ian. 2009. Artificial intelligence for games. Burlington: Morgan Kaufmann.

7 Profiler (Pro only). 2013. Verkkodokumentti. <https://docs.unity3d.com/Documentation/Manual/Profiler.html>. Luettu 10.4.2014.