Oskari Leppäaho

Mikromanageroinnin toteuttaminen StarCraft-reaaliaikastrategiapeliin geneettisellä ohjelmoinnilla muodostettujen potentiaalikenttien avulla

Tietotekniikan pro gradu -tutkielma

13. joulukuuta 2015

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Oskari Leppäaho

Yhteystiedot: oskari.leppaaho@gmail.com

Ohjaajat: Tuukka Puranen ja Timo Männikkö

Työn nimi: Mikromanageroinnin toteuttaminen StarCraft-reaaliaikastrategiapeliin geneettisellä ohjelmoinnilla muodostettujen potentiaalikenttien avulla

Title in English: Implementing micromanagement for the real time strategy game StarCraft using potential fields formed with genetic programming

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Pelit ja pelillisyys

Sivumäärä: 76+103

Tiivistelmä: Tämä tutkielma selvittää geneettisen ohjelmoinnin sovellettavuutta sellaisten potentiaalikenttien optimointiin, jotka ohjaavat taistelussa reaaliaikastrategiapelien yksiköiden mikromanagerointia. Tutkimusympäristönä käytetään StarCraft-peliä. Sovellettavassa menetelmässä pelin yksiköiden ohjaaminen potentiaalikentillä tapahtuu siten, että pelin kohteet: omat yksiköt, vihollisen yksiköt ja pelialueen ulkoreunat, aiheuttavat kukin potentiaalikentän. Omia yksiköitä liikutetaan siihen suuntaan, jossa kohteiden potentiaalikenttien yhteisvaikutus on voimakkain. Geneettistä ohjelmointia käytetään optimoimaan eri kohteiden luomien potentiaalikenttien voimakkuutta määrittäviä funktioita.

Menetelmä suoriutui huonommin kuin aikaisemmassa tutkimuksessa käytetty käsin luotujen potentiaalifunktioiden vakioiden optimointi geneettisellä algoritmilla. On mahdollista, että geneettisen ohjelmoinnin soveltaminen kyseiseen ongelmaan vaatisi huomattavasti suurempaa populaation kokoa kuin tässä tutkimuksessa käytetyt 128 ja 500 yksilöä.

Avainsanat: tekoäly, reaaliaikastrategiapelit, geneettinen ohjelmointi, potentiaalikentät, StarCraft

Abstract: This thesis investigates the applicability of genetic programming to optimizing the potential fields that control the units in battle in a real time strategy game. The research was

conducted in a game called StarCraft. The applied method of guiding the units with potential fields works by generating potential fields for each of the significant objects in the game: the player's own units, the enemy units and the outer edges of the playing area. The player's units are then moved in the direction in which the combined effect of the different potential fields is the highest. Genetic programming is used for optimizing the functions that define the potential fields for the different objects.

The performance of the method was inferior to an earlier study where the potential fields were crafted by hand and the constants were then optimized with the genetic algorithm. It is possible that a significantly larger population size than that of 128 and 500 individuals used in this study would be required to apply genetic programming to this problem.

Keywords: artificial intelligence, real time strategy games, genetic programming, potential fields, StarCraft

Termiluettelo

Agentti	tarkkailee ympäristöään sensoreiden kautta ja vaikuttaa ympäristöönsä aktuaattoreiden kautta.
A*-reitinhaku	on A*-hakua hyödyntävä, suosittu menetelmä lyhyimmän reitin hakuun.
Botti	on agentti, joka toimii ohjelmistoympäristössä.
Fog of War	on monissa strategiapeleissä käytetty pelimekaniikka, joka piilottaa pelialueen maaston ja vastustajan yksiköt pelaajan näkyvistä silloin kun pelaajan omia yksiköitä ei ole riittävän lähellä.
FPS-pelit	(first person shooter) ovat pelihahmon näkökulmasta kuvattuja ammuntapelejä.
Mikromanagerointi	tarkoittaa RTS-peleissä yksiköiden yksityiskohtaista kontrollointia taistelutilanteessa.
MOBA-pelit	(multiplayer online battle arena) ovat pelejä, joissa kukin pelaaja ohjaa yhtä voimakasta yksikköä tavoittenaan tuhota vihollisjoukkueen päärakennus heikompien, tekoälyn ohjaamien yksiköiden avustamana.
Osumapisteet	(engl. *hit points, HP*) ovat peleissä yleinen tapa mitata pelin kohteiden kuntoa. Kohteen vahingoittuessa sen osumapisteiden määrä vähenee. Pisteiden vähennyttyä nollaan kohde tuhoutuu.
RTS-pelit	eli reaaliaikastrategiapelit ovat strategiapelien alalaji, jota edustavissa peleissä pelaajat komentavat yksiköitään reaaliaikaisesti. Lyhenne RTS tulee sanoista real time strategy.
Tukikohta	(engl. *base*) tarkoittaa RTS-pelien kontekstissa keskittymää pelaajan rakennuksia.
Yksikkö	on RTS-peleissä pienin kokonaisuus, jolle pelaaja pystyy antamaan käskyjä. Ne voivat liikkua pelimaailmassa ja hyökätä kohti muita yksiköitä.

Kuviot

Taulukot

Sisältö

1 Johdanto

Tämän tutkielman tavoitteena on kehittää eteenpäin potentiaalikenttiin perustuvaa menetelmää yksiköiden mikromanagerointiin reaaliaikastrategiapeleissä.

1.1 Taustaa

RTS- eli reaaliaikastrategiapelit (engl. real time strategy) ovat olleet kirjoittajan eniten pelaamia videopelejä noin vuosikymmenen ajan. Erityisesti hänen suosiossaan ovat olleet Blizzard Entertainment -studion tuottamat StarCraft, Warcraft III ja StarCraft II -pelit. Warcraft III oli myös ensimmäinen kirjoittajan pelaama verkkomoninpeli. Moninpeli onkin RTS-pelien kiehtovin pelimuoto. Kilpailullisuus, strategisen suunnittelun ja nopean toiminnan tasapaino, omien taitojen näkyvä kehittyminen ja kaksinpeli, jossa tulos riippuu täysin pelaajasta itsestään, ovat RTS-peleissä tasapainossa, jota muista peligenreistä on vaikea löytää. RTS-pelit ovat lisäksi mainioita yleisölajeja: Niiden strateginen ulottuvuus ja näkökulma, jossa kamera kuvaa pelitapahtumia lintuperspektiivistä, tekevät niistä helpompia seurattavia kuin taktisemmat pelit, kuten FPS-pelit, joissa pelitapahtumia seurataan yksittäisen pelaajan pelihahmon ensimmäisestä persoonasta tai tappelu- ja MOBA-pelit, joissa merkittävät tapahtumat seuraavat toisiaan toisinaan vain millisekuntien viiveellä. Lisäksi erilaiset yksiköt tuovat peliin monipuolisuutta, mutta yksiköiden ja erityiskykyjen määrä on MOBA-peleihin verrattuna vielä opittavissa melko nopeasti.

Inspiraatiota potentiaalikenttien käyttämiseen yksiköiden kontrolloinnissa ovat antaneet vuoden 2010 AIIDE:n StarCraft-tekoälyturnauksen voittanut Overmind-botti (Huang 2011) sekä tutkimukset (Sandberg 2011) ja (Rathe ja Svendsen 2012). Sandberg (2011) sekä Rathe ja Svendsen (2012) käyttivät yksiköiden ohjaamiseen potentiaalikenttiä, joiden funktiot muodostettiin kohdealuetuntemusta hyödyntämällä manuaalisesti. Geneettistä algoritmia käytettiin funktioiden parametrien optimointiin. Tällä menetelmällä luodut funktiot eivät välttämättä ole optimaalisia. Luomalla funktiot geneettisellä ohjelmoinnilla saatetaan myös löytää helpommin käyttäytymismalleja, jotka eivät perustu ohjelmoijan valmiiksi tuntemiin taktiikoihin.

1.2 Tutkielman rakenne

Tässä tutkielmassa toteutetaan StarCraft-peliä pelaava ohjelmisto. Toteutettava ohjelmisto käyttää yksiköiden ohjaamiseen pelissä menetelmää, joka perustuu pelin kohteiden, kuten omien yksiköiden, vihollisen yksiköiden ja pelialueen ulkoreunojen muodostamien potentiaalikenttien yhteisvaikutukseen. Näitä potentiaalikenttiä muodostavia funktioita optimoidaan käyttäen geneettistä ohjelmointia. Geneettinen ohjelmointi on evoluutiolaskentamenetelmä, jossa luodaan optimoituja tietokoneohjelmia testaamalla satunnaisesti luotujen tietokoneohjelmien suoriutumista käytännössä ja yhdistelemällä sitten parhaiden tietokoneohjelmien ominaisuuksia.

Tutkielman rakenne on seuraava: Luvussa 2, esitellään RTS-pelien genre sekä RTS-pelejä pelaava botti ongelmana tekoälyn näkökulmasta. Luvussa 3 käsitellään evoluutiolaskentaa keskittyen pääasiassa geneettiseen ohjelmointiin. Luvussa 4 esitellään potentiaalikentät menetelmänä RTS-pelissä mikromanageroivan botin toteuttamiseen. Luvussa 5 esitellään tutkimuksessa toteutetun botin toteutusta, käytettyjä StarCraft-skenaarioita sekä geneettisen ohjelmoinnin parametreja. Luvussa 6 analysoidaan saatuja tuloksia sekä verrataan niitä aiempaan tutkimukseen. Lopuksi luvussa 7, tehdään johtopäätökset saaduista tuloksista sekä ehdotetaan mahdollisia kohteita jatkotutkimukselle.

1.3 Tutkimusongelma

Tämän tutkielman tavoitteena on selvittää, voidaanko geneettisellä ohjelmoinnilla toteutetulla optimoinnilla saada aikaan aikaisempaa parempia tuloksia menetelmässä, jossa potentiaalikentät ohjaavat yksiköitä taisteluissa reaaliaikastrategiapelissä (RTS-pelissä). RTS-pelien tyypillisistä pelimekaniikoista tutkimuksen ulkopuolelle rajataan resurssien kerääminen, tukikohdan rakentaminen, tiedustelu ja yksiköiden tuottaminen. Näiden sijaan keskitytään taistelutilanteeseen, jossa kahdella pelaajalla on kummallakin hallittavanaan joukko yksiköitä ja tehtävänään tuhota vastapuolen yksiköt.

Ongelmaa tutkitaan toteuttamalla ohjelmisto, joka kommunikoi StarCraft-pelin kanssa kolmannen osapuolen peliin kehittämän ohjelmistorajapinnan kautta ja käyttää ehdotettua menetelmää omien yksiköidensä hallintaan. Vastustajana käytetään peliin sisäänrakennettua yk-

sinkertaista tekoälyä, ja tuloksia verrataan aikaisemmassa tutkimuksessa käytettyyn optimointimenetelmään.

1.4 Aiheen merkitys

Tutkimusongelma on merkittävä paremman tekoälyn kehittämiseksi RTS-peleihin. Menetelmän etuna voidaan pitää oppimisprosessin autonomisuutta. Kun oppimisympäristö on kerran pystytetty, voidaan tekoäly myös opettaa uudelleen esimerkiksi pelitasapainon muutosten jälkeen tarvitsematta enää uusia syötteitä kehittäjiltä. Menetelmällä voisi olla pelien lisäksi muitakin sovelluksia, sitä voitaisiin esimerkiksi käyttää itsenäisesti liikkuvien robottien navigointiin.

RTS-pelien tutkimus on tekoälytutkimukselle yleisestikin mielenkiintoinen alue. Niiden muuttuva ympäristö ja epätäydellinen informaatio sekä reaaliaikaisuus ovat haastavia ongelmia ratkaistavaksi. Vastaavia ongelmia on monissa muissakin ympäristöissä, joten niiden ratkaisuja RTS-pelien ympäristöissä voidaan mahdollisesti soveltaa myös muihin ympäristöihin.

1.5 Tutkimusmenetelmä

Tutkimus toteutettiin konstruktiivisena tutkimuksena. Tutkimuksessa kehitettiin botti, jonka toimintaa ohjaavat pelimaailman kohteiden ympärille luodut yksiköitä puoleensa vetävät tai poispäin työntävät potentiaalikentät. StarCraftiin luotiin skenaario, jossa haluttu kokoonpano yksiköitä taistelee. Lisäksi luotiin ohjelmisto, joka muodostaa botin käyttämät potentiaalikentät luovat funktiot ja muokkaa funktioita geneettisen ohjelmoinnin keinoin.

Kun ohjelmisto ja koejärjestelyt oli toteutettu, ja botin potentiaalikentät oli optimoitu, tutkittiin miten hyvin parhaat botit suoriutuivat mikromanagerointitehtävästä. Tämä tehtiin asettamalla ne vastakkain StarCraftin sisäänrakennetun yksinkertaisen tekoälyn kanssa ja mittaamalla botin suoritusta taistelun jälkeen hengissä olevien botin ja vastustajan yksiköiden määrällä ja jäljellä olevilla osumapisteillä. Botin suoritusta verrattiin sitten aiemmassa tutkimuksessa kehitetyn menetelmän suoriutumiseen vastaavassa tilanteessa.

2 RTS-pelit

RTS-pelit ovat peligenre, jossa pelitapahtumat etenevät reaaliajassa. Tyypillisessä RTS-pelissä pelaajat rakentavat tukikohtaa, keräävät resursseja, tuottavat yksiköitä ja taistelevat vastapelaajan yksiköitä vastaan. RTS-pelejä pelaavan botin kehittäminen on mielenkiintoinen ongelma tekoälytutkimukselle.

2.1 Taustaa

RTS-pelit ovat yksinkertaistettuja sotasimulaatioita (Buro 2003). Genren peleissä tyypillisesti kaksi pelaajaa tai useammasta pelaajasta koostuvat joukkueet kilpailevat tavoitteenaan päihittää vastapelaaja sotilaallisesti tai alueellisesti (Chan ym. 2007). Pelaajat aloittavat pelin hallinnassaan joukko yksiköitä tai rakennuksia, jotka pystyvät tuottamaan lisää erityyppisiä rakennuksia ja yksiköitä. Rakennusten ja yksiköiden tuottaminen maksaa resursseja (esimerkiksi puu, raha, mineraalit tai asuintila), joita pelaajan on ensin kerättävä tai rakennettava. Pelaaja muodostaa rakennuksistaan tukikohtia maastosta löytyvien resurssien läheisyyteen. Resurssien keräyksen optimointi mahdollisimman tehokkaaksi mahdollisimman nopeasti on oleellista voiton kannalta (Chan ym. 2007). Pelaajan on kuitenkin tasapainoteltava resurssien tuotannon maksimoinnin ja sotilaallisen voiman kasvattamisen välillä, jotta vastustaja ei tuhoa puolustuskyvytöntä pelaajaa alkupelissä. Yksiköt ja rakennukset muodostavat teknologiapuun, jossa tietyt rakennukset ja yksiköt vaativat toisen rakennuksen tai yksikön valmistamista, ennen kuin niiden tuottaminen on mahdollista (Dillon 2008).

RTS-pelien genresukulaisia ovat vuoropohjaiset strategiapelit ja reaaliaikaiset taktiikkapelit. Vuoropohjaisista strategiapeleistä RTS-pelit eroavat ajanhallinnan osalta: Vuoropohjaisissa peleissä pelaaja voi miettiä seuraavaa siirtoaan pidempään, eivätkä pelitapahtumat etene ennen kuin pelaaja tekee siirtonsa. RTS-peleissä taas pelitapahtumat etenevät reaaliaikaisesti pelaajasta riippumatta ja pelaaja voi antaa vastustajien toimista riippumatta milloin tahansa yksiköilleen komentoja (Aarseth, Smedstad ja Sunnanå 2003). Reaaliaikaisista taktiikkapeleistä taas puuttuu resurssienhallinta ja rakennusten ja yksiköiden tuottaminen, ja pelaajat käyttävät taistelun aikana ennalta määrättyä joukkoa yksiköitä.

Motivaatio RTS-pelien tekoälyn kehittämiseen on ollut vähäistä kaupallisella puolella. Tämä johtuu esimerkiksi siitä, että pelien kehittäjät luottavat mielenkiintoista vastustajaa hakevien pelaajien hakeutuvan pelaamaan toisia ihmisvastustajia vastaan internetissä (Buro 2003). Lisäksi kaupallisissa peleissä on helppoa tehdä tekoälystä haastavampi vastustaja huijaamalla: tekoälyvastustaja voi esimerkiksi kerätä resursseja ihmispelaajaa nopeammin ja nähdä Fog of Warin peitossa olevat alueet (Buro 2003). Jos tekoäly huijaa liikaa, saattaa se tuntua pelaajasta epäreilulta (Yildirim ja Stene 2008). Tämä on yksi kannuste kehittää tekoälyä paremmaksi myös kaupallisissa peleissä.

2.2 StarCraft

Tutkimusympäristöksi valikoitui Blizzard Entertainmentin vuonna 1998 julkaisema RTS-peli StarCraft. Kuten Synnaeve ja Bessiere (2011) asian ilmaisevat (suomennos kirjoittajan), "StarCraft on kanoninen RTS-peli, samaan tapaan kuin shakki on kanoninen lautapeli, koska se on ollut olemassa vuodesta 1998, myi 10 miljoonaa lisenssiä ja oli paras kilpailullinen RTS[-peli]." Kuvassa 1 on ruutukaappaus StarCraftista.

StarCraftissa on pelattavana kolme eri rotua: protossit, terranit ja zergit (Blizzard Entertainment 1998, s. 26–90). Eri rotujen yksiköt eroavat täysin toisistaan ulkoasunsa lisäksi myös pelimekaanisilta ominaisuuksiltaan. Yksiköiden lisäksi myös pelin ydinmekaniikoissa on eroja rotujen välillä. Esimerkiksi protossit ja terranit tuottavat yksiköitä tietyistä rakennuksistaan, mutta zergien yksiköt tuotetaan muodonvaihdoksella toukkaa muistuttavista larva-yksiköistä.

StarCraftille on kehitetty tekoälyohjelmointirajapinta nimeltä BWAPI ("BWAPI Documentation" 2015). Sen myötä pelin ympärille on kehittynyt aktiivinen RTS-pelien tekoälystä kiinnostuneiden harrastajien ja tutkijoiden yhteisö. Vuosittain peliin ohjelmoitujen bottien välillä käydään tällä hetkellä ainakin kaksi turnausta: AIIDE (Artificial Intelligence and Interactive Digital Entertainment) -konferenssin sponsoroima StarCraft AI Competition (Buro ja Churchill 2015) sekä opiskelijoille tarkoitettu Student StarCraft AI Tournament (Čertický ym. 2015). Suuren aktiivisen käyttäjäkunnan ansiosta BWAPI on hyvin dokumentoitu ja ongelmatilanteissa on mahdollista kysyä neuvoa. Mainittakoon, että BWAPI vaatii toimiakseen

StarCraftista version, joka on laajennettu StarCraft: Brood War -lisäosalla.

Kuvio 1: Ruutukaappaus StarCraft-pelistä.

2.3 StarCraftin oleelliset mekaniikat

Luvussa avataan tutkimuksen kannalta oleellisia pelimekaniikkoja StarCraft-pelissä.

Fog of War on strategiapeleissä yleinen mekanismi, joka estää pelaajaa näkemästä vastustajan yksiköitä ja rakennuksia, jos omia yksiköitä tai rakennuksia ei ole riittävän lähellä (Hagelbäck ja Johansson 2008a). Pelaajalta piilossa olevat alueet havainnollistetaan pelinäkymässä tummennettuina.

Yksikkö (engl. *unit*) tarkoittaa RTS-peleissä yksittäistä sotilasta, rakentajaa, ajoneuvoa tai muuta vastaavaa liikkuvaa peliobjektia.

Rakentaja tai työläinen (engl. *builder* tai *worker*) on yksikkö, joka kyke-

nee keräämään resursseja ja rakentamaan rakennuksia (Blizzard Entertainment 2015i). Rakentaja kykenee myös heikkotehoiseen hyökkäykseen (Blizzard Entertainment 2015c).

Rakennus on paikallaan pysyvä yksikkö. Eri rakennuksilla on erilaisia käyttötarkoituksia: Ne voivat esimerkiksi tuottaa yksiköitä, kehittää yksiköille päivityksiä, tuottaa huoltopisteitä tai hyökätä vihollisyksiköiden kimppuun (Blizzard Entertainment 1998, s. 85–88).

Huoltokapasiteetti (engl. *supply*). Yksiköt varaavat valmistuessaan osan pelaajan huoltokapasiteetista käyttöönsä (Blizzard Entertainment 1998, s. 13–16). Tuhoutuessaan yksiköt vapauttavat varaamansa huoltokapasiteetin taas uusien yksiköiden käyttöön. Huoltokapasiteettia lisätään huoltorakennuksia rakentamalla. Jos käyttämätöntä huoltokapasiteettia ei ole riittävästi, ei uusia yksiköitä voida valmistaa. Yksikön tarvitsema huoltokapasiteetti vaihtelee yksikkötyypeittäin (Blizzard Entertainment 2015f). Huoltokapasiteetin maksimiraja on useimmissa pelimuodoissa 200 huoltokapasiteettiyksikköä, mikä rajaa myös pelaajien rakennettavissa olevien yksiköiden määrää (Blizzard Entertainment 2015j).

Osumapisteet (engl. *hit points, HP*) kuvaavat yksikön terveydentilaa (Blizzard Entertainment 1998, s. 19). Yksikön vahingoittuessa sen osumapisteiden määrä vähenee. Jos terveyspisteet vähenevät nollaan, yksikkö tuhoutuu. Osumapisteiden maksimimäärä vaihtelee yksikkötyypistä riippuen. Osumapisteiden määrä ei vaikuta yksikön ominaisuuksiin kuten nopeuteen tai sen tekemään vahinkoon.

Lentäminen Osa yksiköistä on lentäviä. Tällöin maaston esteet eivät vaikuta niiden liikkumiseen. Joidenkin yksiköiden hyökkäyksiä ei voi kohdistaa lentäviin yksiköihin, toisten hyökkäyksiä taas ei voi kohdistaa maassa kulkeviin yksiköihin (Blizzard Entertainment 2015f).

Kyvyt (engl. *abilities*) ovat joillain yksiköillä ja rakennuksilla olevia toimintoja, jota täytyy aktivoida painikkeella tai pikanäppäimellä (Blizzard Entertainment 1998, s. 18). Esimerkiksi protossien High Templar -yksiköllä on kyky nimeltä Psionic Storm, joka vahingoittaa kyvyn vaikutusalueella olevia yksiköitä 4 sekunnin ajan.

Näkymättömyys (engl. *cloaking*). Osa yksiköistä on pysyvästi näkymättömiä, osa voi käyttää kykyään tullakseen näkymättömäksi tilapäisesti. Vastustajan yksiköt eivät voi kohdistaa kykyään tai hyökkäystään näkymättömään yksikköön (Blizzard Entertainment 2015e). Tietylle alueelle kohdistuvat kyvyt tosin vaikuttavat myös näkymättömiin yksiköihin. Jotkin yksiköt ja rakennukset paljastavat näköpiirissään olevat näkymättömät yksiköt, jolloin niihin voi kohdistaa kykyjä tai hyökkäyksiä normaalisti. On myös kykyjä, jotka paljastavat näkymättömät yksiköt tietyltä alueelta tilapäisesti. Tällainen on esimerkiksi terranien Command Center -rakennuksen ComSat Station -laajennuksen Scanner Sweep -kyky (Blizzard Entertainment 2015d).

Panssarointi (engl. *armor*) on yksikön ominaisuus, joka vähentää vihollisen hyökkäysten tekemää vahinkoa (Liquipedia 2015a).

Hyökkäys Yksiköillä voi olla lähitaistelu- tai ampumahyökkäys. Lähitaisteluhyökkäyksen hyökkäysetäisyys on huomattavasti am-

pumahyökkäystä lyhyempi. Yksittäinen hyökkäys vastaa yhtä lyöntiä lähitaisteluaseella tai ammusta ampuma-aseella. Yksikön hyökkäyksellä on tietty vahinko, joka määrittää, paljonko kohteen osumapisteet vähenevät jokaisesta kohteen vastaanottamasta osumasta. Kohteen panssarointi ja koko vaikuttavat myös hyökkäyksen kohteeseen tekemään todelliseen vahinkoon.

Hyökkäysetäisyys (engl. *range*) tai ampumaetäisyys määrittelee hyökkäyksen kohteen maksimietäisyyden hyökkääjästä. Joillain yksiköillä voi olla myös minimihyökkäysetäisyys, jota lähempänä olevaan yksikköön ne eivät voi kohdistaa hyökkäystä.

Jäähtymisaika (engl. *cooldown*) on aika, joka yksikön pitää odottaa hyökkäyksen jälkeen, ennen kuin se voi hyökätä uudelleen. Termi viittaa aseen ylikuumenemiseen, mutta mekaniikka koskee kaikkia yksiköitä ja aseita.

Osumatodennäköisyys Jos hyökkäys ei osu kohteeseensa, kohde ei vastaanota lainkaan vahinkoa. BWAPI Wikin (2015) mukaan ampuma-asetta käyttävän yksikön todennäköisyys osua kohteeseen, joka on samalla korkeudella hyökkääjän kanssa, on 99,609375 % (255/256) ja todennäköisyys osua hyökkääjää korkeammalla tai esimerkiksi puuston alla olevaan kohteeseen on 53,125 % (136/256). Lähitaisteluasetta käyttävä yksikkö osuu kohteeseensa aina.

BWAPI Wiki ei kerro, miten luvut on saatu, mutta helpoin tapa tutkia ilmiötä lienee rakentaa ympäristö, jossa osumatodennäköisyyttä voi kokeilla käytännössä. BWAPIn avulla olisi yksinkertaista rakentaa järjestely, jossa voidaan kerätä automati-

soidusti riittävän suuri määrä dataa, jotta osumatarkkuus voidaan määrittää melko tarkasti. Toinen vaihtoehto tutkia asiaa lienee StarCraftin binääritiedostojen takaisinmallinnus.

Blizzard Entertainmentin (2015a) oma StarCraft-aiheinen verkkosivusto väittää, että matalampi osumatodennäköisyys olisi 70 %, mutta BWAPIn luku vaikuttaa uskottavammalta, sillä samat luvut esiintyvät StarCraft-aiheisessa Liquipedia-Wikissä (2015a) ja vaikuttavat myös olevan lähellä GosuGamers-sivuston keskustelijoiden tekemiä käytännön havaintoja (GosuGamers 2009).

Yksikön koko Kohdeyksikön koko ja hyökkäävän yksikön aseen tyyppi vaikuttavat kohdeyksikön hyökkäyksestä saamaan vahinkoon (Blizzard Entertainment 2013). Jokaiselle yksikölle on määritelty tietty koko (pieni, keskisuuri tai suuri) ja jokaiselle aseelle tietty vahingon tyyppi (isku, plasma, normaali tai räjähdys). Joillain yksiköillä on kaksi asetyyppiä, yksi lentäviä yksiköitä vastaan ja toinen maayksiköitä vastaan (ks. esim. Blizzard Entertainment (2015f)). Nämä yksikön ja aseen ominaisuudet vaikuttavat hyökkäyksen tekemään vahinkoon taulukon 1 mukaisesti.

Aseen tyyppi ks. yksikön koko.

Yksikön nopeus määrittelee, kuinka nopeasti kyseinen yksikkö liikkuu.

Yksikkötekoäly helpottaa pelaajan omien yksiköiden hallintaa. Pelaajan antaessa yksiköille komennon liikkua pisteestä A pisteeseen B tekoäly ohjaa yksiköt kulkemaan lyhyintä reittiä ja kiertämään esteet. Yksiköiden ollessa paikallaan ne hyökkäävät omatoi-

misesti lähelle tulevien vastustajan yksiköiden kimppuun ja myös seuraavat vastustajan yksiköitä jonkin matkaa, jos nämä pakenevat (Blizzard Entertainment 2015g). Pelaaja voi muuttaa tätä käyttäytymistä antamalla yksikölle komennon pysyä paikallaan.

Yksiköitä liikutettaessa on mahdollista valita kahdesta liikkumismuodosta. Liikkumiskäskyn saanut yksikkö ei hyökkää matkalla lähelle osuvien vastustajan yksiköiden kimppuun, ei edes, vaikka nämä hyökkäisivät liikkuvan yksikön kimppuun (Blizzard Entertainment 2015g). Sitä vastoin yhdistetyn hyökkäys- ja liikkumiskäskyn saanut yksikkö hyökkää matkan varrelle tulevien vihollisen yksiköiden kimppuun.

Myös rakentajayksiköt kykenevät hyökkäykseen, mutta niitä on erikseen käskettävä hyökkäämään. Paikallaan oleva rakentajayksikkö pakenee sen kimppuun hyökkäävää vihollista. Myös sotilasyksiköt toimivat näin ollessaan hyökkäyksen kohteena, jos ne eivät voi vastata hyökkäykseen (Blizzard Entertainment 2015g). Näin voi tapahtua esimerkiksi, jos mutalisk-tyyppinen yksikkö kohdistaa hyökkäyksen vulture-tyyppiseen yksikköön. Mutalisk on lentävä yksikkötyyppi, ja vultureilla on ainoastaan maayksiköihin kohdistettavissa oleva hyökkäys.

Taulukko 1: Erityyppisen vahingon vaikutus erikokoisiin yksiköihin (Blizzard Entertainment 2013).

Yksikön koko	**Isku/Plasma**	**Normaali**	**Räjähdys**
Pieni	100 %	100 %	50 %
Keskisuuri	50 %	100 %	75 %
Suuri	25 %	100 %	100 %

2.4 StarCraftin yksiköitä

StarCraftissa on useita kymmeniä erilaisia rakennuksia ja yksiköitä. Työssä mainittujen yksiköiden merkittävimpiä ominaisuuksia on esitelty taulukossa 2.

Goliath, hydralisk ja dragoon ovat eri rotujen melko samantapaisia yksiköitä. Ne liikkuvat maassa ja kykenevät ampumaan sekä maassa että ilmassa liikkuvia yksiköitä. Vulture on näitä yksiköitä selvästi nopeampi, mutta se kykenee ampumaan vain maassa liikkuvia yksiköitä. Vulture kykenee myös jättämään maahan viholliselle näkymättömiä miinoja, jotka räjähtävät vihollisyksiköiden tullessa riittävän lähelle (Blizzard Entertainment 2015h). Mutalisk on puolestaan lentävä, nopea yksikkö, joka kykenee ampumaan sekä maassa että ilmassa liikkuvia yksiköitä. Mutaliskin hyökkäys kimpoaa osuttuaan lähellä toisiaan olevien vihollisen yksiköiden välillä kahdesti ensimmäisen osuman jälkeen tehden siten vahinkoa kolmeen yksikköön kerralla (Blizzard Entertainment 2015b).

Drone on zergien ja SCV terraneiden rakentajayksikkö. Rakentajayksiköiden päätehtävä on kerätä resursseja ja rakentaa rakennuksia, mutta ne kykenevät myös heikkotehoiseen hyökkäykseen. Pelin alussa pelaajalla on tukikohtansa päärakennuksen lisäksi vain rakentajayksiköitä.

Taulukko 2: Joidenkin StarCraftin yksikkötyyppien ominaisuuksia (“BWAPI Wiki: Unit Types in BWAPI” 2015).

Muuttuja	Goliath	Hydralisk	Dragoon	Vulture	Mutalisk	Drone	SCV
Kuva							
Rotu	Terran	Zerg	Protoss	Terran	Zerg	Zerg	Terran
Liikkumistapa	Maa	Maa	Maa	Maa	Ilma	Maa	Maa
Max HP	125	80	100	80	120	40	60
Suojakilpi	0	0	80	0	0	0	0
Panssari	1	0	1	0	0	0	0
Yksikön koko	Suuri	Keskisuuri	Suuri	Keskisuuri	Pieni	Pieni	Pieni
Mineraalihinta	100	75	125	75	100	50	50
Kaasuhinta	50	25	50	0	100	0	0
Huoltokapasiteetin käyttö	2	1	2	2	2	1	2
Maksiminopeus	4,57	3,66	5	6,4	6,67	4,92	4,92
Vahinko	12	10	20	20	9	5	5
Vahingon tyyppi	Normaali	Räjähdys	Räjähdys	Isku	Normaali	Normaali	Normaali
Ampumaetäisyys	192	128	128	160	96	32	10
Cooldown	22	15	30	30	30	22	15
Vahinko ruudunpäivitystä kohden	0,545	0,667	0,667	0,667	0,3	0,227	0,333
Pisteet rakentamisesta	200	125	250	75	300	50	50

2.5 RTS-pelien luokittelu tekoälyongelmana

Russell ja Norvig (2010, s. 41–46) luokittelevat erilaiset ongelmat tekoälyn näkökulmasta. Luokitteluperusteita ovat informaation täydellisyys ja agenttien määrä sekä ympäristön kilpailullisuus, deterministisyys, stokastisuus, dynaamisuus, jatkuvuus ja tunnettuus.

Täydellisen informaation ympäristössä kaikki ongelmaan liittyvä informaatio on agentin havaittavissa. Jos näin ei ole, kyseessä on epätäydellisen informaation ympäristö. Useimmissa RTS-peleissä, niin myös StarCraftissa, informaatio on epätäydellistä. Tämä johtuu Fog of Wariksi kutsutusta pelimekaniikasta.

Ongelmat voidaan jakaa yhden agentin ja monen agentin ympäristöihin, joista jälkimmäiset voidaan vielä jakaa kilpailullisiin ympäristöihin ja yhteistyöympäristöihin. RTS-pelit ovat monen agentin ympäristöjä, sillä niissä pelaaja pelaa joko tekoälyn tai ihmisen ohjaamaa vastustajaa vastaan. Lisäksi kyseessä on kilpailullinen ympäristö.

Deterministisessä ympäristössä ympäristön seuraava tila riippuu täysin agentin toiminnasta ja ympäristön nykyisestä tilasta, kun taas epädeterministisessä ympäristössä toiminnan seurauksiin liittyy satunnaisuutta. Stokastinen ympäristö on sellainen epädeterministinen ympäristö, jossa toiminnan seurausten todennäköisyydet ovat tiedossa. Deterministisyys eri RTS-peleissä vaihtelee. StarCraft 2:ssa esimerkiksi on hyvin vähän merkittäviä satunnaiselementtejä. Alkuperäisessä StarCraftissa yksiköiden osumatodennäköisyys on merkittävä satunnaiselementti. Warcraft III -pelissä puolestaan yksiköiden tekemän vahingon määrä on aina satunnainen minimi- ja maksimivahingon välillä. Jos jokin RTS-peli ei ole täysin deterministinen, on se tavallisesti stokastinen, eli toimintojen mahdollisten tulosten todennäköisyys on tiedossa.

Staattinen ympäristö ei muutu, kun agentti pohtii seuraavaa toimenpidettään, dynaamisessa ympäristössä näin voi tapahtua. RTS-pelit ovat dynaamisia ympäristöjä, sillä vastustajan toiminta muuttaa ympäristöä jatkuvasti botin pohtiessa seuraavaa toimenpidettään.

Epäjatkuvuus ja jatkuvuus voivat liittyä ympäristön tiloihin, agentin havaintoihin ja agentin toimintoihin. Esimerkiksi shakissa ympäristöllä on rajallinen määrä tiloja ja rajallisia ovat myös shakkia pelaavan agentin havainnot ja toiminnot. RTS-pelin mahdolliset tilat, pelistä

tehtävät havainnot ja pelaajan tekemät toiminnot ovat periaatteessa epäjatkuvia, koska yksiköiden sijainti voidaan määritellä yhden pikselin välein ja pelitila päivittyy vain jokaisen ruudunpäivityksen yhteydessä. Peli saattaa tosin sisäisesti säilyttää tietoa yksikön sijainnista myös tarkemmalla tasolla. Käytännössä mahdollisten tilojen joukko on kuitenkin niin suuri, että RTS-peliä voidaan pitää pelin tilan, pelistä tehtävien havaintojen ja pelaajan tekemien toimintojen suhteen jatkuvana.

Ympäristö voi olla tunnettu tai vieras sen mukaan, tunteeko agentti ympäristössä pätevät säännöt, eli esimerkiksi toimenpiteiden vaikutukset. RTS-pelin säännöt ovat yleensä tiedossa melko tarkkaan, esimerkiksi yksiköiden osumapisteet ja niiden yhdellä iskulla tekemä vahinko ovat tiedossa, joten ympäristöä voidaan pitää tunnettuna. Pelimoottorin ohjelmakoodia ei kaupallisessa pelissä tavallisesti ole saatavilla, joten kaikkien pelimekaniikan yksityiskohtien selvittäminen saattaa olla vaikeaa.

2.6 Mikromanagerointi

RTS-pelien pelaaminen voidaan jakaa kahteen tasoon, joita kutsutaan mikro- ja makromanageroinniksi. Mikromanageroinnilla tarkoitetaan yksittäisten yksiköiden hallintaa taistelutilanteessa, kun taas makromanagerointi tarkoittaa strategisemman tason päätöksiä kuten rakennettavien yksiköiden valintaa tai tukikohdan laajennuksen tai hyökkäyksen ajankohdan valintaa (Szczepański ja Aamodt 2008). Ihmispelaajien pelatessa mikromanagerointia tapahtuu enemmän pelin alkuvaiheessa, kun kontrolloitavia yksiköitä ja hallittavia tehtäviä ei ole vielä niin paljon (Weber, Mateas ja Jhala 2011). Furtak ja Buro (2010) osoittivat, että mikromanagerointi on monimutkaisuudeltaan PSPACE-täydellinen ongelma, mikä tarkoittaa, että se voidaan ratkaista Turingin koneella polynomisessa tilassa, ja kaikki muut ongelmat, jotka voidaan ratkaista polynomisessa tilassa voidaan muuntaa täksi ongelmaksi (Sipser 1997).

Koska tämä tutkimus keskittyy mikromanagerointiongelman ratkaisuun, on syytä käsitellä tarkemmin StarCraftissa käytettyjä mikromanagerointitaktiikoita (Liquipedia 2015b). Monet kuvatuista taktiikoista soveltuvat myös useisiin muihin RTS-peleihin.

2.6.1 Tiedustelu

Fog of war -mekaniikan vuoksi pelaaja ei näe, mitä vastustaja tekee, ellei hänellä ole omia yksiköitä lähettyvillä. Jotta pelaaja voisi reagoida vastustajan toimiin, on hänen tehtävä tiedustelua. Alkupelissä vastustajan tukikohtaa tiedustellaan tavallisesti yhdellä rakentajayksiköllä, jota on kontrolloitava vastustajan tukikohdassa, jotta vastustaja ei tuhoa yksikköä omilla rakentajillaan. Keski- ja loppupelissä tiedustelua voidaan suorittaa erilaisilla yksiköillä ja kyvyillä. Esimerkiksi terraneiden Command Center -rakennukseen voidaan rakentaa ComSat station -laajennus, jonka kyky mahdollistaa pienen alueen näkemisen mistä tahansa kohdasta kartalta 10 sekunnin ajan. Tiedustelussa tärkeää informaatiota on vastustajan yksiköiden sijainti ja tyyppi sekä tieto siitä, mitä yksiköitä vastustaja on aikeissa tulevaisuudessa tuottaa. Tämä voidaan päätellä esimerkiksi vastustajan tukikohdan rakennusten perusteella.

2.6.2 Sijoittuminen

Taistelun sijainnin valinta vaikuttaa sen lopputulokseen. Solan läpi tuleva joukko yksiköitä on altavastaajan asemassa, jos solan suulla odottaa joukko vihollisen yksiköitä. Myös ylempänä olevat yksiköt ovat etulyöntiasemassa, koska alempaa tulevien yksiköiden hyökkäyksistä suurempi osa menee ohitse tekemättä vahinkoa. Omien yksiköiden sijoittelulla suhteessa toisiinsa on merkitystä, sillä halvemmat yksiköt kannattaa sijoittaa etulinjaan suojaamaan kalliimpia, tehokkaampia yksiköitä, jotka kannattaa sijoittaa taemmaksi. Omia yksiköitä kannattaa levittää leveäksi rintamaksi suhteessa vastustajan yksiköihin, jotta mahdollisimman moni niistä pääsee ampumaetäisyydelle nopeasti, eivätkä yksiköt ole toistensa tiellä. Koukkaus, jossa vastustajan kimppuun hyökätään edestä ja takaa, estää vastustajan pakenemisen, ja lisäksi omat yksiköt ovat tällöin vähemmän toistensa tiellä. Jos osa käytettävistä yksiköistä on lähitaisteluyksiköitä, kannattaa näillä ensin liikkua hyökkäämättä osittain vastustajan ohitse ja antaa hyökkäyskomento vasta, kun yksiköt ovat ympäröineet vastustajan joukot. Tällöin useampi yksikkö pääsee hyökkäämään vastustajan yksiköiden kimppuun, koska vähemmän omia yksiköitä on tiellä.

2.6.3 Häirintä

Häirinnällä tarkoitetaan vastustajan tukikohdan selustaan hyökkäämistä pienellä joukolla yksiköitä. Jos vastustajan yksiköt ovat muualla eikä hänellä ole puolustusrakennuksia, voidaan vastustajan rakentajia ja rakennuksia onnistua tuhoamaan. Samalla saadaan tietoa vastustajan rakentamista rakennuksista.

2.6.4 Taistelu

Yksi taistelussa käytetty taktiikka on hyökkäysten keskittäminen yhteen yksikköön kerrallaan. StarCraftissa ja yleensä myös muissa RTS-peleissä yksikön vahingoittuminen ei vähennä sen tekemää vahinkoa, joten vastustajan yksiköiden tuhoaminen mahdollisimman nopeasti yksi kerrallaan on tehokkaampaa kuin omien yksiköiden tulivoiman jakaminen useampaan vastustajan yksikköön. Toisaalta omia yksiköitä kannattaa pyrkiä pitämään hengissä mahdollisimman pitkään. Tanssimiseksi kutsutussa taktiikassa yksiköt, joiden elämäpisteet alkavat vähentyä, vedetään hetkeksi kauemmaksi taistelusta, jolloin vastustajan yksiköt vaihtavat kohdetta toiseen yksikköön. Kun vastustajan yksiköt ovat vaihtaneet kohdetta, voidaan vahingoittunut yksikkö tuoda takaisin taisteluun.

Nopeat yksiköt, joilla on pitkän kantaman ampumahyökkäys, voivat käyttää hitaampia, lyhyemmän kantaman hyökkäyksen tai lähitaisteluhyökkäyksen omaavia yksiköitä vastaan taktiikkaa, josta käytetään englanninkielisiä termejä *kiting* tai *hit-and-run*. Taktiikassa yksikkö perääntyy lyhyemmän hyökkäysetäisyyden omaavan yksikön hyökkäyksen kantomatkan ulkopuolelle aseen jäähtymisaikana. Nimitys kiting tulee tavasta, jolla omat ja vastustajan yksiköt liikkuvat samaan suuntaan saman välimatkan päässä toisistaan ikään kuin leijaa lennätettäessä.

2.7 Aiempia tekoälymenetelmiä mikromanagerointiin

Mikromanagerointiin on aikaisemmin sovellettu esimerkiksi Bayesin verkkoja, tapauspohjaista päättelyä, reaktiivista suunnittelua ja hakua.

2.7.1 Bayesin verkot

Synnaeve ja Bessiere (2011) käyttivät mikromanagerointiin Bayesin verkkoihin perustuvaa ratkaisua, jossa eri muuttujien perusteella laskettiin todennäköisyydet yksikön liikkumiselle tiettyyn suuntaan. He kokeilivat kahta menetelmää, joista toisessa liikkumissuunnaksi valittiin joka kerta suurimman todennäköisyyden saanut suunta (BAIPB, Bayesian AI picking best), toisessa taas suunta valittiin todennäköisyysjakauman mukaan (BAIS, Bayesian AI sampling).

Synnaeve ja Bessiere (2011) päihittivät menetelmällään StarCraftin sisäänrakennetun tekoälyn peilatuissa skenaarioissa (kummallakin pelaajalla käytettävissään samat yksiköt) 12 ja 36 dragoonin kokoonpanoilla yli 90 %:ssa otteluista. BAIS-menetelmä toimi maassa liikkuville yksiköille paremmin erityisesti enemmän yksiköitä sisältäneessä skenaariossa. Synnaeve ja Bessiere (2011) arvelivat tämän johtuvan siitä, että usemmasta liikkumissuunnasta todennäköisyysjakauman mukaan suuntansa valinneet yksiköt törmäilivät toisiinsa vähemmän kuin BAIPB-menetelmän vain parhaaseen suuntaan liikkuneet yksiköt. Tätä hypoteesia tukee se, että lentävillä yksiköillä menetelmien suoriutumisessa ei ollut samankaltaista eroa. Tutkimusympäristönä toimineessa StarCraftissa lentävät yksiköt eivät törmää toisiinsa, vaan ne voivat sijaita keskenään päällekkäin.

2.7.2 Tapauspohjainen päättely

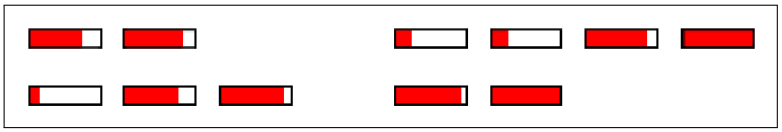
Szczepański ja Aamodt (2008) tutkivat tapauspohjaisen päättelyn (engl. Case Based Reasoning, CBR) soveltumista mikromanagrointiin Warcraft III -pelin karttaeditorin skriptausmahdollisuutta käyttäen. Tapauskohtainen päättely perustuu joukkoon ratkaistuja tapauksia, joihin pelitilannetta verrataan. Pelitilannetta lähinnä olevaa tapausta ratkaistujen tapausten joukossa hyödyntämällä päätellään, kuinka menetellä. Szczepańskin ja Aamodtin (2008) käyttämässä menetelmässä ratkaisut eri tapausten tietokantoihin annettiin asiantuntijatiedon perusteella siten, että heidän kehittämänsä botti pelasi pelin sisäänrakennettua tekoälyvastustajaa vastaan asiantuntijan tarkkaillessa. Jos botti joutui valitsemaan ratkaisuksi tilanteeseen soveltumattoman tapauksen, koska parempaa tapausta ei ollut saatavilla, asiantuntija pysäytti pelin ja lisäsi uuden tapauksen ja siihen sopivan käyttäytymismallin tietokantaan. Menetel-

mässä pelitilanteen samankaltaisuutta tietokannassa oleviin tapauksiin verrattiin laskemalla jokaiselle yksikölle euklidinen etäisyys muuttujille: 1) jäljellä olevat osumapisteet, 2) jäljellä oleva mana[1] ja 3) yksikön sijainti (x- ja y-koordinaatit).

CBR vaikuttaa mielenkiintoiselta tavalta toteuttaa mikromanagerointi, mutta tapauksien lisääminen tietokantaan käsin lienee työlästä, jos tavoitteena olisi kattaa riittävästi tapauksia täyden pelin pelaamiseen tietyn skenaarion sijaan. Weber ja Ontanón (2010) kokeilivatkin CBR:ää rakentaen tapaustietokannan automaattisesti StarCraftin ottelutallenteiden pohjalta. Kyseisessä tutkimuksessa kuitenkin käytettiin vain neljän ottelutallenteen pohjalta muodostettuja tapauksia, eikä StarCraftin sisäänrakennettua tekoälyä onnistuttu voittamaan. Tutkimuksesta ei selvinnyt, käytettiinkö menetelmää mikromanagerointiin, mutta ainakin väitöskirjassaan Weber (2012) käytti CBR:ää ainoastaan tuotettavien yksiköiden ja rakennusten valintaan ja vastustajan tuotannon ennakointiin.

Toinen tapa rakentaa tapaustietokanta automaattisesti on käyttää jotain koneoppimismenetelmää. Gunnerud (2009) käytti CBR:ää ja vahvistusoppimista (engl. Reinforcement Learning, RL) mikromanagerointiin. Hän havaitsi Szczepańskin ja Aamodtin (2008) menetelmässä ongelman tavassa, jolla he vertailivat tapaustensa samankaltaisuutta. Szczepański ja Aamodt nimittäin järjestivät yksiköt jäljellä olevien osumapisteiden mukaan ja vertasivat yksiköitä sitten järjestyssijoittain toisiinsa. Kuvio 2 havainnollistaa Gunnerudin havaitsemaa ongelmaa. Vasemmalla olevassa tapauksessa alarivin ensimmäistä yksikköä verrataan ylärivin ensimmäiseen yksikköön ja alarivin viimeistä yksikköä ei verrata mihinkään yksikköön, koska tapauksissa on eri määrä yksiköitä. Szczepańskin ja Aamodtin (2008) vertailutapaa käytettäessä nämä tapaukset vaikuttavat erilaisilta, vaikka todellisuudessa ne ovat lähellä toisiaan. Vastaavasti oikeanpuoleisessa tapauksessa ylärivin kahta ensimmäistä yksikköä verrataan vastaaviin alarivin yksiköihin ja kahta jälkimmäistä yksikköä ei verrata mihinkään yksikköön. Paras tulos saataisiin etsimällä mahdollisimman samankaltaiset yksiköt ja vertaamalla näitä toisiinsa. Tämä oli yksi Szczepańskin ja Aamodtinkin vaihtoehdoista, mutta he päätyivät käyttämään järjestettyjen yksiköiden vertailua, koska se oli heidän alustavassa testauksessaan päihittänyt Warcraft III:n sisäänrakennetun tekoälyn pienemmällä tapausmäärällä.

1. Mana on taikaenergiaa, joita osa Warcraft III:n yksiköistä tarvitsee käyttääkseen kykyjään tai loitsujaan (toinen nimitys kyvyille fantasiateemaisessa Warcraft III:ssa).



Kuvio 2: Ongelma pelitilanteiden vastaavuuden vertailussa havainnollistettuna jäljellä olevia osumapisteitä kuvaavilla palkeilla (Gunnerud 2009).

Gunnerud päätyi vertaamaan tapauksia toisiinsa ainoastaan yksikkötyyppien määrien perusteella. Mitä enemmän kahdessa tapauksessa on samoja yksikkötyyppejä samassa armeijassa, sitä lähempänä tapaukset ovat toisiaan. Jos kummassakin tapauksessa pelaajan ja vastustajan armeijoilla on sama koostumus, kyseessä on sama tapaus. Gunnerudin menetelmässä tapausten ratkaisut olivat prioriteettilistoja hyökkäyksen kohteeksi valittavan vihollisyksikön yksikkötyypille. Gunnerud yritti ensin menetelmää, jossa prioriteetit olisivat olleet lukuarvoja välillä 0–100. Hänen mielestään tässä menetelmässä ongelmaksi muodostui uusien ratkaisujen muodostaminen, joten hän päätyi käyttämään systeemiä, jossa yksi yksikkötyyppi sai prioriteetin 1 ja muut 0. Gunnerudin botin yksiköt hyökkäsivät vain prioriteetin 1 tyyppisten yksiköiden kimppuun. Kun samantyyppisiä yksiköitä oli useampi, hyökkäyksen kohde valittiin käyttäen hyödyllisyysfunktiota, jonka parametreja olivat yksikön jäljellä olevat osumapisteet, manapisteet ja etäisyys hyökkäävästä yksiköstä. Botti rakensi tapaustietokantansa pelaamalla Gunnerudin ohjelmoimia yksinkertaisia heuristiikkoja noudattavia botteja vastaan. Uusia ratkaisuja kokeiltiin sattumanvaraisesti priorisoimalla sellaista yksikkötyyppiä, jota ei oltu aikaisemmin vastaavassa tapauksessa priorisoitu. Kaikkien yhdessä ottelussa käytettyjen ratkaisujen kelpoisuutta arvioitiin ottelun lopputuloksen perusteella.

Gunnerud testasi ratkaisuaan yksinkertaisia tekoälyjään vastaan ja onnistui menetelmällään päihittämään nämä testivastustajat. Suurimpia ongelmia aiheutti tapauksen vaihtuminen esimerkiksi oman yksikön tuhoutumisen johdosta, jolloin yksiköt saattoivat vaihtaa kohdetta jo melkein tuhotusta yksiköstä toiseen yksikköön. Lisäksi todellisuudessa hyvä ratkaisu saattoi saada huonon kelpoisuuden pitkäksi aikaa, jos muut samassa ottelussa käytetyt ratkaisut sattuivat olemaan huonoja. Gunnerud käytti tutkimusympäristönä itse toteuttamaansa pelimoottoria.

Gunnerudin menetelmässä, varsinkin ensimmäisessä ratkaisussa, jossa prioriteetit olivat liukuvia arvoja, voisi olla potentiaalia. Uusien ratkaisujen luominen voisi onnistua geneettisellä

algoritmilla. Ratkaisujen kelpoisuus kannattaisi mahdollisesti laskea esimerkiksi 10 ottelun keskiarvon perusteella, jolloin hyvä ratkaisu ei niin suurella todennäköisyydellä saisi huonoa kelpoisuutta vain muiden ottelussa käytettyjen tapausten perusteella. Tällainen menetelmä muistuttaisi yhteisevoluutioita (engl. *co-evolution*) eri ratkaisujen kesken.

2.7.3 Reaktiivinen suunnittelu

Weber ym. (2010) kiinnittävät huomiota siihen, että mikromanagerointia käsittelevissä tutkimuksissa harvoin käsitellään mikronamageroinnin integroimista koko peliä pelaavaan bottiin. He rakensivat täyttä StarCraft-peliä pelaavan EISBotin käyttäen reaktiivista suunnittelua (engl. *reactive planning*).

Menetelmässä reaktiiviset suunnitelmat määritellään ABL-kielellä (A Behavior Language), josta ne käännetään Java-ohjelmakoodiksi. Suunnitelmat muodostetaan puurakenteeksi, jonka solmut ovat käyttäytymismalleja (engl. *behaviour*), tavoitteita (engl. *goal*) ja toimintoja (engl. *action*).

Menetelmä mahdollistaa tavoitteiden ja käyttäytymismallien yhdistämisen strategisella, taktisella ja yksittäisen yksikön tasolla. Yksiköllä voi olla oma mikromanagerointikäyttäytymisensä, mutta jos se liitetään suurempaan joukkoon, siirtyy kontrolli taktiikkamanagerille. Taktiikkamanageri koordinoi useamman yksikön joukon yhteistyötä. Järjestelmä mahdollistaa myös eri tasojen managerien välisen kommunikoinnin. Esimerkiksi mikromanagerointimanageri voi muuttaa käytöstään toiselta managerilta saamansa viestin perusteella. Itse mikromanageroinnin toteutusta Weber ym. eivät käsittele.

2.7.4 Haku

Churchill ja Buro (2012) simuloivat StarCraftin taisteluita ja etsivät optimaalisia komentoja yksiköille mikromanagerointitilanteeseen simulaatioiden perusteella. Simuloitujen pelien kulku ei vastaa täysin StarCraftin pelimoottorilla pelattujen pelien kulkua, koska StarCraftin pelimoottori on epädeterminstinen, eikä kaikkia sen yksityiskohtia tunneta. Artikkelissa saavutettiin menetelmällä lupaavia tuloksia, mutta sitä ei oltu vielä yhdistetty täyttä peliä pelaavaan bottiin.

Menetelmää käsitellään tarkemmin artikkelissa Churchill, Saffidine ja Buro (2012). Churchill, Saffidine ja Buro (2012) kehittivät StarCraftin simulointia hyödyntäen algoritmin, jota he kutsuvat siirtojen keston huomioivaksi alfa-beta-hauksi (Alpha-Beta Considering Durations, ABCD). Koska RTS-pelin reaaliaikaisuus rajoittaa käytettävissä olevaa aikaa, on haun syvyys rajoitettu, ja lehtisolmut arvioidaan suorittamalla läpipeluu deterministisellä skriptillä lehtisolmun tilanteesta. ABCD on saanut inspiraationsa Go-pelissä hyviä tuloksia tuottaneesta Monte Carlo -puuhausta (engl. *Monte Carlo tree search*, *MTSC*) (Coulom 2007). ABCD eroaa MCTS:stä esimerkiksi lehtisolmujen arvioinnissa, joka tehdään deterministisellä läpipeluulla. MCTS:ssä lehtisolmut arvioidaan pelaamalla ne loppuun useita kertoja satunnaisilla siirroilla.

3 Evoluutiolaskenta

Tässä luvussa käsitellään evoluutiolaskentaan liittyvää teoriaa. Luku 3.1 käsittelee evoluutiolaskennan taustaa. Luku 3.2 keskittyy tässä tutkielmassa käytettyyn evoluutiolaskennan osa-alueeseen, geneettiseen ohjelmointiin.

3.1 Taustaa

Evoluutiolaskenta on tekoälyyn kuuluva tieteenala, joka käsittelee evoluutiota matkivia optimointialgoritmeja (Back, Hammel ja Schwefel 1997). Evoluutiolaskenta voidaan jakaa kolmeen itsenäisesti kehittyneeseen haaraan, jotka ovat geneettinen algoritmi, evoluutio-ohjelmointi ja evoluutiostrategiat. Back, Hammel ja Schwefel (1997) esittelevät runsaan määrän kirjallisuutta, jossa kyseisiä evoluutiolaskennan haaroja on alun perin kehitetty. Evoluutiolaskennan tavoitteena on välttää hakualgoritmin jumiutumista ongelmien paikallisesti parhaisiin ratkaisukohtiin, jotta globaalisti paras ratkaisu löydettäisiin. Evoluutiolaskenta on intuitiivinen ja yksinkertainen ongelmanratkaisutapa, mutta se on yleisen tason menetelmä ja sitä pitää muokata kulloiseenkin ongelmaan sopivaksi. Russell ja Norvig (2010, s. 129) huomauttavat, että on epäselvää, johtuuko evoluutiolaskennan suosio sen tehokkuudesta vai evoluution mallintamisen tuomasta estetiikasta.

Tässä tutkimuksessa käytettävä menetelmä, geneettinen ohjelmointi on evoluutiolaskennan osa-alue, joka pohjautuu lähtökohdiltaan John Hollandin (1975) geneettiseen algoritmiin. Geneettisessä algoritmissa eri ratkaisuvaihtoehdot esitetään merkkijonoina. Yhtä ratkaisuvaihtoehtoa kutsutaan kromosomiksi tai yksilöksi. Yksittäisiä merkkejä kutsutaan geeneiksi. Koza (1992, s. 17–51) käyttää esimerkkinä ongelmaa, jossa tavoitteena on optimoida hampurilaisravintolaketjun strategiaa. Strategia on koodattu kolmella bitillä taulukon 3 mukaisesti.

Geneettinen algoritmi aloitetaan muodostamalla joukko yksilöitä satunnaisesti. Kulloinkin käsiteltävänä olevaa yksilöjoukkoa kutsutaan populaatioksi. Evoluution aikana luodaan vanhan populaation pohjalta aina uusi populaatio. Näitä peräkkäisiä populaatioita kutsutaan sukupolviksi. Ravintolaketjun tapauksessa ensimmäinen sukupolvi voisi koostua esimerkiksi yksilöistä 001, 100, 101 ja 110. Yksilö 001 tarkoittaisi siis hampurilaisravintolaa, jossa on

Taulukko 3: Hampurilaisravintolaketjun strategia koodattuna binäärijonoksi (Koza 1992).

Arvo	Hinta	Juoma	Palvelun nopeus
1	Halpa	Coca-Cola	Nopea
0	Kallis	Viini	Hidas

Taulukko 4: Ensimmäinen sukupolvi hampurilaisravintoloiden strategioita, yksilöiden kelpoisuudet ja niiden osuudet kokonaiskelpoisuudesta.

Yksilö	Kelpoisuus	Osuus kokonaisuudesta
001	1	0,06
100	4	0,25
101	5	0,31
110	6	0,38

kalliit hinnat, juomana tarjotaan viiniä ja palvelu on nopeaa. Seuraavaksi arvioidaan kukin yksilö kelpoisuusfunktiolla (engl. *fitness function*). Ravintoloiden tapauksessa voidaan harjoittaa liiketoimintaa vaikkapa kuukauden verran ja laskea saatu voitto. Tässä esimerkissä kelpoisuus on yksinkertaisesti kunkin strategian koodauksen arvo binärilukuna. Kelpoisuudet on esitetty taulukossa 4.

Seuraavaksi muodostetaan uusi sukupolvi risteyttämällä ensimmäisen sukupolven yksilöitä. Risteytystä varten nykyisestä sukupolvesta valitaan kaksi yksilöä satunnaisesti rulettivalinnalla, jossa kunkin yksilön todennäköisyys tulla valituksi on suoraan verrannollinen yksilön kelpoisuuden osuuteen sukupolven yhteenlasketusta kelpoisuudesta. Valituksi voivat tulla esimerkiksi yksilöt 101 ja 110. Seuraavaksi valitaan risteytyskohta satunnaisesti joko ensimmäisen ja toisen tai toisen ja kolmannen bitin välistä. Valitaan tässä esimerkissä mielenkiinnon vuoksi toisen ja kolmannen bitin väli (ensimmäisessä tapauksessa jälkeläiset ovat samat kuin vanhemmat). Risteytys toteutetaan katkaisemalla bittijonot risteytyskohdasta ja yhdistämällä toisen vanhemman alkuosa toisen loppuosaan ja päinvastoin. Tällöin saadaan jälkeläisiksi yksilöt 111 ja 100. Risteytystä jatketaan, kunnes uudessa sukupolvessa on riittävästi yksilöitä. Toiset uudet yksilöt 100 ja 110 voidaan saada risteyttämällä yksilöt 100 ja 110 toisen ja kolmannen bitin kohdalta. Tässä risteytyksessä ei syntynyt uusia strategioita.

Risteytyksen jälkeen populaatiolle suoritetaan vielä mutaatio-operaatio. Mutaatiossa jokai-

Taulukko 5: Toinen sukupolvi hampurilaisravintoloiden strategioita kelpoisuuksineen.

Yksilö	Kelpoisuus
100	4
110	6
110	6
111	7

sella yksilöllä on pieni mahdollisuus tulla valituksi mutaatioon. Myös useampi yksilö voidaan valita. Mikäli yksilö valitaan mutaatioon, vaihdetaan yksi satunnainen bitti yksilöstä ja käännetään bitti päinvastoin ($0 \rightarrow 1$ tai $1 \rightarrow 0$). Jos mutaatioon valitaan yksilö 100 ja käännettäväksi bitiksi yksilön toinen bitti, saadaan uudeksi yksilöksi 110. Mutaatio-operaation tehtävänä on säilyttää yksilöiden erilaisuus, joka valinnan seurauksena vähenee. Toisen sukupolven populaatio on taulukossa 5. Populaation keskimääräinen kelpoisuus nousi yhden sukupolven aikana 16:sta 23:een ja optimaalinen ratkaisu 111 löydettiin.

Geneettinen algoritmi on kuvattu pseudokoodina algoritmissa 1. Algoritmia on hieman yksinkertaistettu siten, että risteytyksestä tulee vain yksi jälkeläinen.

Geneettisen algoritmin toimintaan liittyy käsite skeema. Skeema tarkoittaa mallia, johon osa mahdollisista merkkijonoista kuuluu (Holland 1975, s. 66–74). Yksi skeema voisi olla esimerkiksi 1**, johon siis kuuluvat kaikki sellaiset kolmen bitin jonot, joissa ensimmäinen bitti on ykkönen. Merkkijonon alussa olevan ykkösen merkitystä voidaan arvioida laskemalla keskiarvo niiden yksilöiden kelpoisuuksista, jotka kuuluvat kyseiseen skeemaan. Esimerkin toisessa sukupolvessa kyseisen skeeman keskimääräinen kelpoisuus on 5,75 (kaikki yksilöt kuuluvat tähän skeemaan).

Geneettisessä algoritmissa käytetty rulettivalinta hyödyntää (engl. *exploit*) tunnettuja hyviä skeemoja ja etsii (engl. *explore*) uusia skeemoja optimaalisessa suhteessa (Koza 1992, s. 43–47) maksimoiden prosessin aikana saatavan voiton. Risteytys ja mutaatio heikentävät hieman tätä optimia, mutta ne ovat välttämättömiä uusien skeemojen löytämiseksi.

Geneettisestä algoritmista on myös useita variaatioita. Esimerkiksi yksilöiden valinta voidaan tehdä rulettivalinnan sijaan turnausvalinnalla, jossa valintatilanteessa valitaan satunnaisesti n kappaletta yksilöitä ja näistä korkeimman kelpoisuuden omaava yksilö valitaan.

Algoritmi 1 Geneettinen algoritmi (Russell ja Norvig 2010, s. 129)

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
    inputs population, a set of individuals
           FITNESS-FN, a function that measures the fitness of an individual

    repeat
        new_population ← empty set
        for i ← 1 to SIZE(population) do
            x ← RANDOM-SELECTION(population, FITNESS-FN)
            y ← RANDOM-SELECTION(population, FITNESS-FN)
            child ← REPRODUCE(x, y)
            if (small random probability) then
                child ← MUTATE(child)
            end if
            add child to new_population
        end for
        population ← new_population
    until some individual is fit enough, or enough time has elapsed
    return the best individual in population, according to FITNESS-FN
end function
```

```
function REPRODUCE(x, y) returns an individual
    inputs x, y, parent individuals

    n ← LENGTH(x); c ← random number from 1 to n
    return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
end function
```

Sukupolvittain etenevän prosessin sijaan voidaan käyttää vakaan tilan (engl. *steady-state*) menetelmää. Tässä menetelmässä täysin uuden populaation luomisen sijaan korvataan olemassaolevasta populaatiosta vain yksi yksilö kerrallaan (Whitley ym. 1989).

3.2 Geneettinen ohjelmointi

1970- ja 1980-luvuilla alettiin etsiä keinoja kehittää Hollandin geneettistä algoritmia, jotta voitaisiin vakiomittaisten merkkijonojen lisäksi käsitellä monimutkaisempia rakenteita (Koza 1992, s. 64). Vuonna 1992 julkaistussa teoksessaan *Genetic programming: on the programming of computers by means of natural selection* (Koza 1992) John Koza esitteli geneettisenä ohjelmointina tunnettuun mallin, joka mahdollisti tietokoneohjelmien luomisen geneettistä algoritmia hyödyntäen.

3.2.1 Esitysmuoto

Koza (1992, s. 68–72) päätyi käyttämään Lisp-ohjelmointikielen symbolisia lausekkeita (S-lausekkeita) menetelmässään tietokoneohjelmien esitysmuotona. Lispin S-lausekkeet koostuvat sulkeiden sisällä esitetyistä listoista, joiden ensimmäistä alkiota käsitellään funktiona ja muita alkioita argumentteina. Tällaista matemaattista merkintätapaa kutsutaan puolalaiseksi notaatioksi. Listauksessa 3.1 on esimerkki yksinkertaisesta S-lausekkeesta, joka laskee yhteen $2 + 1$.

Listaus 3.1: Yksinkertainen Lisp-ohjelmointikielen symbolinen lauseke

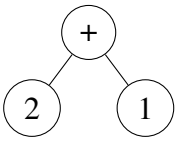
```
(+ 2 1)
```

Kozalla on useita perusteluja Lispin valintaan ohjelmien esitystavaksi. Olennaisimpia syitä ovat seuraavat:

- Ohjelmia voidaan käsitellä datana, jolloin niitä voidaan muokata helposti ohjelmallisesti ja sen jälkeen suorittaa ne (Koza 1992, s. 71).
- Useimmat ohjelmointikielet jäsentävät ohjelman käännösvaiheessa puurakenteeksi (Koza 1992, s. 71). Lisp-ohjelma on käytännössä valmiiksi jäsennetty näin, joten puurakennetta päästään helposti muokkaamaan. Listauksen 3.1 S-lauseke on esitetty puu-

rakenteena kuviossa 3.

- Tietokoneohjelmien koko ja muoto vaihtelevat (Koza 1992, s. 64), ja Lisp tukee erityisen hyvin tällaisia dynaamisia rakenteita (Koza 1992, s. 72).



Kuvio 3: Listauksen 3.1 S-lauseke puurakenteena.

S-lausekkeita vastaavien puurakenteiden sisäiset solmut koostuvat funktioista ja lehtisolmut terminaaleista eli vakioista ja muuttujista. Kullakin funktiolla on ariteetti, joka kertoo, kuinka monta argumenttia ja siten lapsisolmua funktio vaatii (Koza 1992, s.80). Esimerkiksi yhteenlaskuun tarvitaan 2 argumenttia, joten sen ariteetti on 2.

3.2.2 Alustus

Geneettisen ohjelmoinnin prosessi alkaa geneettisen algoritmin tavoin ensimmäisen sukupolven populaation satunnaisella luonnilla (Koza 1992, s. 73–74). S-lausekkeiden satunnaiseen luontiin on useita eri tapoja. Kaksi perusmenetelmää ovat täyttömenetelmä (engl. *full*) ja kasvatusmenetelmä (engl. *grow*), joissa kummassakin valitaan satunnaisesti solmu kerrallaan solmuun tuleva funktio tai terminaali (Koza 1992, s. 92–93). Täyttömenetelmä etenee valitsemalla solmut pelkkien funktioiden joukosta, kunnes puun maksimisyvyys saavutetaan, jolloin solmut valitaan terminaalien joukosta. Näin luodun puun kaikki lehtisolmut ovat samalla syvyydellä. Kasvatusmenetelmässä voidaan mihin tahansa solmuun valita joko funktio tai terminaali. Näin luodut puut ovat vaihtelevan muotoisia. Puun maksimisyvyydellä kuitenkin valitaan pelkistä terminaaleista, jotta puusta ei tule haluttua suurempaa.

Kozan (1992, s. 93) mukaan alustusmenetelmä nimeltä “ramped half-and-half” toimii parhaiten laajassa joukossa ongelmia. Menetelmässä puolet puista luodaan täyttömenetelmällä ja puolet kasvatusmenetelmällä eri maksimisyvyyksillä kahdesta haluttuun maksimisyvyyteen. Halutun maksimisyvyyden ollessa esimerkiksi 6 luodaan puista 20 % maksimisyvyydellä 2, 20 % maksimisyvyydellä 3 ja niin edelleen, aina kuuteen asti.

3.2.3 Kelpoisuus

Yksilöiden kelpoisuus selvitetään ajamalla luotu ohjelma, tyypillisesti useita kertoja eri syötteillä (Koza 1992, s. 74). Kelpoisuus voi olla esimerkiksi virhe tavoitefunktioon verrattaessa tai peliä pelattaessa saadut pisteet. Kelpoisuuden selvittämiseen voidaan käyttää myös yhteisevoluutiota (engl. *co-evolution*), jossa esimerkiksi pelin ollessa kyseessä yksilöitä testataan laittamalla ne pelaamaan toisiaan vastaan (Koza 1992, s. 94). Jokainen yksilö pelaa joko kaikkia muita populaation yksilöitä vastaan tai satunnaista otosta vastaan.

Käsittelemättömästä kelpoisuusluvusta (engl. *raw fitness*), eli esimerkiksi saaduista pisteistä tai yhteenlasketusta virheestä eri testitapauksissa, voidaan johtaa myös *standardisoitu kelpoisuus*, *muokattu kelpoisuus* ja *normalisoitu kelpoisuus* (engl. *standardized fitness*, *adjusted fitness* ja *normalized fitness*) (Koza 1992, s. 95).

Standardisoidussa kelpoisuudessa käsittelemätöntä kelpoisuutta muokataan siten, että pienempi arvo on parempi ja jos mahdollista, paras kelpoisuus on 0 (Koza 1992, s. 96). Jos käsittelemättömässä kelpoisuudessa suurempi arvo on parempi ja maksimikelpoisuus on tiedossa, saadaan yksilön i standardisoitu kelpoisuus $s(i,t)$ ajan hetkellä t kaavalla

$$s(i,t) = r_{max} - r(i,t),$$

jossa r_{max} on maksimikelpoisuus ja $r(i,t)$ käsittelemätön kelpoisuus.

Muokatussa kelpoisuudessa arvot ovat välillä 0–1 ja suurempi arvo on parempi (Koza 1992, s. 97). Muokattu kelpoisuus $a(i,t)$ lasketaan kaavalla

$$a(i,t) = \frac{1}{1+s(i,t)}.$$

Muokattu kelpoisuus korostaa pieniä eroja siinä vaiheessa, kun yksilöiden kelpoisuus lähestyy optimikelpoisuutta.

Normalisoitua kelpoisuutta käytetään rulettivalinnassa määrittämään todennäköisyys tietyn yksilön valinnalle lisääntymisoperaatioon (Koza 1992, s. 97–98). Normalisoitu kelpoisuus vaihtelee välillä 0–1 ja suurempi luku on parempi. Lisäksi yhden sukupolven normalisoitujen

kelpoisuuksien summa on 1. Normalisoitu kelpoisuus $n(i,t)$ lasketaan kaavalla

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)},$$

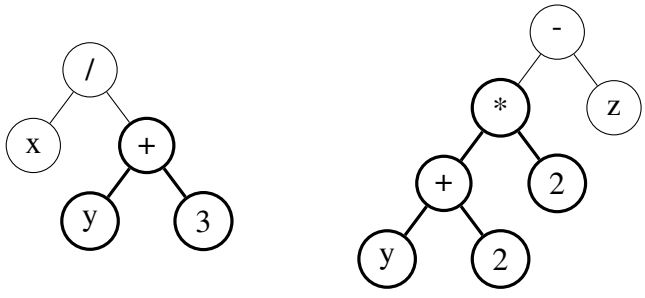
jossa M on populaation koko.

3.2.4 Operaatiot

Merkittävimmät operaatiot geneettisessä ohjelmoinnissa ovat kopiointi (engl. *reproduction*) ja risteytys (Koza 1992, s. 99). Seuraavan sukupolven populaatio luodaan käyttämällä näitä kahta operaatiota ennalta määrätyssä suhteessa. Esimerkiksi 10 % yksilöistä luodaan kopioimalla ja 90 % risteytyksellä.

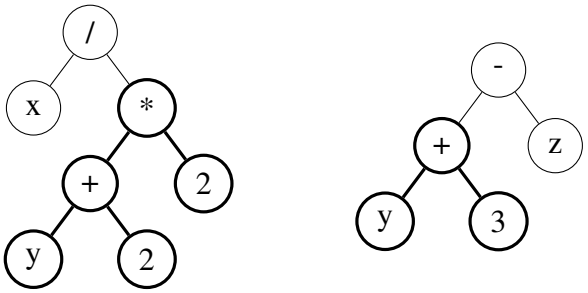
Kopioinnissa yksilö kopioidaan sellaisenaan seuraavaan sukupolveen (Koza 1992, s. 99). Kopioitavan yksilön valinta tehdään siten, että jokaisella yksilöllä on normalisoidun kelpoisuutensa suuruinen todennäköisyys tulla valituksi.

Risteytyksessä valitaan ensin kaksi vanhempiyksilöä samoin kuin kopioinnissa (Koza 1992, s. 101–102). Tämän jälkeen kummastakin yksilöstä valitaan satunnaisesti yksi solmu (kuvio 4). Uudet yksilöt luodaan siten, että valituista solmuista alkavat alipuut vaihtavat paikkaa (kuvio 5).



Kuvio 4: Risteytettävät alipuut valittu.

Merkkijonoja käsittelevässä geneettisessä algoritmissa käytetään yleensä risteytyksen lisäksi mutaatiota (Koza 1992, s. 105–107). Geneettisessä ohjelmoinnissa mutaatiolle on vähemmän tarvetta, koska terminaaleilla ja funktioilla ei ole sidottua paikkaa ja lisäksi eri terminaaleja ja funktioita on yleensä paljon vähemmän kuin perinteisessä geneettisessä algoritmissa



Kuvio 5: Uudet yksilöt.

geenejä. On epätodennäköistä, että jokin terminaali tai funktio kuolisi sukupuuttoon. Yksi syy mutaation käyttöön perinteisessä geneettisessä algoritmissa on myös monimuotoisuuden säilyttäminen. Tähänkään mutaatiota ei geneettisessä ohjelmoinnissa välttämättä tarvita. Toisin kuin merkkijonoja käytettäessä, geneettisessä ohjelmoinnissa esimerkiksi kahden samanlaisen yksilön tullessa valituksi risteytykseen jälkeläiset eivät todennäköisesti ole identtiset vanhempiensa kanssa, koska on todennäköistä, että kummastakin vanhemmasta risteytyskohdaksi valitaan eri solmu.

Koza (1992, s.106) kuitenkin esittelee myös mutaatio-operaation geneettiseen ohjelmointiin, vaikkei sitä omissa esimerkeissään käytäkään. Operaatio toimii siten, että valitaan S-lausekkeesta yksi solmu ja korvataan tästä solmusta alkava alipuu uudella, satunnaisesti luodulla alipuulla.

Olisiko mutaatio kuitenkin hyödyllinen operaatio silloin, jos geneettisessä ohjelmoinnissa käytetään satunnaisvakioita? Tällöin ei oltaisi täysin riippuvaisia ensimmäisen populaation luonnissa syntyneistä satunnaisvakioista, vaan niitä voisi mutaation kautta tulla lisää myös evoluution aikana. Toki uusia vakioita voi syntyä myös suorittamalla laskutoimituksia alustuksen yhteydessä luoduilla satunnaisvakioilla.

Evett ja Fernandez (1998) itse asiassa tutkivat geneettisessä ohjelmoinnissa esiintyvää numeeristen vakioiden löytämisen ongelmaa ja ehdottivat ratkaisuksi numeeriseksi mutaatioksi kutsumaansa prosessia. Prosessissa osalle jokaisen sukupolven parhaista yksilöistä suoritetaan numeeriseksi mutaatioksi kutsuttu operaatio, jossa kaikkia yksilön satunnaisvakioita muutetaan hieman. Muutosjakauma on suoraan verrannollinen yksilön standardisoituun kelpoisuuteen, eli mitä parempi kelpoisuus on, sitä pienempiä muutoksia vakioihin tehdään.

3.2.5 Valinta

Geneettisiä operaatioita varten on suoritettava operoitavien yksilöiden valinta. Mahdollisia valintamenetelmiä ovat esimerkiksi rulettivalinta ja turnausvalinta. Rulettimenetelmässä kunkin yksilön todennäköisyys tulla valituksi on suoraan verrannollinen yksilön kelpoisuuteen. Turnausvalinnassa valitaan ensin satunnaisesti n kappaletta yksilöitä ja näistä operoitavaksi valitaan yksilö, jonka kelpoisuus on paras turnausvalintaan osallistuvista yksilöistä (Miller ja Goldberg 1995). Turnausvalinnassa on se etu, että tilanteessa, jossa yksilöiden väliset kelpoisuuserot ovat pieniä, on lievästi parempien yksilöiden valinta todennäköisempää kuin rulettivalinnassa, joten kehitys ei kelpoisuuserojen kaventuessa hidastu yhtä paljon kuin rulettivalintaa käytettäessä. Valintapainetta on myös helppo säädellä turnauksen kokoa vaihtamalla (Miller ja Goldberg 1995).

Kelpoisuusjärjestykseen perustuvassa valinnassa (engl. *rank selection*), yksilöt asetetaan järjestykseen kelpoisuuden mukaan ja tietyllä järjestysnumerolla olevat yksilöt valitaan lisääntymään ennalta määrätyn kappalemäärän mukaan. Genitor-valintamenetelmässä taas sukupolvien sijaan edetään korvaamalla yksi kerrallaan populaation sillä hetkellä huonoin yksilö uudella (Goldberg ja Deb 1991).

3.2.6 Lopetus

Geneettisen ohjelmoinnin prosessi lopetetaan, kun sukupolvien maksimimäärä on saavutettu tai onnistumisehto täyttyy (Koza 1992, s. 113). Onnistumisehto on usein maksimikelpoisuuden saavuttaminen, mutta voidaan myös tyytyä ratkaisuun, joka on riittävän lähellä tätä. Joissain tapauksissa maksimikelpoisuutta ei ole olemassa tai ei voida tunnistaa. Tällöin evoluutioprosessi yksinkertaisesti lopetetaan sukupolvien maksimimäärän saavuttamiseen.

Prosessin tulos on koko prosessin aikana havaittu paras yksilö (Koza 1992, s. 113). Jos elitismiä (joukon parhaiden yksilöiden kopioimista uuteen sukupolveen) ei käytetä, eikä evoluutioprosessi päättynyt optimiyksilön löytämiseen, vaan maksimisukupolvimäärän täyttymiseen, ei paras yksilö välttämättä esiinny viimeisessä sukupolvessa.

4 Keinotekoiset potentiaalikentät

Tässä luvussa tarkastellaan keinotekoisten potentiaalikenttien teoreettista perustaa. Luku 4.1 kertoo potentiaalikenttien taustasta. Luku 4.2 kertoo potentiaalikenttien sovelluksista RTS-peleissä.

4.1 Taustaa

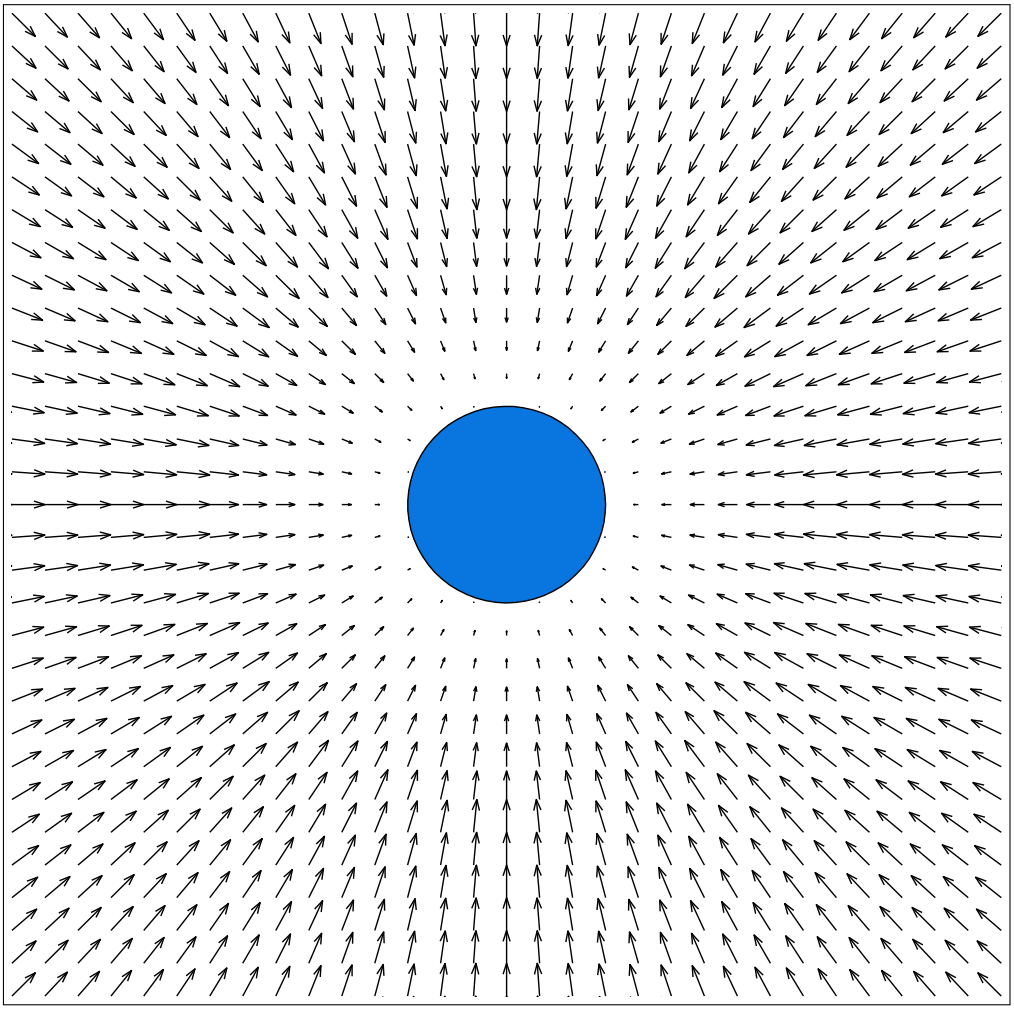
Keinotekoiset potentiaalikentät (engl. *artificial potential fields, APF*, tässä tutkielmassa myös pelkkä potentiaalikenttä) on kehitetty ratkaisemaan ongelma, jossa agentti valitsee reittiä tavoitteeseensa ympäristössä, jossa on esteitä. Menetelmän tavoitteena on siirtää optimaalisen reitin valinta korkean tason suunnitteluongelmasta matalan tason kontrollin vastuulle (Khatib 1986). Tämän menettelyn tavoitteena on parantaa järjestelmän suorituskykyä. Menetelmä on reaktiivinen, eli ympäristön havainnoista johdetaan suoraan suoritettava toiminta (Balch 1993). Näin ollen muistia säästyy, kun järjestelmän ei tarvitse ylläpitää sisäistä mallia ympäristöstään. Potentiaalikenttien suorituskyky mahdollistaa reaaliaikaisen toiminnan muuttuvassa ympäristössä (Balch 1993). Sen vahvuuksia ovat myös yksinkertaisuus sekä intuitiivinen tapa laskea reittiä haettaessa huomioitavien kohteiden yhteisvaikutus.

Termiä keinotekoiset potentiaalikentät käytti ensimmäisenä Khatib (1986). Khatib käytti APF:iä robottikäsivarren liikuttamiseen ympäristössä, jossa oli esteitä. Khatibin metodissa esteet aiheuttavat robottikäsivarren osiin virtuaalisen voimakentän, joka työntää robottikäsivartta poispäin esteistä. Käsivarren tarttujan kohde taas aiheuttaa puoleensa vetävän voiman. Robotin moottoreita ohjataan sitten tuottamaan nuo voimat todellisuudessa. Potentiaalikenttien avulla navigoivan agentin voi ajatella liikkuvan kuten varattu hiukkanen magneettikentässä tai marmorikuula kaltevalla alustalla (Goodrich 2002).

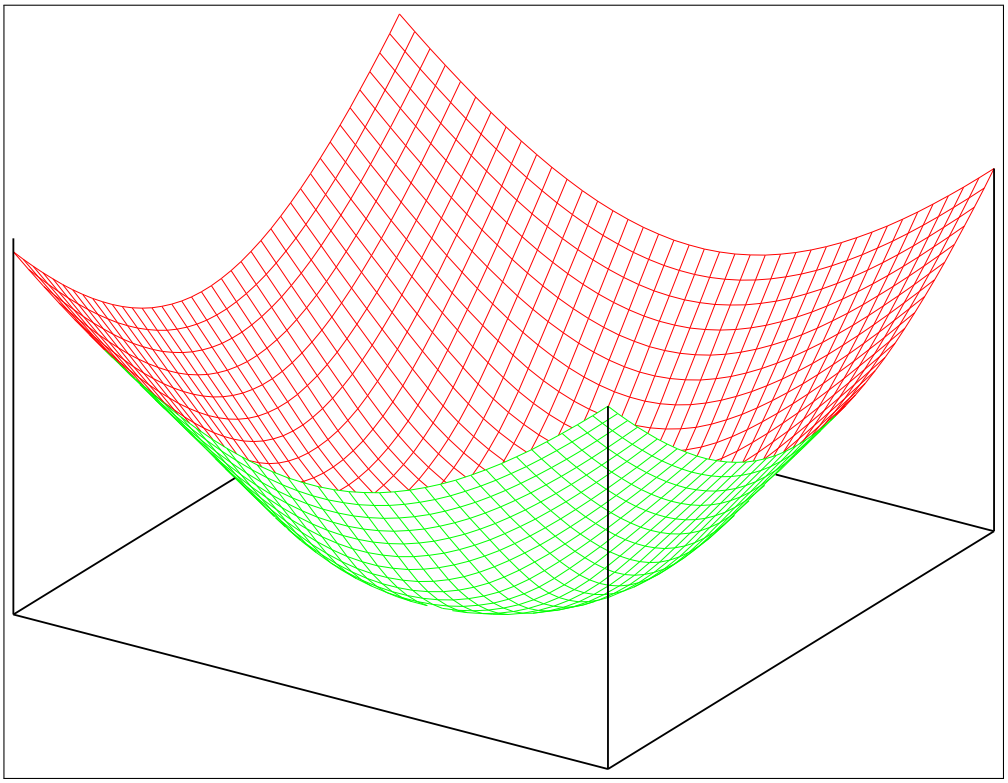
Arkin (1987) hyödynsi potentiaalikenttiä muistuttavaa tekniikkaa liikkuvan robotin ohjaamiseen. Arkin kutsui metodiaan motoriseksi skeemaksi (engl. *motor schema*). Arkinin inspiraationa oli robotiikan lisäksi psykologia ja aivotutkimus. Sana skeema tulee tässä yhteydessä psykologiasta tarkoittaen mielessä olevaa mallia ulkopuolisesta maailmasta. Arkin yhdisti eri kohteiden vaikutuksen summaamalla kohteiden aiheuttamat nopeusvektorit yhteen. Arkinin

menetelmässä lasketaan vain suunta ja nopeus, johon robotin tietyllä hetkellä tulisi liikkua. Koko potentiaalikenttää ei lasketa, jotta laskentatehoa säästyisi. Kuviossa 6 on esimerkki puoleensa vetävästä potentiaalikenttä. Jokainen nuoli vastaa robotin nopeusvektoria kyseisessä sijainnissa. Kuviossa 7 kyseinen potentiaalikenttä on visualisoitu kolmiulotteisesti.

Peleihin potentiaalikenttiä sovelsivat esimerkiksi Thurau, Bauckhage ja Sagerer (2004), jotka käyttivät potentiaalikenttiä kehittäessään Quake II -peliin bottia, jonka tarkoitus oli mukailla ihmispelaajien noudattamia liikkumisreittejä. Botin käyttämät potentiaalikentät muodostettiin ihmispelaajien liikkeitä seuraamalla. Käytetty potentiaalikenttä riippui myös botin tilasta: terveydentilasta, haarniskan vahvuudesta ja kerätyistä aseista.



Kuvio 6: Puoleensa vetävä potentiaalikenttä (Goodrich 2002).



Kuvio 7: Kuvion 6 potentiaalikenttä visualisoituna kolmiulotteisesti.

4.2 Keinotekoisten potentiaalikenttien aiemmat sovellukset mikromanagerointiin RTS-peleissä

Hagelbäck ja Johansson (2008c) kehittivät ORTS-pelimoottorilla (Open Real-Time Strategy) vuonna 2007 järjestettyyn turnaukseen botin, joka toimi monen agentin potentiaalikenttiä käyttäen. He käyttivät käsin valittuja parametreja potentiaalikentille. Ratkaisussa maaston esteet, omat yksiköt ja rakennukset sekä kartan satunnaisesti liikkuvat neutraalit yksiköt aiheuttavat poispäin työntävän kentän ja vastustajan yksiköt aiheuttavat puoleensa vetävän kentän, joka on kaikkein voimakkaimmillaan omien yksiköiden maksimiampumaetäisyyden päässä vastustajan yksiköistä.

Turnauksessa, johon botti kehitettiin, kilpailtiin mikromanageroinnissa kahdessa skenaariossa. Toisessa kummallakin pelaajalla oli käytössään 50 tankkia ja 5 tukikohtaa, ja tehtävänä oli tuhota vastustajan tukikohdat. Toisessa skenaariossa kummallakin pelaajalla oli 50 yksikköä jalkaväkeä, ja tehtävänä oli tuhota vastustajan yksiköt. Kummassakaan skenaariossa pelaajat eivät voineet rakentaa rakennuksia tai tuottaa lisää yksiköitä.

Hagelbäck ja Johansson (2008c) käyttivät potentiaalikenttien lisäksi hyökkäyksen koordinoijaa, joka toimii pitäen kirjaa siitä, mitkä omista yksiköistä kykenevät ampumaan mitäkin vastustajan yksikköä. Vastustajan yksiköistä priorisoidaan niitä, joiden osumapisteet ovat matalalla. Koordinoija keskittää hyökkäykset niin, että jos jokin vastustajan yksikkö voidaan tuhota, määrätään niin monta omaa yksikköä ampumaan sitä kuin tuhoamiseen tarvitaan. Tulivoimaa ei kuitenkaan tuhlata ampumalla yksikköä enempää kuin on tarpeen, vaan loput yksiköt määrätään ampumaan jotain toista yksikköä.

Kyseinen botti ei menestynyt tutkimuksessa raportoidussa turnauksessa kovin hyvin sijoittuen toisessa osakilpailussa sijalle 6/9 ja toisessa sijalle 6/8. Hagelbäck ja Johansson (2008c) raportoivat seuraavanlaisista ongelmista:

- Maaston esteiden aiheuttamiin lokaaleihin potentiaalimaksimeihin jumiutuminen.
- Yksiköiden hajautuminen laajalle alueelle. Vastustajat saivat ne poimittua yksi kerrallaan.
- Vastustajan yksiköiden puoleensa vetävistä potentiaalikentistä aiheutunut vastustajan yksikköryhmittymän vahvoihin kohtiin hyökkääminen.

Hagelbäck ja Johansson (2008c) pitivät potentiaalikenttiä kuitenkin kiinnostavana vaihtoehtona reaaliaikastrategiapelin tekoälyn toteuttamiseen. Hyvinä puolina he pitivät tehokkuutta ja joustavuutta.

Seuraavassa artikkelissaan Hagelbäck ja Johansson (2008b) yrittivät ratkaista botissaan olleita ongelmia. Aiemmassa botissa yksi ongelma oli yksiköiden jääminen jumiin maaston esteisiin ja muihin esteisiin, kuten omiin rakennuksiin. Tähän auttoi potentiaalikenttien resoluution tarkentaminen 8x8 pikselin tarkkuudesta 1x1 pikselin tarkkuudelle. Aiemmassa versiossaan botistaan Hagelbäck ja Johansson (2008c) optimoivat toteutustaan jakamalla pelialueen 8x8 pikselin kokoisiin alueisiin, joista kullekin laskettiin oma potentiaaliarvonsa. 1x1 pikselin resoluution mahdollistaakseen he laskivat nyt koko potentiaalikentän sijaan potentiaalin vain 8 pisteessä jokaisen yksikön ympärillä (Hagelbäck ja Johansson 2008b). Aiemmassa versiossa eri objektien potentiaalikentät oli myös laskettu valmiiksi ja ne haettiin taulukosta. Muistin säästämiseksi tästä ratkaisusta luovuttiin, ja potentiaalit laskettiin nyt silloin kun niitä tarvittiin. Maaston staattinen potentiaalikenttä laskettiin kuitenkin edelleen

valmiiksi.

Toinen ongelma oli yksiköiden altistuminen vastustajan tulitukselle omien aseidensa jäähtymisajan aikana. Tähän heikkouteen luotiin sellainen parannus, että yksikön aseen jäähtyessä vihollisen yksiköt aiheuttavatkin poispäin työntävän potentiaalikentän. Tämä mahdollistaa vastustajan yksiköitä nopeampien yksiköiden perääntymisen vastustajan ulottumattomiin, kunnes ne voivat hyökätä uudelleen.

Kolmas ongelma olivat potentiaalikentän lokaalit minimikohdat, joihin yksiköt jäivät ajoittain jumiin. Tämä ongelma ratkaistiin siten, että yksiköt jättävät viimeiseen 20:een sijaintipaikkaansa poispäin työntävän potentiaalikentän. Tämän ratkaisun lokaaleihin minimikohtiin jumiutumiseen on kuvannut tarkemmin Balch (1993).

Neljäs ongelma oli se, että vastustajan yksiköt vetivät botin yksiköitä eniten puoleensa keskelle omaa suurinta ryhmittymäänsä, koska kaikkien vastustajan yksiköiden potentiaalit summattiin yhteen. Tämä ei tietenkään ole hyvä sijainti yksiköille. Ongelma ratkaistiin siten, että vastustajan yksiköiden yhdistetty potentiaalikenttä muodostetaan niin, että potentiaali lasketaan ainoastaan sen yksikön perusteella, jonka potentiaali on kyseisessä pisteessä vahvin. Näin omat yksiköt saatiin sijoittumaan maksimiampumaetäisyytensä päähän lähimmästä vihollisesta.

Näillä parannuksilla Hagelbäck ja Johansson (2008b) saivat bottinsa voittamaan yli 90 % otteluista vuoden 2007 ORTS-kilpailun botteja vastaan. Botti voitti myös vuoden 2008 turnauksen.

Vuonna 2009 järjestetyssä ORTS-turnauksessa pelattava peli oli monipuolisempi kuin aiempana kahtena vuonna (Hagelbäck ja Johansson 2009). Pelaajat aloittivat pelin hallussaan komentokeskus ja muutamia rakentajia. Heidän täytyi kerätä resursseja, rakentaa tukikohta, tuottaa yksiköitä ja tuhota vastustajan tukikohta. Kaupallisiin RTS-peleihin verrattuna asetelma oli edelleen yksinkertaistettu, sillä rakentajien lisäksi ainoat yksiköt olivat jalkaväki ja tankit, joita tuotettiin kumpaakin omasta rakennuksestaan. Hagelbäck ja Johansson kehittivät bottiaan eteenpäin ja osallistuivat jälleen turnaukseen.

Tähän bottiin lisättiin muutamia uusia potentiaalikenttiä: Resurssit vetävät rakentajia puo-

leensa. Fog of Warin peittämästä kartan osasta valitaan tiedusteltavaksi yksi paikka. Valintaan vaikuttaa se, miten kauan sitten paikka on ollut viimeksi näkyvissä, ja se, miten lähellä tiedustelijayksikköä paikka on. Tämä korkeimman prioriteetin saanut sijainti vetää tiedustelijayksikköä puoleensa. Myös rakennusten sijoittelussa hyödynnettiin potentiaalikenttiä.

Uusien potentiaalikenttien lisäksi botin arkkitehtuuria muutettiin lisäämällä uusia kokonaisuutta hallitsevia agentteja. CommanderInChief tekee korkean tason suunnitelman sille, mitä rakennuksia ja yksiköitä tuotetaan, ja missä järjestyksessä. CommanderInField toteuttaa CommanderInChiefin tekemiä suunnitelmia asettaen yksiköille tavoitteita. GlobalMapAgent pitää kirjaa vihollisen rakennuksista, jotka on havaittu, mutta jotka ovat Fog of Warin peitossa. (ORTS-palvelin ei lähetä näiden rakennusten sijaintia asiakasohjelmalle.) AttackCoordinator päättää tulituksen kohteet samoin kuin aiemmissakin toteutuksissa (Hagelbäck ja Johansson 2008c, 2008b). Lisäksi SpatialPlanner valitsee sijainnit rakennuksille.

Pelkkien potentiaalikenttien käyttäminen yksiköiden liikkeiden kontrollointiin tuotti Hagelbäckille ja Johanssonille vaikeuksia. Tietyissä tilanteissa yksiköt jumiutuivat lokaaleihin minimikohtiin. Näin saattoi käydä esimerkiksi tukikohdassa, jonka ainoa uloskäynti rajatulta alueelta oli tukikohdan pohjoispuolella, mutta vihollisen tukikohta sijaitsi tukikohdan eteläpuolella. Tällöin vihollisen yksiköt vetivät omia yksiköitä puoleensa niin paljon, että omat yksiköt eivät päässeet omasta tukikohdasta ulos. Tämän takia Hagelbäck ja Johansson päätyivät käyttämään A*-reitinhakua, kunnes omat yksiköt olivat riittävän lähellä vastustajan yksiköitä. A*-reitinhausta huolehti AttackCoordinator-agentti.

Liu, Louis ja Nicolescu (2013) yhdistelivät potentiaalikenttiä ja vaikutusaluekarttoja (engl. *influence maps*, *IM*) siten, että vihollisen yksiköiden sijainnin perusteella laskettiin vaikutusaluekartta vastustajan yksiköiden heikoimman kohdan löytämiseksi. Potentiaalikentät ohjasivat yksiköiden liikkumista siten, että omat ja vihollisen yksiköt työnsivät yksiköitä pois päin ja IM:illä havaittu heikko kohta veti yksiköitä puoleensa.

Potentiaalikenttien sijaan myös potentiaalivirtauksia on käytetty yksiköiden kontrollointiin tiedustelu- ja taistelutilanteissa (Nguyen, Wang ja Thawonmas 2013), (Nguyen, Nguyen ja Thawonmas 2013). Potentiaalivirtausten etuna on, että niissä ei ole lokaaleja minimikohtia, joihin yksiköt jäisivät jumiin, eivätkä ne ohjaa yksiköitä esteitä päin (Nguyen, Wang

ja Thawonmas 2013). Niiden avulla on myös mahdollista toteuttaa helposti pyörteitä, jotka ohjaavat yksiköitä kulkemaan kohdetta kiertäen. Tekniikkaa on hyödynnetty Kiotossa sijaitsevan Ritsumeikan-yliopiston ICE- laboratorion[1] kehittämässä ICEbot-botissa (Nguyen, Nguyen ja Thawonmas 2013), (Nguyen, Wang ja Thawonmas 2013), joka voitti AIIDE:n StarCraft-tekoälyturnauksen vuonna 2014 ("StarCraft AI Competition Results" 2014).

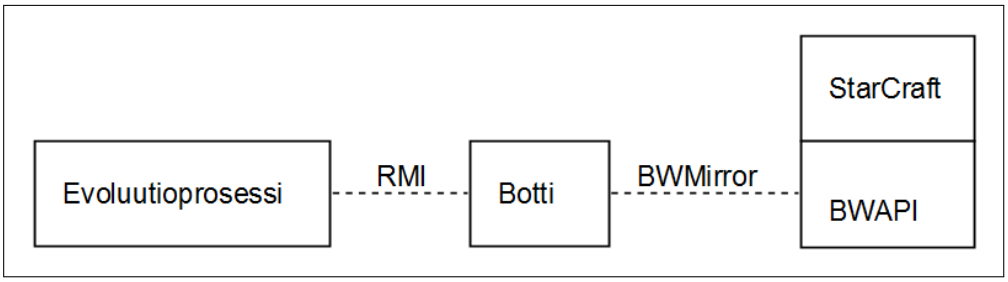
Sandberg (2011) käytti potentiaalikenttiä mikromanageroivan tekoälyn kehittämiseen. Hän käytti samantapaista menetelmää kuin Hagelbäck ja Johansson (2008a, 2008b, 2008c, 2009). Sandberg käytti geneettistä algoritmia Hagelbäckin ja Johanssonin tutkimuksissa käytetyn potentiaalikenttiin perustuvan ratkaisun parametrien optimointiin.

1. Intelligent Computer Entertainment

5 Toteutus

Työn tavoitteena on tutkia käytännössä geneettisen ohjelmoinnin suoriutumista keinotekoisia potentiaalikenttiä hyödyntävän mikromanagerointimenetelmän optimoinnissa. Tämän tutkimiseksi toteutettiin StarCraftia pelaava botti, joka hyödyntää mikromanagerointiin keinotekoisia potentiaalikenttiä. Lisäksi toteutettiin evoluutioprosessi, joka optimoi bottia ohjaavia potentiaalikenttiä geneettisellä ohjelmoinnilla.

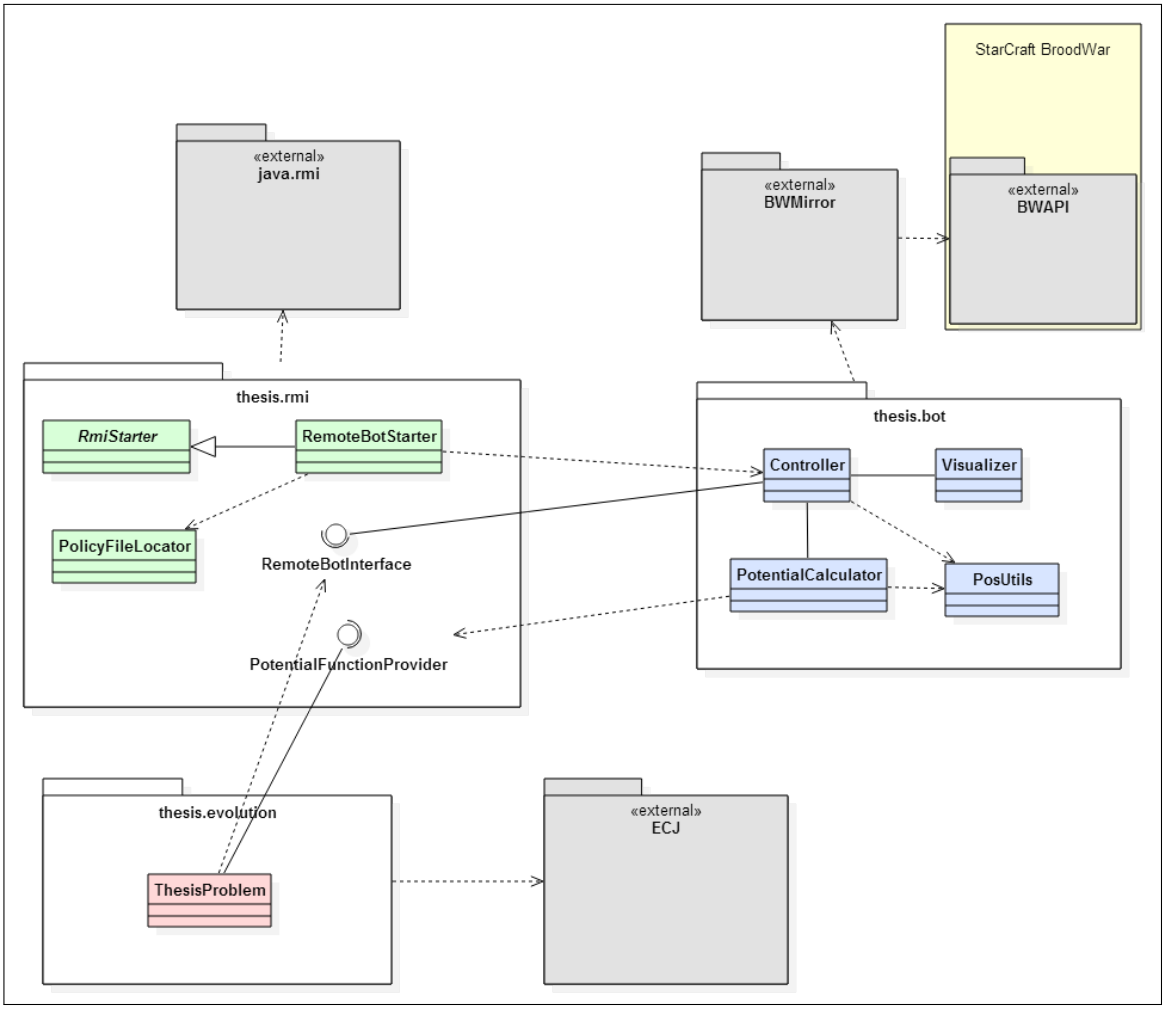
5.1 Arkkitehtuuri



Kuvio 8: Korkean tason systeemikuva.

Kuviossa 8 on havainnollistettu toteutetun järjestelmän arkkitehtuuria korkealla tasolla. Kuviossa 9 on kuvattu järjestelmän muodostavat luokat. Esitellään seuraavaksi järjestelmän muodostavat luokat lyhyesti.

Paketti `thesis.bot` sisältää StarCraft-botin toimintalogiikan. Botin toiminnasta päävastuussa on `Controller`-luokka, joka lähettää komennot yksiköille StarCraftiin käyttäen C++:lla toteutettua BWAPI-kirjastoa BWMirror-kääreen (engl. *wrapper*) kautta. `Visualizer`-luokka piirtää StarCraft-ikkunaan erilaisia visualisointeja. Luokka mahdollistaa esimerkiksi liikkuvien yksiköiden päämäärän sekä hyökkäävien yksiköiden kohdeyksikön visualisoinnin. `PotentialCalculator`-luokka laskee yksikön liikkumissuunnan joko kovakoodattuja potentiaalifunktioita tai `PotentialFunctionProvider`-rajapinnan toteuttavaa `ThesisProblem`-luokkaa käyttäen. Kovakoodatuilla potentiaalifunktioilla voidaan testata evoluution tuloksena saatuja potentiaalifunktioita, kun `ThesisProblem`-luokka taas välittää potentiaalifunktiot botille evoluutioprosessin aikana. `PosUtils`-luokka sisältää vektorilaskutoimituksia BWMirrorin `Position`-olioille.



Kuvio 9: Luokkakaavio.

Evoluutioprosessin nopeuttamiseksi kokeiltiin useamman StarCraft-instanssin ajamista rinnakkain. Ensimmäinen lähestymistapa oli ajaa evoluutioprosessia ja StarCraft-botteja rinnakkaisissa säikeissä, mutta tämä ei ollut mahdollista, koska BWAPI erottaa usempaa bottia rinnakkain ajettaessa botit toisistaan prosessitunnisteen perusteella. Rinnakkaisuus päädyttiin toteuttamaan ajamalla jokaista botin instanssia omassa prosessissaan ja evoluutioprosessia omassaan. Bottiprosessit kommunikoivat evoluutioprosessin kanssa Javan RMI:n (Remote Method Invocation) avulla. Tutkimuksessa käytetyssä ajoympäristössä 8 rinnakkaisen instanssin ajaminen johti 3,5-kertaiseen nopeuteen evoluutioprosessissa.

Paketti `thesis.rmi` huolehtii RMI-kommunikaatiosta. Paketin luokat perustuvat Rasulin (2010) oppaaseen. `RMIStarter` on abstrakti luokka, joka toteuttaa toimenpiteet, jotka ovat samat mille tahansa RMI-rekisteriin lisättävälle oliolle. RemoteBotStarter-luokka perii RMIStarterin. Se käynnistää botin luomalla uuden `thesis.bot.Controller`-luokan olion ja rekisteröi sen RMI-rekisteriin. `Controller`-luokka toteuttaa `RemoteBotInterface`-käyttöliittymän, joka määrittää RMI:n kautta toisesta prosessista kutsuttavissa olevat metodit. Käyttöliittymä sisältää vain yhden metodin: `getRoundScore(PotentialFunctionProvider problem)`. Metodi palauttaa yhden StarCraft-ottelun pisteet, jotka on laskettu luvussa 5.4 esitetyllä evaluaatiofunktiolla. Botti käyttää parametrina välitetyssä `PotentialFunctionProvider`-käyttöliittymän toteuttavassa oliossa olevaa `getPotential()`-metodia yksiköiden liikettä ohjaavien potentiaalikenttien laskemiseen. `PolicyFileLocator`-luokka huolehtii RMI:n käyttöön tarvittavat käyttöoikeudet sisältävän tiedoston etsimisestä.

Paketti `thesis.evolution` sisältää `ThesisProblem`-luokan, jonka tehtävä on huolehtia evoluutioprosessissa yksilöiden evaluoinnista. Tämä tapahtuu käytännössä `getRoundScore()`-kutsulla `Controller`-oliolle.

5.2 Yksiköiden kontrollointi

Botti ohjaa yksiköitään laskemalla potentiaalin kunkin yksikön sijainnille sekä kahdeksalle pisteelle yksikön sijainnin ympärillä. Yksiköitä liikutetaan suuntaan, jossa potentiaali on suurin, tai ne pysyvät paikallaan, jos potentiaali on suurempi yksikön sijainnissa kuin mis-

sään sitä ympäröivässä pisteessä. Mikäli vihollisyksiköitä on aseen kantaman päässä, hyökätään näiden kimppuun. Hyökkäyksen kohteen valinta riippuu ainoastaan yksiköiden järjestyksestä StarCraftin muistissa.

Hyökkäyskomennon antaminen BWAPI:n kautta vaatii ylimääräistä logiikkaa, jotta yksikölle ei anneta muita käskyjä, ennen kuin hyökkäyksen ammus on lähtenyt liikkeelle. Jos näin tehtäisiin, hyökkäystä ei suoritettaisi loppuun, vaan yksikkö alkaisi välittömästi toteuttaa uutta käskyä. BWAPI tarjoaa `isAttacking()` ja `isAttackFrame()`-metodit yksikön hyökkäystilan tarkasteluun mutta kuten taulukosta 6 näkyy, nämä eivät yleensä palauta arvoa `true` heti pelaajan annettua hyökkäyskäskyn. Churchill ja Buro (2012) muodostivat taulukon 6 tarkastelemalla StarCraftin sisäisiä binäärimuotoisia skriptitiedostoja PyICE-työkalun avulla. (Kyseistä työkalua voidaan käyttää StarCraftin binäärimuotoisten IScript-tiedostojen tarkasteluun. IScript-tiedostot kontrolloivat esimerkiksi yksiköiden käyttäytymistä, animaatioita, hyökkäysten vahingon ajastusta ja ääniefektien ajastusta.)

Taulukosta 6 havaitaan lisäksi, että hyökkäysanimaatio on mahdollista keskeyttää antamalla esimerkiksi liikkumiskomento hieman yksiköstä riippuen (ks. Churchill 2015) joko hyökkäysanimaation alettua tai sitten, kun tietty määrä hyökkäysanimaation ruutuja on kulunut. Tällöin osa hyökkäysanimaation ruuduista jätetään näyttämättä (hyökkäyksen 9. vaihe: “vapaaehtoinen animaatio” taulukossa 6), ja yksikkö voi liikkua jo ennen kuin `isAttackFrame()`-metodi palauttaa arvon `false`. Työssä kehitetty botti ei hyödynnä tätä mahdollisuutta, vaan liikkuu hyökkäyksen jälkeen vasta, kun `isAttacking()` ja `isAttackFrame()` palauttavat kumpikin arvon `false`.

Bottia kehittäessä havaittiin, että dragoon-yksikköä käytetyllä menetelmällä liikutettaessa yksikkö seuraa viimeistä annettua liikkumiskomentoa noin 20 pikseliä jäljessä. Tämä johtunee pääasiassa yksiköiden täyteen liikkumisnopeuteen pääsyyn vaadittavasta kiihdytyksestä. Yksiköiden liikkumiskomennot päädyttiin siis antamaan 20 pikseliä pidemmälle kohdasta, jossa yksikön haluttiin sijaitsevan kullakin hetkellä ja tämä johti liikkumiseen, joka oli lähempänä haluttua sijaintia. Tarkka etäisyys riippunee yksikkökohtaisesti yksikön nopeudesta ja kiihtyvyydestä.

Taulukko 6: Tapahtumien järjestys hyökkäyskäskyn antamisen jälkeen. Taulukossa näkyvät myös BWAPI:n `unit.isAttacking()` ja `unit.isAttackframe()` -metodien paluuarvot eri vaiheissa. (Churchill ja Buro 2012)

Hyökkäyksen vaihe	isAttacking	isAttackFrame	Lisähuomautuksia
1. Yksikkö on jouten.	`false`	`false`	Yksikkö on jouten tai toteuttaa jotain muuta komentoa (esim. liikkuminen).
2. Hyökkäyskomento annetaan.	`false`	`false`	Pelaaja antaa käskyn hyökätä kohti toista yksikköä.
3. Kääntyminen kohdetta päin.	`false`	`false`	Kesto 0, jos suunta on jo oikea.
4. Kohteen lähestyminen.	`false`	`false`	Kesto 0, jos yksikkö on jo ampumaetäisyydellä.
5. Pysähtyminen.	`false`	`false`	Joidenkin yksiköiden täytyy pysähtyä kokonaan ennen ampumista.
6. Hyökkäysanimaatio alkaa.	`true`	`true`	Hyökkäysanimaatio alkaa, vahinkoa ei ole vielä tehty.
7. Vahinkoanimaatio.	`true`	`true`	Hyökkäysanimaation ruutuja, kunnes ammus lähtee liikkeelle.
8. Pakollinen animaatio.	`true`	`true`	Lisää animaatioruutuja ammuksen jälkeen (voi olla 0).
9. Vapaaehtoinen animaatio.	`true`	`true`	Toinen komento voidaan antaa ylimääräisten ruutujen keskeyttämiseksi.
10. Odota latausta.	`true`	`false`	Yksikölle voidaan antaa muita komentoja, kunnes se voi ampua uudelleen.
11. Siirry kohtaan 3.	`false`	`false`	Toista hyökkäys.

5.3 Käytetyt kirjastot

Työssä käytettiin BWAPI-[1] ja BWMirror[2]-kirjastoja StarCraftin kanssa kommunikointiin. Evoluutiolaskentaan käytettiin ECJ-kirjastoa[3].

BWAPI on ilmainen avoimen lähdekoodin C++-ohjelmistokehys, joka tarjoaa ohjelmointirajapinnan StarCraftiin. Kirjasto mahdollistaa StarCraft-ottelun tilan tarkastelun ja yksiköiden kontrolloinnin ohjelmallisesti. Rajapinnan tarjoamaa tietoa voidaan rajoittaa niin, että yksiköitä, jotka ovat Fog of Warin peitossa, ei ole mahdollista havaita sen kautta. Tämä mahdollistaa sellaisten StarCraft-bottien kehittämisen, jotka ovat lähtökohdiltaan samalla viivalla ihmispelaajien kanssa.

BWMirror on kirjasto, joka mahdollistaa BWAPIn käyttämisen Java-ohjelmointikielellä. Ennen BWMirroria BWAPI:n käyttäminen Javalla oli mahdollista JNIBWAPI-kirjaston[4] kautta, mutta JNIBWAPI tuki vain osaa BWAPI:n toiminnallisuuksista. BWMirror on toteutettu automaattisesti peilaamalla koko BWAPI:n rajapinta, joten sen avulla BWAPI on kokonaan käytettävissä Javalla.

1. https://github.com/bwapi/bwapi
2. https://github.com/vjurenka/BWMirror
3. https://cs.gmu.edu/~eclab/projects/ecj/
4. https://github.com/JNIBWAPI/JNIBWAPI

Evoluutiolaskentamenetelmille on olemassa lukuisia valmiita ohjelmointikirjastoja, joten geneettistä ohjelmointia ei kannata toteuttaa itse. Työn alussa botin toteutus tehtiin C++-kielellä, ja evoluutiokirjastoksi oli kaksi kandidaattia: Evolving Objects[5] ja Open Beagle[6]. Evolving Objects vaikutti näistä pidemmälle viedyltä, mutta Open Beaglen käyttöönotto osoittautui Windows-ympäristössä helpommaksi, joten työssä päädyttiin aluksi käyttämään kyseistä kirjastoa.

Melko pian Open Beaglessa havaittiin virhe, jossa mutaatiotodennäköisyys oli oletuksena 100 % dokumentoidun 5 % sijaan. Virheen seurauksena evoluutioprosessissa risteytyksessä luoduista yksilöistä tuli kaikista vanhempiyksilöihinsä verrattuna hyvin erilaisia, eikä niiden kelpoisuus siten parantunut lainkaan. Näin merkittävä virhe sai kirjaston vaikuttamaan epäluotettavalta. Kirjaston dokumentaation kattavuus ja kieliasu olivat myös epätyydyttäviä, eikä kirjastoa ollut kehitetty enää vuoden 2012 jälkeen.

Koska toimivan C++-evoluutiokirjaston löytäminen oli haastavaa, päädyttiin botin toteutuskieli vaihtamaan Javaan ja käyttämään evoluutiokirjastona ECJ-kirjastoa, jota kehitetään edelleen aktiivisesti. ECJ on toiminut työssä hyvin, ja sen käyttöön oli helppo perehtyä kattavan dokumentaation avulla. ECJ:n painopiste on geneettisessä ohjelmoinnissa (Luke 2014), mikä on tämän tutkimuksen kannalta positiivista.

5.4 Geneettinen ohjelmointi

Evoluutioprosessin parametrien pohjana käytettiin ECJ:n `koza.params`-konfiguraatiotiedoston parametreja, jotka pohjautuvat Kozan (1992) käyttämiin parametreihin. Kyseiset parametrit nähtiin hyvänä lähtökohtana testata geneettisen algoritmin toimivuutta tutkimusongelman ratkaisussa. Niiden käyttö nopeutti botin toteuttamista, koska lähtökohtana käytetty parametritiedosto kuului valmiina evoluutiokirjastoon. Merkittävimmät `koza.params`-listan parametrit on lueteltu alla.

- Lisääntymisparametrit: 10 % yksilöistä tuotetaan kopioimalla edellisen sukupolven yksilöitä, 90 % risteytyksellä. Mutaatiota ei käytetä lainkaan.

5. http://eodev.sourceforge.net/
6. https://code.google.com/p/beagle/

- Valintamenetelmä: Sekä kopiointi että risteytys käyttävät valintamenetelmänä turnausvalintaa, jossa turnauskoko on 7 yksilöä. Tämä itseasiassa poikkeaa Kozan (1992) menetelmästä, jossa käytettiin yhtä kokeilua lukuunottamatta rulettivalintaa.
- Alustuksessa käytetään ramped half-and-half -menetelmää, jossa puiden syvyydet vaihtelevat kahdesta kuuteen.
- Risteytyskohtaa valittaessa valitaan 10 % todennäköisyydellä terminaalisolu ja 90 % todennäköisyydellä funktio.

Prosessissa käytetään pääosin seuraavaa Sandbergin (2011) tutkimuksesta lainattua evaluaatiofunktiota

$$F = \sum_{i=0}^{n} S_{Bi} + \sum_{i=0}^{n_k} S_{Ki} - \sum_{i=0}^{n_d} S_{Ki} + B,$$

jossa S_B on pelaajalla pelin alussa olevasta rakennetusta yksiköstä annettavat pisteet (kyseiset pisteet tutkimuksessa esiintyville yksiköille on esitetty taulukossa 2), n on botin hallussa pelin alussa olevien yksiköiden määrä, S_K on tuhotusta vihollisyksiköstä annettavat pisteet ($S_K = 2S_B$), n_k on tuhottujen vihollisyksiköiden määrä, n_d on vihollisen tuhoamien botin yksiköiden määrä ja

$$B = \begin{cases} \sum_0^{n_s} H_i, & \text{jos } t \leq t_{max} \\ 0, & \text{muutoin} \end{cases},$$

jossa H on yksiköllä ottelun lopussa jäljellä olevat osumapisteet, n_s on botin pelin lopussa jäljellä olevien yksiköiden määrä, t on ottelun kesto ja t_{max} on ottelun maksimikesto. Evaluaatiofunktion termi $\sum_{i=0}^{n} S_{Bi}$ ei itse asiassa riipu botin suorituksesta, vaan se on samalla yksikkökokoonpanolla joka ottelussa sama. Funktiossa esiintyy StarCraftin satunnaisista elementeistä johtuvaa kohinaa, eli vaikka botin toiminta säilyy samanlaisena, ottelujen tulokset vaihtelevat.

Evoluutioprosessissa vaikutti usein syntyvän botteja, jotka käyttäytyivät passiivisesti keräänten yhteen kasaan lähelle aloitussijaintiaan tai kartan reunaa odottamaan vihollista. Kyseessä lienee lokaali optimiratkaisu, sillä se ei mahdollista esimerkiksi kiting-taktiikan hyödyntämistä. Passiivisten ratkaisujen ongelmaa yritettiin ratkaista luomalla skenaarioita, joissa vihollisyksiköt aktivoituvat vasta pelaajan yksiköiden tultua riittävän lähelle. Tavoitteena oli suosia aktiivisia ratkaisuja. Tässä menetelmässä on kuitenkin ongelma Sandbergin evaluaa-

tiofunktiota käytettäessä, sillä yksilö, joka odottaa ottelun maksimikestoon asti kohtaamatta vihollista saattaa saada paremman kelpoisuuden kuin yksilö, joka taistelee vihollisyksiköiden kanssa huonolla menestyksellä. Esimerkiksi yksikkökokoonpanolla, jossa botilla on käytössän 9 goliathia ja 1 SCV ja vastustajalla puolestaan 21 hydraliskia ja 1 drone (ks. taulukko 7), botti, joka pysyttelee paikallaan odottaen ottelun päättymistä maksimikeston ylittymisen vuoksi saa käsittelemättömäksi kelpoisuudekseen 1850. Nämä pisteet tulevat pelin alussa käytössä olevista yksiköistä. Botti, joka taistelee vihollisen kanssa, mutta häviää ottelun saa kelpoisuudekseen -1850–5600 tuhottujen vihollisyksiköiden määrästä riippuen.

Yhdessä toteutetuista koeajoista (koeajo K7) evaluaatiofunktiota muutettiin niin, että kelpoisuus ei enää laske vihollisen tuhoamista omista yksiköistä. Hengissä säilyneiden omien yksiköiden jäljellä olevat osumapisteet joka tapauksessa kasvattavat kelpoisuutta, joten kelpoisuuden laskeminen yksiköitä menetettäessä vaikutti turhalta toistolta. Tällä ei kuitenkaan vaikuttanut olevan suurta vaikutusta.

Toisaalta on parempi, jos taistelusta selviää kaksi yksikköä, joiden osumapisteistä on puolet jäljellä kuin että selvinneitä yksiköitä olisi yksi, ja sillä olisi täydet osumapisteet. Tämä johtuu siitä, että kaksi yksikköä kykenee tekemään tuplasti vahinkoa yhteen yksikköön verrattuna. Evaluaatiofunktiota kannattaisi siis kehittää vielä niin, että jokaisesta selvinneestä yksiköstä saisi tietyn määrän pisteitä jäljellä olevien osumapisteiden määrästä riippumatta. Nämäkin pisteet tulisi antaa vain mikäli ottelu päättyy ennen maksimikeston saavuttamista, jotta aktiivinen botti saisi paremman kelpoisuuden kuin taistelua välttävä botti.

6 Tulokset

Luvussa 6.1 tarkastellaan suoritettuja koeajoja ja luvussa 6.2 vertaillaan niiden tuloksia aiempaan tutkimukseen.

6.1 Koeajot

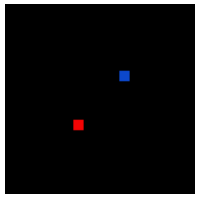
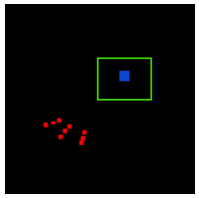
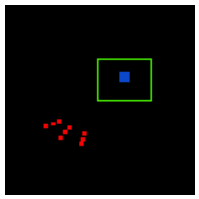
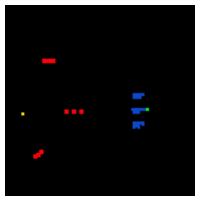
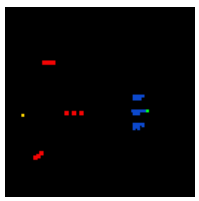
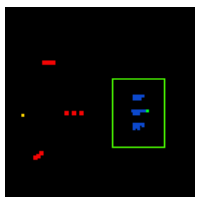
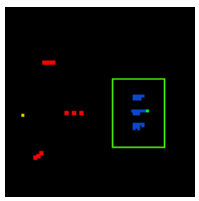
Toteutettua järjestelmää testattiin suorittamalla seitsemän koeajoa hieman erilaisilla parametreilla. Tavoitteena oli selvittää, soveltuuko geneettinen ohjelmointi bottia kontrolloivien potentiaalikenttien optimointiin. Etenkin käytettävillä funktioilla ja geneettisille ohjelmille syötettävillä parametreilla arveltiin olevan mahdollisuus vaikuttaa menetelmän suoriutumiseen. Osa koeajoista tehtiin myös käyttäen karttaa, jota Sandberg (2011) käytti omassa tutkimuksessaan, jotta vertailu hänen tuloksiinsa olisi mahdollista.

Koeajot suoritettiin laitteistolla, jonka merkittävimmät komponentit olivat AMD FX-8350 -prosessori, 8 GB 1333 MHz:n DDR3-muistia ja Windows 8 -käyttöjärjestelmä. Osassa koeajoista prosessoria ei voitu kuitenkaan käyttää täydellä teholla laitteiston prosessorin ylikuumenemisongelmien vuoksi.

6.1.1 Kartat

Koeajoissa käytetyt kartat on esitetty taulukossa 7. Kartat jakaantuvat yksikkökokoonpanon puolesta kahteen ryhmään. Ensimmäisessä ryhmässä asetelma on symmetrinen, sekä botilla että vastustajalla on käytössä 9 dragoon-yksikköä. Tässä asetelmassa kiting-taktiikka ei ole mahdollinen, koska se vaatii omilta yksiköiltä vastustajan yksiköitä suurempaa nopeutta ja pidempää aseiden kantomatkaa. Toimiva taktiikka olisi asettua rintamaksi vastustajaan nähden, jotta mahdollisimman moni yksikkö pääsee ampumaetäisyydelle. Lisäksi yksiköiden pitäisi perääntyä hetkeksi, mikäli ne vahingoittuvat liikaa, jotta vastustajan yksiköt vaihtaisivat tulituksen kohdetta. Myös omien yksiköiden tulituksen keskittäminen yksittäisiin vastustajan yksiköihin olisi tehokasta, mutta tämä ei tutkimuksessa käytetyllä botilla ole mahdollista, koska tulituksen kohteeksi valittavaa vihollisyksikköä ei kontrolloida.

Taulukko 7: Koejärjestelyissä käytetyt skenaariot. Passiiviset vihollisyksiköt aktivoituvat omien yksiköiden saapuessa vihreällä suorakulmiolla merkatulle alueelle (ks. luku 5.4).

Tunniste	Kartta	Omat yksiköt	Vihollisen yksiköt	Vihollisen tekoäly
K1		9 dragoonia	9 dragoonia	Aktiivinen
K2		9 dragoonia	9 dragoonia	Passiivinen
K3		9 dragoonia	9 dragoonia	Passiivinen
K4		9 goliathia 1 SCV	21 hydraliskia 1 drone	Aktiivinen
K5		9 goliathia 1 SCV	21 hydraliskia 1 drone	Aktiivinen
K6		9 goliathia 1 SCV	21 hydraliskia 1 drone	Passiivinen
K7		9 goliathia 1 SCV	21 hydraliskia 1 drone	Passiivinen

Toisessa ryhmässä käytetään samaa yksiköiden kokoonpanoa ja sijoittelua kuin Sandbergin (2011) tutkimuksessa. Botilla on käytössään 9 goliathia ja vastustajalla 21 hydraliskia. Lisäksi kummallakin puolella on yksi rakentajayksikkö: botilla SCV ja vastustajalla drone. Sandbergin mukaan rakentajayksiköiden lisäämisellä kokoonpanoon tavoiteltiin eri ratkaisujen vahvuuden parempaa erottelua ja tasapainottelua aggressiivisen ja puolustavan käyttäytymisen välillä. Liian passiivinen botti menettää rakentajan helposti. Ilmeisesti myös Sandbergilla on esiintynyt luvussa 5.4 kuvattu liian passiivisten strategioiden ongelma.

Goliathit vastaan hydraliskit -asetelmassa riittävän hyvällä kiting-taktiikan hallinnalla pitäisi teoriassa olla mahdollista saavuttaa lopputulos, jossa omat yksiköt eivät kärsi lainkaan vahinkoa. Goliathit täyttävät hydraliskeihin nähden molemmat kiting-taktiikan käyttöön tarvittavat ominaisuudet (ks. taulukko 2).

BWAPI mahdollistaa tarvittaessa boteille täydellisen informaation saannin myös Fog of Warin peittämistä alueista. Tutkimuksessa päädyttiin käyttämään tätä mahdollisuutta, koska Fog of Warin peittämien alueiden tiedustelu rajattiin tutkimuksen ulkopuolelle. Botin oli näin ollen mahdollista luoda potentiaalikentät vihollisyksiköille, vaikka ne eivät olleet vielä omien yksiköiden näköpiirissä.

6.1.2 Evoluutioparametrit

Geneettisessä ohjelmoinnissa käytetyt parametrit on esitetty taulukossa 8. Parametreissa kokeiltiin useita erilaisia yhdistelmiä. Populaation kokona kokeiltiin 128:aa ja 500:aa.

Geneettisessä ohjelmoinnissa kokeiltiin kahta funktiojoukkoa (taulukko 9). Koeajoissa K1, K2, K3, K4 ja K5 käytettiin minimaalista funktiojoukkoa, jossa oli peruslaskutoimitukset sekä satunnaisvakio väliltä [500, 500). Koeajoissa K6 ja K7 käytettiin huomattavasti laajempaa funktiojoukkoa, joka sisälsi peruslaskutoimitusten lisäksi kaksi eri vaihteluvälin satunnaisvakiota, trigonometrisia funktioita, potenssi-, juuri- ja eksponenttifunktioita sekä neliparametrisen ehtolauseen, joka palautti 3. parametrin, jos 1. parametri oli arvoltaan pienempi kuin 2. parametri ja muutoin 4. parametrin.

Koeajossa K6, jossa laajennettua funktiojoukkoa käytettiin ensimmäistä kertaa, unohdettiin tarvittavat uudet funktiot suojata virheellisiltä parametreilta. Esimerkiksi neliöjuurifunktio

palauttaa Javassa negatiivisella parametrilla epäluvun (engl. not a number, NaN), jonka vertaaminen mihin tahansa lukuun puolestaan tuottaa aina arvon epätosi. Koeajoon K7 tarvittavat funktiot suojattiin siten, että ne palauttavat kaikilla parametreilla jonkin arvon. Esimerkiksi negatiivisen luvun neliöjuuri palauttaa parametrina annetun luvun itseisarvon neliöjuuren.

Laajemman funktiojoukon valinta toteutettiin ottaen mallia ECJ-kirjaston lähdekoodin mukana jaettavasta esimerkkiprojektista, jossa geneettistä ohjelmointia käytettiin symboliseen regressioon, eli datajoukkoa vastaavan funktion etsimiseen. Lisäksi mukaan otettiin vielä ehtofunktio, josta arveltiin mahdollisesti olevan hyötyä potentiaalifunktioiden rakentamisessa. Kaikissa koeajoissa käytettiin neljää potentiaalifunktiota:

- Vihollisyksiköt, kun oman yksikön ase on ampumavalmis.
- Vihollisyksiköt, kun oman yksikön ase odottaa jäähtymisajan päättymistä.
- Omat yksiköt.
- Lähin kartan reuna.

Vihollisyksiköihin liittyvissä potentiaalikentissä otettiin huomioon vain se vihollisyksikkö, joka aiheutti suurimman potentiaalin pisteessä, jolle potentiaalia laskettiin. Näin tehtiin, koska Hagelbäck ja Johansson (2008b) havaitsivat potentiaalikenttiä mikromanagerointiin käyttäessään useamman vihollisyksikön potentiaalin summaamisen johtavan omien yksiköiden hakeutumiseen vihollisyksiköiden ryhmittymän vahvimpaan kohtaan. Omien yksiköiden potentiaalit puolestaan summattiin kaikki yhteen.

Omien yksiköiden ja kentän reunan potentiaalifunktioille annettiin parametriksi ainoastaan kohteen etäisyys pisteestä, jolle potentiaalia laskettiin. Vihollisyksiköiden potentiaalifunktioille testattiin eri parametreja. Taulukossa 8 on esitetty, missä koeajossa käytettiin mitäkin taulukossa 10 esitetyistä parametrijoukoista.

Koeajossa K7 käytettiin ramped half-and-half -menetelmässä puun maksimikokona kahdeksaa, kun se muissa yksilöissä oli kuusi. Suurempien puiden ajateltiin mahdollisesti olevan alustuksessa tarpeen aiempaa suuremman funktiojoukon vuoksi.

Taulukko 8: Koeajoissa käytetyt evoluutioparametrit.

#	Populaatio	Sukupolvia	Funktiot (Taulukko 9)	Parametrit vihollisyksiköiden potentiaalille	Huomautuksia
K1	128	26	Funktiojoukko 1	Parametrijoukko 1	
K2	128	26	Funktiojoukko 1	Parametrijoukko 1	
K3	128	26	Funktiojoukko 1	Parametrijoukko 2	
K4	128	26	Funktiojoukko 1	Parametrijoukko 2	
K5	500	26	Funktiojoukko 1	Parametrijoukko 2	
K6	500	14	Funktiojoukko 2	Parametrijoukko 3	Suojaamattomat funktiot (Ks. luku 6.1.2)
K7	500	26	Funktiojoukko 2	Parametrijoukko 3	Half builder 2–8 (Ks. luku 6.1.2)

Taulukko 9: Koeajoissa käytetyt funktiojoukot.

Funktiojoukko 1	+, -, *, / ja satunnaisvakio väliltä [-500, 500)
Funktiojoukko 2	+, -, *, /, satunnaisvakio väliltä [-500, 500), satunnaisvakio väliltä [-1, 1), kosini, sini, tangentti, hyperbolinen tangentti, neliö, neliöjuuri, kuutio, eksponentti, ehtofunktio ja negatiivinen eksponentti

Taulukko 10: Parametrijoukot vihollisyksiköiden potentiaalifunktioille.

Parametrijoukko 1	Vihollisyksikön etäisyys
	Oman yksikön ampumaetäisyys
Parametrijoukko 2	Vihollisyksikön etäisyys
	Oman yksikön ampumaetäisyys
	Oman yksikön jäljellä olevat HP:t verrattuna kaikkien omien yksiköiden keskiarvoon
Parametrijoukko 3	Vihollisyksikön etäisyys
	Oman yksikön ampumaetäisyys
	Oman yksikön jäljellä olevat HP:t verrattuna kaikkien omien yksiköiden keskiarvoon
	Omasta yksiköstä vihollisyksikköön osoittavan vektorin X-koordinaatti
	Omasta yksiköstä vihollisyksikköön osoittavan vektorin Y-koordinaatti

Taulukko 11: Linkit videoihin koeajojen parhaiden yksilöiden pelityyleistä.

K1:	https://www.youtube.com/watch?v=D3wuNmzqfOQ
K2:	https://www.youtube.com/watch?v=1Bdbs7laQvw
K3:	https://www.youtube.com/watch?v=fpFrZJWSmmc
K4:	https://www.youtube.com/watch?v=V1fzx-6j1xY
K5:	https://www.youtube.com/watch?v=RmQPA4HRFos
K6:	https://www.youtube.com/watch?v=E4tDVihy5GQ
K7:	https://www.youtube.com/watch?v=KuKXWABDtDU

6.1.3 Koeajojen tulokset

Videonäyteet kunkin koeajon parhaan yksilön pelityylistä löytyvät taulukossa 11 olevien linkkien kautta. Alla kuvaillaan kyseisten yksilöiden pelityyliä myös sanallisesti.

Koeajon K1 paras yksilö pitää yksikkönsä tiukassa ryhmittymässä, kunnes vihollisen yksiköt tulevat taisteluetäisyydelle. Yhteenoton alkaessa botti levittää muodostelmaa hieman, jolloin useampi yksikkö yltää ampumaan vihollista. Muutaman yksikön ollessa enää jäljellä, yksiköt noudattavat kiting-taktiikkaa. Vaikuttaa siltä, että yksiköt yrittävät käyttää kiting-taktiikkaa jo taistelun alkaessa, mutta kauempana vihollisesta olevat yksiköt ovat lähempänä olevien yksiköiden tiellä, eivätkä nämä siksi voi paeta. Botin suoriutuminen saattaisi olla parempaa, jos omien yksiköiden toisilleen aiheuttamat potentiaalikentät hylkisivät toisiaan voimakkaammin.

Koeajossa K2 vihollisyksiköiden käyttäytymistä muutettiin niin, että ne aktivoituvat vasta omien yksiköiden tullessa riittävän lähelle. Vihollisyksiköillä käytettiin kuitenkin tekoälyskriptiä, joka estää niitä liikkumasta kauaksi lähtöpisteestään. Siksi ne lähtevät palaamaan kohti alkusijaintiaan jouduttuaan siitä liian kauaksi. Tässä koeajossa paras yksilö päihittää vihollisen helpohkosti. Botin yksiköt lähestyvät vihollista ja noudattavat vihollisyksiköt kohdattuaan kiting-taktiikkaa onnistuneesti päihittäen vihollisen useita kertoja. On kuitenkin mahdollista, että hyvä suoriutuminen johtuu vihollisyksiköiden edellä mainitusta käyttäytymismallista.

Koeajossa K3 vihollisyksiköiden käytösmallia muutettiin niin, että botin yksiköiden tultua riittävän lähelle, vihollisyksiköt eivät enää palaa lähtöpisteeseen, vaan jahtaavat botin yk-

siköitä loputtomasti. Koeajon paras yksilö vaikuttaa silti suoriutuvan melko hyvin, voittaen useimmat ottelut. Voittomarginaali on kuitenkin pienempi (botin yksiköistä pienempi osa jää henkiin), ja osa otteluista hävitään. Koeajon K3 parhaan yksilön noudattama taktiikka on samantapainen kuin koeajon K2 parhaan yksilön, tosin botin yksiköt pysyvät hieman tiiviimmässä ryhmässä, jolloin kiting-taktiikka ei näytä toimivan aivan optimaalisesti, vaan yksiköt ovat silloin tällöin toistensa tiellä.

Koeajossa K3 testattiin ensimmäistä kertaa omien yksiköiden jäljellä olevien osumapisteiden ja kaikkien omien yksiköiden jäljellä olevien osumapisteiden keskiarvon välisen suhteen käyttämistä parametrina vihollisyksiköiden aiheuttamissa potentiaalikentissä. Pelkän videokuvan perusteella on vaikea analysoida, onko tällä parametrilla vaikutusta koeajon parhaan yksilön käyttäytymiseen. Parametri näyttää esiintyvän siinä potentiaalifunktiossa, joka määrittää vihollisyksiköiden aiheuttaman potentiaalin oman yksikön aseen ollessa ampumavalmis. Parametrin vaikutusta kyseisen yksilön suoriutumiseen voisi mahdollisesti testata ottamalla vertailukohdaksi saman yksilön muokattuna siten, että kyseinen parametri saisi aina vakioarvon yksikön jäljellä olevista osumapisteistä riippumatta.

Koeajot K4, K5, K6 ja K7 suoritettiin Sandbergin (2011) goliathit vastaan hydraliskit -skenaariossa. Kyseisessä skenaariossa botti on pahasti alakynnessä, ellei se pysty hyödyntämään yksiköidensä vastustajan yksiköitä parempaa liikkumisnopeutta ja hyökkäysetäisyyttä kiting-taktiikan avulla.

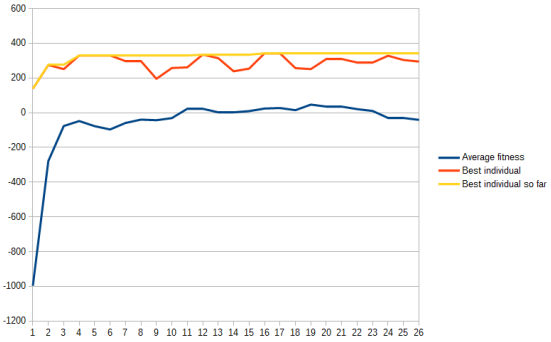
Koeajossa K4 parhaan yksilön yksiköt käyvät suoraviivaisesti vihollisen kimppuun, eivätkä näytä noudattavan mitään hyödyllistä taktiikkaa. Botti häviää tehden viholliseen vain vähän vahinkoa.

Koeajosta K5 alkaen populaation koko kasvatettiin 128:sta yksilöstä 500:aan. Koeajossa K5 parhaan yksilön yksiköt kerääntyvät melko tiiviiksi ryhmittymäksi vihollisten vastaiseen kentän reunaan odottamaan vihollisyksiköiden lähestymistä. Videon perusteella botti kykenee tällä menetelmällä tekemään huomattavasti koeajon K4 parasta yksilöä enemmän vahinkoa vihollisyksiköille tiiviimmän muodostelmansa ansiosta. Tässä muodostelmassa useammat goliatheista ampuivat samoja vihollisyksiköitä ja osa vihollisyksiköistä on toistensa tiellä, eivätkä ne pääse kaikki tulittamaan botin yksiköitä samanaikaisesti. Taktiikka on kui-

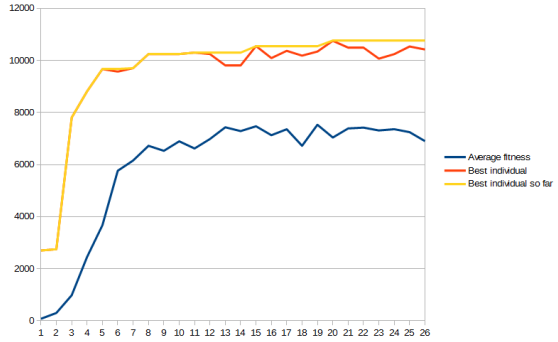
tenkin passiivinen, ja kiting on vaikeaa, koska kentän reuna estää omia yksiköitä pakenemasta vihollisyksiköitä vastakkaiseen suuntaan. Kentän reunaa pitkin pakeneminen saattaisi olla mahdollista, mutta koska omat yksiköt vaikuttavat vetävän toisiaan puoleensa, lienee tällaisen käyttäytymismallin suuntaan kehittyminen evoluutioprosessissa vaikeaa. Kyseessä on siis lokaali optimikohta, johon botti jää helposti jumiin evoluutioprosessissa. Koeajon K5 paras yksilö vaikuttaisi suoriutuvan parhaiten Sandergin ja Togeliuksen skenaariota käyttäneiden koeajojen boteista.

Koeajoissa K6 ja K7 käytettiin koeajojen K2 ja K3 tapaan vihollisyksiköille käyttäytymismallia, jossa ne aktivoituvat vasta botin yksiköiden saavuttua riittävän lähelle. Lisäksi koeajoissa K6 ja K7 käytettiin laajennettua funktiojoukkoa ja vihollisyksiköitä koskeville potentiaalifunktiolle annettiin aikaisempien parametrien lisäksi parametriksi myös vihollisyksikön suunnan ja etäisyyden osoittavan vektorin x- ja y-koordinaatit. Koeajossa K7 käytettiin myös populaation alustuksessa hieman aiempaa syvempiä puita. Koeajo K6 keskeytyi, kun tutkimuskone jouduttiin käynnistämään uudelleen virhetilanteen vuoksi 16. sukupolven kohdalla. Suurimmassa osassa koeajoista ei näyttänyt tapahtuvan merkittävää kehitystä enää evoluutioprosessin tässä vaiheessa, joten koeajo 6 otettiin keskeytymisestä huolimatta mukaan aineistoon.

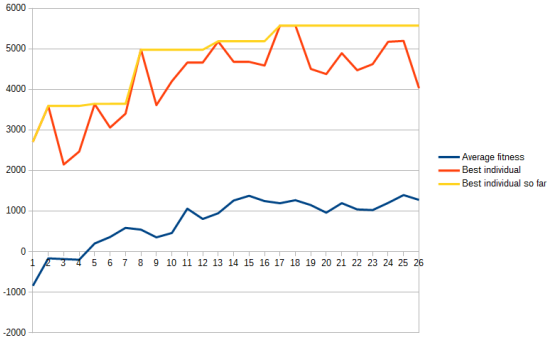
Koeajojen K6 ja K7 parhaat yksilöt käyttäytyvät melko samaan tapaan kuin koeajon K4 paras yksilö, eli ne käyvät melko suoraviivaisesti vihollisyksiköiden kimppuun. Yksilöissä on kuitenkin havaittavissa ominaisuus, jossa botin käytössä olevista kolmesta kolmen goliathin ryhmittymästä skenaarion alussa lähinnä vihollisyksiköitä oleva ryhmittymä hidastaa vauhtiaan, jolloin botin yksiköt kohtaavat vihollisyksiköt yhtenäisempänä ryhmänä kuin koeajon K4 parhaan yksilön taktiikassa. Vihollisyksiköiden passiivinen käyttäytyminen mahdollistaa botin yksiköille odottamisen vihollisyksiköt aktivoivan alueen reunalla, kunnes kaikki yksiköt ovat saapuneet riittävän lähelle. Käytösmalli on voimakkaampi koeajon K7 parhaassa yksilössä. Osa koeajon K7 parhaan yksilön yksiköistä jää taistelun alkaessa hieman liian kauaksi taistelusta, eikä yllä ampumaan vihollisyksiköitä. Tämän vuoksi kyseinen yksilö näyttää suoriutuvan hieman koeajon K6 parasta yksilöä huonommin. Koeajon K6 parhaan yksilön yksiköt käyttävät tulivoimansa taistelun alettua tehokkaasti. Laajemman funktiojoukon vaikutusta evoluutioprosessin tuloksiin on vaikea analysoida tarkkaan, koska skenaario poikke-



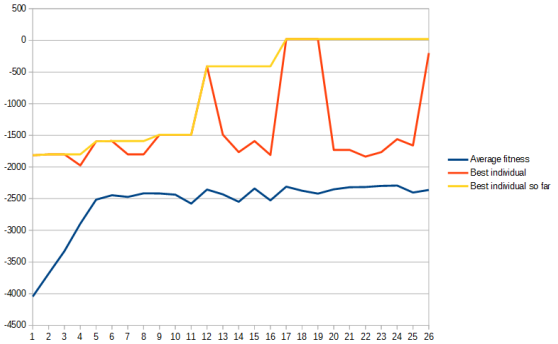
(a) Koeajo K1.



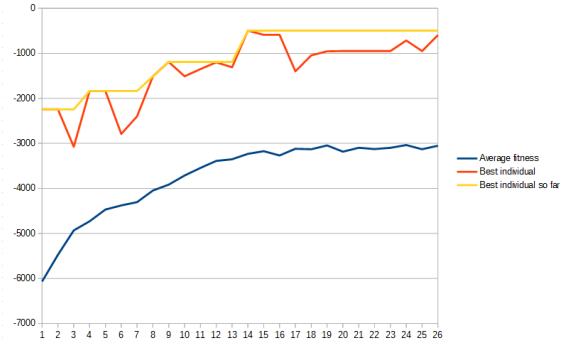
(b) Koeajo K2.



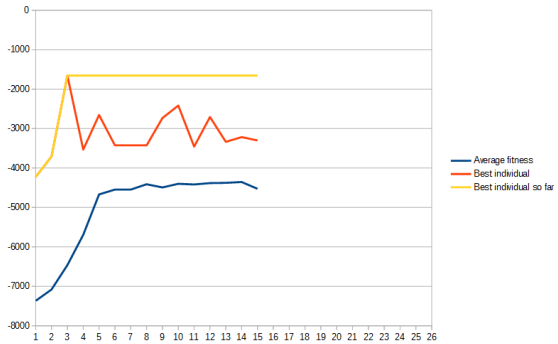
(c) Koeajo K3.



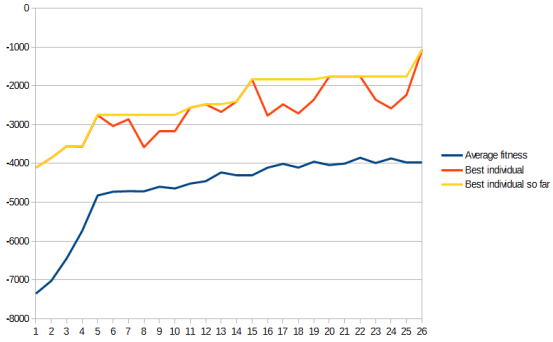
(d) Koeajo K4.



(e) Koeajo K5.



(f) Koeajo K6.



(g) Koeajo K7.

Kuvio 10: Koeajoissa tapahtunut kelpoisuuden kehitys.

si koeajoista K4 ja K5 vihollisyksiköiden käyttäytymisen osalta. Suurta muutosta suuntaan tai toiseen funktiojoukon laajentamisesta ei kuitenkaan seurannut. Koeajon K7 alustuksessa käytetty suurempi puiden maksimikoko ei myöskään näytä merkittävästi vaikuttavan parhaan yksilön suoritukseen koeajoon K6 verrattuna.

Koeajojen K6 ja K7 parhaiden yksilöiden taktiikka saattaa itse asiassa edustaa ylisovittamista, eli sopeutumista nimenomaan tämän harjoitusskenaarion olosuhteisiin taktiikalla, joka ei välttämättä toimi kaikissa mahdollisissa skenaarioissa. Toki yhtenäisenä ryhmänä taisteleminen on useimmiten eduksi, mutta saattaa olla mahdollista, että tapa, jolla yksiköt ryhmittyvät vihollisyksiköt aktivoivan alueen ulkopuolelle, ei välttämättä toimisi muissa skenaarioissa. Tutkimuksessa käytetyssä evoluutioprosessissa onkin hieman ongelmallista vain yhden skenaarion käyttäminen evoluutioprosessin aikana harjoitusmateriaalina.

Kuviossa 10 on esitelty koeajoissa tapahtunutta kelpoisuuden kehitystä. Jokaisessa kuvaajassa on sukupolven keskiarvokelpoisuus, sukupolven parhaan yksilön kelpoisuus ja evoluutioprosessin kyseiseen sukupolveen mennessä parhaan yksilön kelpoisuus. Kyseessä ovat raa'at kelpoisuusluvut muunnettuna siten, että positiivinen kelpoisuus yhdessä ottelussa tarkoittaa voittoa ja negatiivinen tappiota. Yksilön kelpoisuus on kymmenen ottelun tuloksen keskiarvo. Kuvaajista nähdään, että kelpoisuudet ovat kehittyneet evoluutioprosessin aikana. Kehitys prosessin alussa näyttää useimmissa koeajoissa olevan nopeaa, mutta se tasoittuu kaikissa koeajoissa noin 10. sukupolven kohdalla. Koeajojen K2 ja K3 parhaiden yksilöiden kelpoisuus on korkealla tasolla, mutta varsinkin koeajojen K4, K5, K6 ja K7 parhaiden yksilöiden kelpoisuus on matalalla tasolla. Vaikka koeajojen K4 ja K5 parhaat yksilöt näyttäisivät olevan liki tasaväkisiä pelin sisäänrakennetun tekoälyn kanssa, eivät ne silti voittaneet sitä lainkaan evoluutioprosessin jälkeen testattaessa. Tämä liittyy evoluutioprosessissa havaittujen kelpoisuuksien poikkeavuuteen, jota käsitellään seuraavassa luvussa.

6.1.4 Poikkeavuus koejaojen kelpoisuustuloksissa

Evoluutioprosessissa tallennetuissa yksilöiden kelpoisuuksissa oli havaittavissa poikkeavuus, jossa parhaiden yksilöiden evoluutioprosessissa saama kelpoisuus oli huomattavasti parempi kuin kelpoisuus, joka samoilla yksilöillä havaittiin evoluutioprosessin jälkeen pidempiä 200

ottelun sarjoja pelaamalla. Näissä jälkikäteen tehdyissä pidemmissä testiajoissa havaittu kelpoisuus vastasi evoluutioprosessissa yksilöille arvioitua kelpoisuutta heikon kelpoisuuden yksilöillä, mutta tietyn rajan kohdalla parempi evoluutioprosessissa havaittu kelpoisuus ei tarkoittanut jälkikäteen tehdyssä testissä kelpoisuuden parantumista juuri lainkaan. Ilmiötä on havainnollistettu kuviossa 11. Kuvion kelpoisuusluvut ovat standardisoituja, eli pienempi kelpoisuus on parempi.

Kun poikkeavuus havaittiin ensimmäisen kerran, sen oletettiin liittyvän tulosten satunnaisvaihteluun. Keskinkertainen yksilö saattaisi saada satunnaisvaihtelun tuloksena todellista paremman kelpoisuuden. Poikkeavuus oli kuitenkin melko suuri ja yksilöiden kelpoisuudessa käytettiin kymmenen ottelun keskiarvoa.

Seuraavassa analyysissa pyritään tutkimaan, miten todennäköistä havaitun kaltaisen poikkeavuuden syntyminen satunnaisvaihtelun tuloksena olisi. Eräässä tapauksessa evoluutioprosessissa saatu 10 ottelun pistemäärä oli 7645,0 pistettä (raaka pistemäärä), ja saman yksilön pistemäärä jälkikäteen ajetussa 200 ottelun sarjassa oli 5385,5 pistettä 701,67 pisteen keskihajonnalla. Z-arvo jo esimerkiksi 7000 pisteelle 10 ottelun keskiarvona on

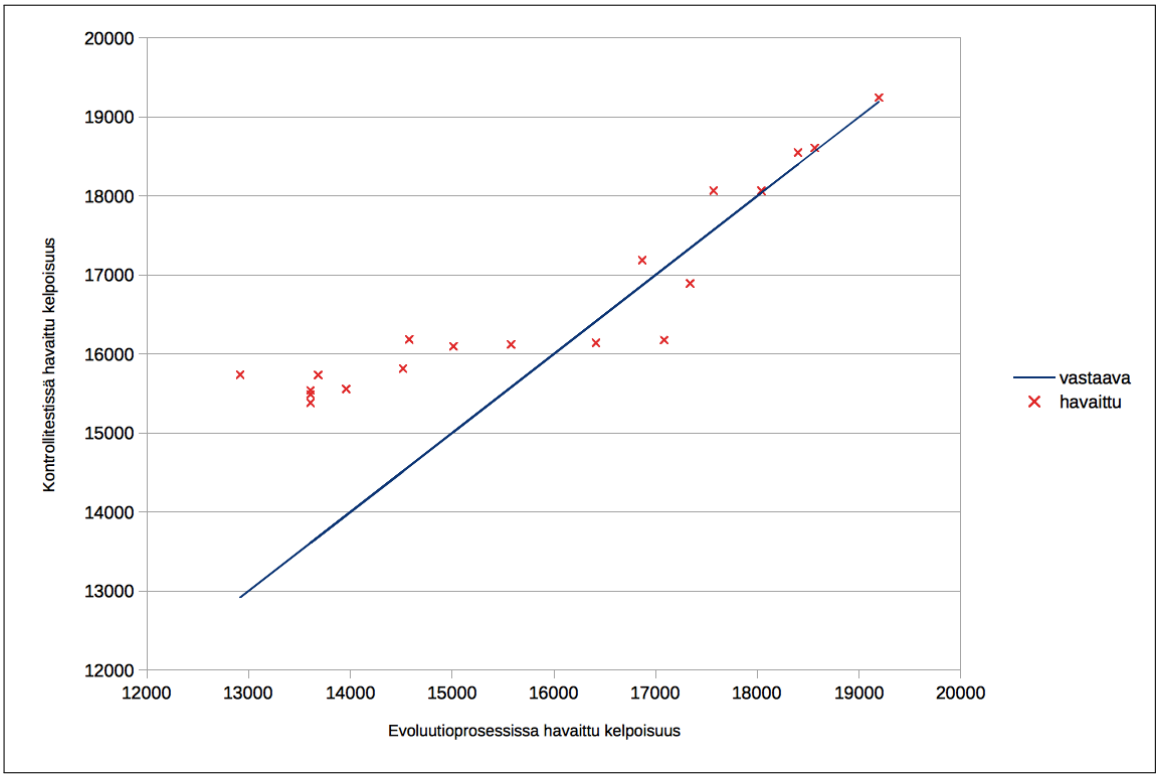
$$(7000 - 5358,5)/(701,67/\sqrt{10}) = 7,4.$$

Jos oletetaan, että kelpoisuuspisteet jakautuvat normaalijakauman mukaisesti, niin todennäköisyys sille, että yhden yksilön kelpoisuus ylittää 7000 pistettä on $6,92 \cdot 10^{-14}$. Todennäköisyys näin suurelle poikkeamalle ainakin yhdessä yksilössä 26:ssa 500 yksilön sukupolvessa on

$$1 - (1 - 6,92 \cdot 10^{-14})^{26*500} = 8,99 \cdot 10^{-10}.$$

Poikkeamia tapahtui kuitenkin useita samassa evoluutioajossa, joten vaikuttaa siltä, että poikkeamien johtuminen satunnaisvaihtelusta on mahdollista sulkea pois.

Testaaminen jälkikäteen toteutettiin kopioimalla evoluutioprosessin tuottamat potentiaalifunktiot kovakoodattuun luokkaan, jonka avulla tiettyä yksilöä voitiin ajaa uudelleen. Virhe, joka olisi johtanut botin erilaiseen käyttäytymiseen evoluutioprosessissa ja jälkikäteen testatessa olisi saattanut selittää kelpoisuuden poikkeamat. Tämä mahdollisuus suljettiin pois luomalla manuaalisesti evoluutiokirjaston yksilöolioita ja vertaamalla samojen potentiaali-



Kuvio 11: Poikkeavuus hyvien yksilöiden kelpoisuudessa.

kenttien käyttäytymistä evoluutioprosessin ja kovakoodatun luokan kautta ajettuna. Potentiaalifunktioiden suoriutumisessa ei havaittu eroa näiden kahden menetelmän välillä.

Kelpoisuuspoikkeamien syytä ei tämän tutkimuksen puitteissa onnistuttu selvittämään. Poikkeamat saattavat olla yksi syy tutkitun menetelmän heikohkoon suoriutumiseen. Jos jokin yksilö saa todellisuutta paremman kelpoisuuden, sen ominaisuudet tulevat esiintymään tulevissa sukupolvissa useammin kuin olisi toivottavaa, jolloin koko populaation todellinen kelpoisuus kärsii.

6.2 Vertailua aiempiin tutkimuksiin

Tässä tutkimuksessa saatuja tuloksia on kiinnostavaa verrata Sandbergin (2011) tutkimukseen. Myös Sandbergilla on ollut tavoitteena kehittää tekoälyratkaisu mikromanagerointiin, tutkimusympäristönä on käytetty StarCraftia ja mikromanageroinnissa on hyödynnetty potentiaalikenttiä. Lisäksi tutkimuksessa käytetyt skenaariot ovat ladattavissa internetistä[1]. Täs-

1. https://code.google.com/p/emapf-starcraft-ai/downloads/list (3.12.2015)

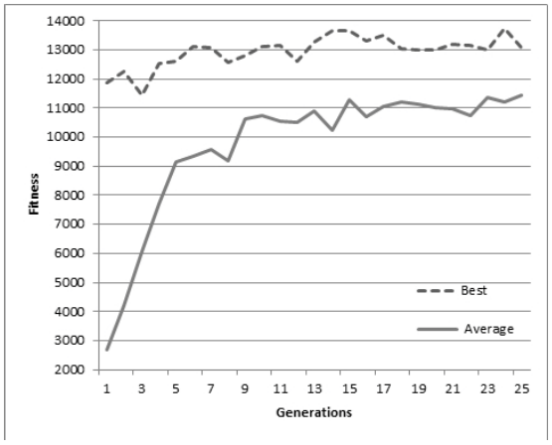
sä tutkimuksessa käytettiin koeajoissa K4, K5, K6 ja K7 Sandbergin goliath-skenaariota. (Koeajoihin K6 ja K7 skenaariota tosin muokattiin siten, että vihollisyksiköt aktivoituvat vasta kun pelaajan yksiköt saapuvat riittävän lähelle.) Sandberg myös saavutti menetelmällään erittäin hyviä tuloksia käyttämissään skenaarioissa, joten hänen tutkimustuloksiinsa vertaaminen on mielekästä.

Kuviossa 12 on Sandbergin koeajoissa tapahtunut kehitys. Hän teki useita kokeiluja eri evaluaatiokertojen määrällä. Sandberg käytti geneettisenä lisääntymisoperaattorina lähes yksinomaan mutaatiota. Vain yhdessä kokeilussa hän käytti lisääntymisoperaattorina myös risteytystä.

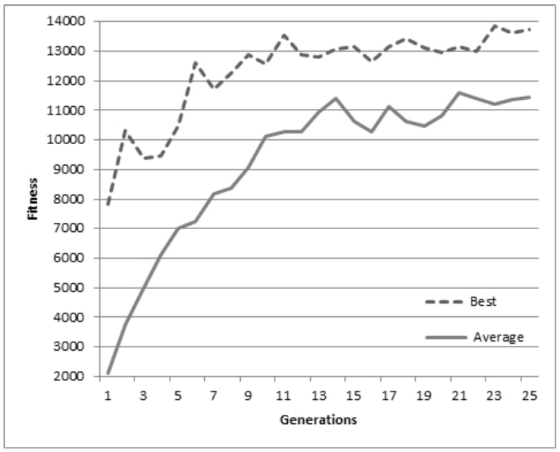
Goliath-skenaarioissa voiton pisteraja on Sandbergin evaluaatiofunktiolla 5600 pistettä. Määritellään tässä voitoksi tilanne, jossa kaikki vastustajan yksiköt saadaan tuhottua. Voitto heikoimmilla mahdollisilla pisteillä saavutetaan, kun sekä kaikki pelaajan omat yksiköt että kaikki vastustajan yksiköt ovat ottelun lopussa tuhoutuneet. Korkeammat pisteet saavutetaan, kun omia yksiköitä säilyy elossa. Voiton raja on laskettu seuraavasti: kartan alussa pelaajalla olevista yhdeksästä goliathista ja yhdestä dronesta 1850 pistettä $(9 \cdot 200 + 50)$ ja 21 tuhotusta hydraliskista ja yhdestä dronesta 7450 pistettä $(21 \cdot 350 + 100)$ sekä -3700 pistettä kaikkien omien yksiköiden tuhouduttua $(2 \cdot 1850)$. 1850 + 7450 - 3700 = 5600. Maksimipisteet tulisivat tuhoamalla kaikki vastustajan yksiköt siten, että kaikki omat yksiköt säilyisivät vahingoittumattomina. Maksimipisteet ovat 21150 pistettä.

Vulture-skenaariossa tekoälypelaajalla on 9 vulturea ja vastustajalla 9 hydraliskia. Voiton raja tässä skenaariossa on samalla tavalla laskettuna 2475 pistettä.

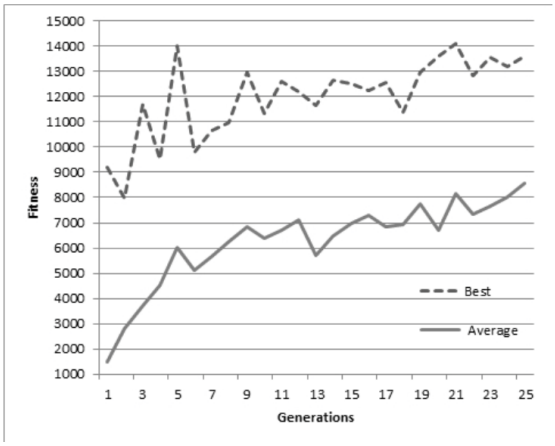
Taulukoissa 12 ja 13 on vertailtu Sandbergin tutkimuksen ja tämän tutkimuksen samassa skenaariossa suoritettujen koeajojen parhaiden yksilöiden kelpoisuuksia. (Tämän tutkimuksen koeajojen tulokset on taulukossa 13 jätetty raaoiksi kelpoisuusluvuiksi Sandbergin kelpoisuuslukujen tapaan vertailun helpottamiseksi. Tämän vuoksi luvut poikkeavat kuvaajan 10 luvuista.) Sandbergin koeajoissa saavutetut parhaiden yksiköiden kelpoisuudet ovat merkittävästi tämän tutkimuksen koeajoissa saavutettuja kelpoisuuksia korkeampia.



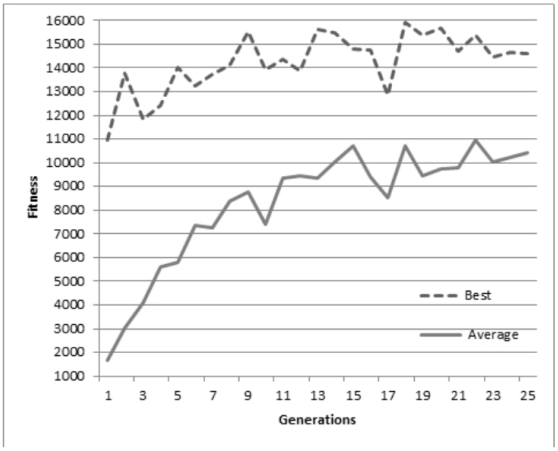
(a) 10 evaluaatiota, mutaatio (goliathit)



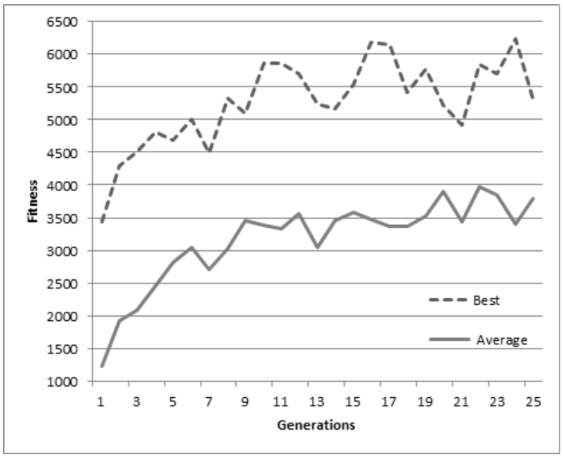
(b) 10 evaluaatiota, mutaatio ja risteytys (goliathit)



(c) 2 evaluaatiota, mutaatio (goliathit)



(d) 1 evaluaatio, mutaatio (goliathit)



(e) 10 evaluaatiota, mutaatio (vulturet)

Kuvio 12: Sandbergin (2011) koeajoissa tapahtunut kelpoisuuden kehitys. Katkoviiva on sukupolven parhaan yksilön kelpoisuus, yhtenäinen viiva sukupolven kaikkien yksilöiden kelpoisuuksien keskiarvo.

Taulukko 12: Sandbergin (2011) parhaiden yksilöiden kelpoisuudet goliath-skenaariossa.

Koeajo	Kelpoisuus
a:	13 800
b:	13 900
c:	14 100
d:	15 900

Taulukko 13: Tämän tutkimuksen parhaiden yksilöiden kelpoisuudet Sandbergin goliath-skenaariossa (2011).

Koeajo	Kelpoisuus
K4:	5 620
K5:	5 105
K6:	3 945
K7:	4 530

7 Johtopäätökset ja jatkotutkimus

Vaikuttaa siltä, että geneettisen ohjelmoinnin käyttäminen RTS-peliä pelaavan, potentiaalikentin ohjautuvan botin optimointiin ei ole toimiva menetelmä ainakaan tutkimuksessa käytössä olleella laitteistolla. Rajallisen laskentatehon ollessa käytössä vaikuttaisi olevan tehokkainta hyödyntää mahdollisimman paljon kohdealuetuntemusta ja rakentaa potentiaalifunktiot mahdollisimman pitkälle valmiiksi käsin. Evoluutiolaskentaa voidaan mahdollisesti hyödyntää näiden käsin suunniteltujen potentiaalifunktioiden parametrien optimointiin.

Olisi mielenkiintoista tutkia, hyötyisikö tutkimuksessa käytetty menetelmä populaatiokoon huomattavasta ylöspäin skaalaamisesta. 500 yksilön populaatiolla evoluutioprosessin ajamiseen kului 8-ytimisellä prosessorilla varustetulla PC:llä, jota käytettiin prosessorin jäähdytysongelmien vuoksi n. 50 % teholla arviolta 5–8 tuntia. Evaluaationopeus oli siis 500 yksilöä/8 h/4 ydintä $\approx$ 16 yksilöä yhdellä ytimellä tunnissa. Amazonin EC2-pilvipalvelussa vuonna 2015 esimerkiksi m3.medium-kone, jossa on yksi virtuaalinen prosessoriydin, 3,75 GB muistia ja 4 GB SSD-kiintolevytilaa, maksaa \$0,13 tunnilta. Miljoonan yksilön populaatiolla yhden sukupolven evaluaatio maksaisi siis $1\,000\,000/16 \cdot \$0,13 \approx \$8\,125$. Jos evoluutioprosessi kestäisi esimerkiksi 50 sukupolvea, nousisivat kustannukset jo yli 400 000 dollariin yhdelle evoluutioprosessille. Tämän tason kustannukset tekisivät menetelmästä todennäköisesti taloudellisesti kannattamattoman, mutta kenties kustannuksia saisi rajattua esimerkiksi hyödyntämällä näytönohjainten rinnakkaista laskentatehoa. Geneettinen ohjelmointi soveltuu rinnakkaiseen suoritusmalliin erittäin hyvin, koska populaatio voidaan jakaa toisistaan riippumattomiin osiin (Andre ja Koza 1998). Näytönohjainten hyödyntäminen vaatisi todennäköisesti sellaisen RTS-pelimoottorin rakentamista, joka olisi suunniteltu toimimaan näytönohjaimella.

Laskentaa voisi yrittää nopeuttaa myös käyttämällä potentiaalifunktioiden evaluointiin itse StarCraftin sijaan David Churchillin kehittämää SparCraft-sovellusta[1], joka simuloi StarCraftin taisteluita, mutta ei esimerkiksi ota yksiköiden välisiä törmäyksiä huomioon, jotta simulointi olisi nopeampaa. SparCraft on myös itse StarCraftista poiketen deterministinen, joten yksilöt tarvitsee sitä käyttäessä evaluoida vain kerran.

1. https://github.com/davechurchill/ualbertabot

Olisi kiinnostavaa tutkia tarkemmin koeajojen hyvien yksilöiden kelpoisuustuloksissa havaittua poikkeavuutta. Mahdollinen lähestymistapa poikkeavuuden tarkempaan havainnointiin voisi olla esimerkiksi tallentaa evoluutioprosessin aikana käydyt ottelut StarCraftin replaytiedostoiksi[2] ja tutkia poikkeavan kelpoisuuden yksilöiden käyttäytymistä otteluissa. Voitaisiin myös tutkia kelpoisuusfunktion kohinan hallintaan kehitettyjen menetelmien vaikutusta tähän poikkeavuuteen. Yksi tällainen menetelmä on esimerkiksi Kalman-laajennettu geneettinen algoritmi (Stroud 2001), jossa yksilöiden kelpoisuuteen yhdistetään tieto varmuudesta, jolla kyseinen kelpoisuus on oikea. Varmuus perustuu yksilön evaluaatioiden hajontaan ja tehtyjen evaluaatiokertojen määrään. Osa evoluutioprosessin aikana suoritettavista evaluaatioista käytetään uusien yksilöiden evaluoinnin sijaan vanhojen yksilöiden uudelleenevaluointiin. Uudelleenevaluoitavaksi pyritään valitsemaan yksiköitä, joiden kelpoisuus on hyvä, mutta kelpoisuuden epävarmuus on suuri. Menetelmää voidaan soveltaa myös tilanteissa, joissa ympäristö, ja siten myös yksilöiden kelpoisuus muuttuu ajan kuluessa, mutta StarCraftin tapauksessa näin ei tapahdu.

Botin hyökkäyskohteiden valintaa ei tässä tutkimuksessa optimoitu. Hyvä hyökkäystrategia voisi olla esimerkiksi Churchillin, Saffidinen ja Buron (2012) esittämä NOK-AV (No-OverKill-Attack-Value), jossa hyökkäyksen kohteeksi valitaan yksikkö, jonka DPF/HP-suhde (Damage Per Frame / Hit Points) on suurin, koordinoiden niin, että yksikköä ei oteta hyökkäyksen kohteeksi, jos siihen on jo kohdistettu tuhoamiseen riittävä määrä hyökkäyksiä.

2. StarCraft mahdollistaa pelattujen otteluiden tallentamisen ja uudelleen katsomisen. Tallennettua ottelua kutsutaan replayksi.

Lähteet

Aarseth, Espen, Solveig Marie Smedstad ja Lise Sunnanå. 2003. “A multidimensional typology of games”. Teoksessa *DiGRA ’03 - Proceedings of the 2003 DiGRA International Conference: Level Up.*

Andre, David, ja John R Koza. 1998. “A parallel implementation of genetic programming that achieves super-linear performance”. *Information Sciences* 106 (3): 201–218.

Arkin, Ronald C. 1987. “Motor schema based navigation for a mobile robot: An approach to programming by behavior”. Teoksessa *Robotics and Automation. Proceedings. 1987 IEEE International Conference on,* 4:264–271. IEEE.

Back, Thomas, Ulrich Hammel ja H-P Schwefel. 1997. “Evolutionary computation: Comments on the history and current state”. *Evolutionary computation, IEEE Transactions on* 1 (1): 3–17.

Balch, Tucker. 1993. “Avoiding the past: A simple but effective strategy for reactive navigation”. Teoksessa *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on,* 678–685. IEEE.

Blizzard Entertainment. 1998. *StarCraft Manual.* Blizzard Entertainment.

———. 2013. “Unit Sizes and Damage”. Viitattu 11. marraskuuta 2013. `http://classic.battle.net/scc/GS/damage.shtml`.

———. 2015a. “High Ground and Cover”. Viitattu 22. marraskuuta 2015. `http://classic.battle.net/scc/GS/hc.shtml`.

———. 2015b. “Mutalisk”. Viitattu 30. marraskuuta 2015. `http://classic.battle.net/scc/zerg/units/mutalisk.shtml`.

———. 2015c. “SCV”. Viitattu 22. marraskuuta 2015. `http://classic.battle.net/scc/terran/uscv.shtml`.

———. 2015d. “Terran Command Center Add-on: ComSat Station”. Viitattu 22. marraskuuta 2015. `http://classic.battle.net/scc/popup/comm.htm`.

Blizzard Entertainment. 2015e. “Terran Special Abilities”. Viitattu 22. marraskuuta 2015. http://classic.battle.net/scc/terran/tspecial.shtml.

———. 2015f. “Terran Unit Stats”. Viitattu 22. marraskuuta 2015. http://classic.battle.net/scc/terran/ustats.shtml.

———. 2015g. “Unit Commands”. Viitattu 22. marraskuuta 2015. http://classic.battle.net/scc/GS/com.shtml.

———. 2015h. “Vulture”. Viitattu 30. marraskuuta 2015. http://classic.battle.net/scc/terran/uv.shtml.

———. 2015i. “Worker”. Viitattu 22. marraskuuta 2015. http://classic.battle.net/scc/popup/worker.htm.

———. 2015j. “Zerg Strategy Guide”. Viitattu 22. marraskuuta 2015. http://classic.battle.net/scc/zerg/units/overlord.shtml.

Buro, Michael. 2003. “Real-Time Strategy Games: A New AI Research Challenge”. *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence:* 1534–1535. https://skatgame.net/mburo/ps/RTS-ijcai03.pdf.

Buro, Michael, ja David Churchill. 2015. “StarCraft AI Competition”. Viitattu 22. marraskuuta 2015. https://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/index.shtml.

“BWAPI Documentation”. 2015. Toukokuu. Viitattu 21. toukokuuta 2015. http://bwapi.github.io.

BWAPI Wiki. 2015. “StarcraftGuide: Advanced Information Guide for Starcraft Broodwar”. Viitattu 9. tammikuuta 2015. https://code.google.com/p/bwapi/wiki/StarcraftGuide.

“BWAPI Wiki: Unit Types in BWAPI”. 2015. Viitattu 11. 28 2015. https://code.google.com/p/bwapi/wiki/UnitTypes.

Čertický, Michal, Paul Paradies, Marek Šuppa, Björn Persson Mattsson, Tomáš Vajda, Rafał Poniatowski ja Sören Klett. 2015. “[SSCAIT] Student StarCraft AI Tournament 2015”. Viitattu 22. marraskuuta 2015. http://sscaitournament.com.

Chan, Hei, Alan Fern, Soumya Ray, Nick Wilson ja Chris Ventura. 2007. “Online Planning for Resource Production in Real-Time Strategy Games.” Teoksessa *ICAPS,* 65–72.

Churchill, David. 2015. “StarCraft: Brood War Attack Animation Frame Data”. Viitattu 6. helmikuuta 2015. https://docs.google.com/spreadsheet/ccc?key=0An3xUNEy2rixdGtleHI4Y2FqUU1wNUthY05leVY2X1E#gid=0.

Churchill, David, ja Michael Buro. 2012. “Incorporating search algorithms into RTS game agents”. Teoksessa *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI).*

Churchill, David, Abdallah Saffidine ja Michael Buro. 2012. “Fast Heuristic Search for RTS Game Combat Scenarios.” Teoksessa *AIIDE.*

Coulom, Rémi. 2007. “Efficient selectivity and backup operators in Monte-Carlo tree search”. Teoksessa *Computers and games,* 72–83. Springer.

Dillon, Beth A. 2008. “Signifying the west: colonialist design in Age of Empires III: The WarChiefs”. *Eludamos. Journal for Computer Game Culture* 2 (1): 129–144.

Evett, Matthew, ja Thomas Fernandez. 1998. “Numeric mutation improves the discovery of numeric constants in genetic programming”. *Genetic Programming:* 66–71.

Furtak, Timothy Michael, ja Michael Buro. 2010. “On the complexity of two-player attrition games played on graphs”. Teoksessa *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference.*

Goldberg, David E, ja Kalyanmoy Deb. 1991. “A comparative analysis of selection schemes used in genetic algorithms”. *Foundations of genetic algorithms* 1:69–93.

Goodrich, Michael A. 2002. “Potential fields tutorial”. Viitattu 3. joulukuuta 2015. http://phoenix.goucher.edu/~jillz/cs325_robotics/goodrich_potential_fields.pdf.

GosuGamers. 2009. “High Terrain Math - StarCraft Strategic forum Forum Discussions”. Viitattu 22. marraskuuta 2015. http://www.gosugamers.net/forums/discussion/344799/high-terrain-math.

Gunnerud, Martin Johansen. 2009. “A CBR/RL system for learning micromanagement in real-time strategy games”. Tutkielma, Norwegian University of Science and Technology.

Hagelbäck, Johan, ja Stefan J Johansson. 2008a. “Dealing with fog of war in a real time strategy game environment”. Teoksessa *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On,* 55–62. IEEE.

———. 2008b. “The Rise of Potential Fields in Real Time Strategy Bots.” *AIIDE* 8:42–47.

———. 2008c. “Using multi-agent potential fields in real-time strategy games”. Teoksessa *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2,* 631–638. International Foundation for Autonomous Agents ja Multiagent Systems.

———. 2009. “A Multi-Agent Potential Field-based Bot for a Full RTS Game Scenario.” Teoksessa *AIIDE.*

Holland, John H. 1975. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.

Huang, H. 2011. “Skynet meets the Swarm: how the Berkeley Overmind won the 2010 StarCraft AI competition”. *Ars Technica* 18.

Khatib, Oussama. 1986. “Real-time obstacle avoidance for manipulators and mobile robots”. *The international journal of robotics research* 5 (1): 90–98.

Koza, John R. 1992. *Genetic programming: on the programming of computers by means of natural selection.* Nide 1. MIT press.

Liquipedia. 2015a. “Damage - Liquipedia Starcraft Wiki”. Viitattu 22. marraskuuta 2015. http://wiki.teamliquid.net/starcraft/Damage.

———. 2015b. “Micromanagement”. Viitattu 7. tammikuuta 2015. http://wiki.teamliquid.net/starcraft/Micromanagement.

Liu, Siming, Sushil J Louis ja Monica Nicolescu. 2013. “Comparing heuristic search methods for finding effective group behaviors in RTS game”. Teoksessa *Evolutionary Computation (CEC), 2013 IEEE Congress on,* 1371–1378. IEEE.

Luke, Sean. 2014. “The ECJ Owner’s Manual”. Viitattu 9. helmikuuta 2015. `http://cs.gmu.edu/~eclab/projects/ecj/docs/manual/manual.pdf`.

Miller, Brad L, ja David E Goldberg. 1995. “Genetic algorithms, tournament selection, and the effects of noise”. *Complex Systems* 9 (3): 193–212.

Nguyen, Kien Quang, Zhe Wang ja Ruck Thawonmas. 2013. “Potential flows for controlling scout units in StarCraft”. Teoksessa *Computational Intelligence in Games (CIG), 2013 IEEE Conference on,* 1–7. IEEE.

Nguyen, Tung, Kien Nguyen ja Ruck Thawonmas. 2013. “Potential flow for unit positioning during combat in StarCraft”. Teoksessa *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on,* 10–11. IEEE.

Rasul, Saqib. 2010. “An Improved RMI Tutorial with Eclipse”. Viitattu 27. helmikuuta 2015. `http://code.nomad-labs.com/2010/03/26/an-improved-rmi-tutorial-with-eclipse/`.

Rathe, Espen Auran, ja Jørgen Bøe Svendsen. 2012. “Micromanagement in Starcraft using potential fields tuned with a multi-objective genetic algorithm”. Tutkielma, Norwegian University of Science and Technology.

Russell, Stuart J, ja Peter Norvig. 2010. *Artificial Intelligence: a Modern Approach.* Third Edition. Prentice Hall.

Sandberg, Thomas Willer. 2011. “Evolutionary Multi-Agent potential field based AI approach for SSC scenarios in RTS games”. Tutkielma, IT University Copenhagen.

Sipser, Michael. 1997. *Introduction to the Theory of Computation.* PWS Publishing Company.

“StarCraft AI Competition Results”. 2014. Viitattu 28. marraskuuta 2015. `http://webdocs.cs.ualberta.ca/~scbw/2014/`.

Stroud, Phillip D. 2001. “Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations”. *Evolutionary Computation, IEEE Transactions on* 5 (1): 66–77.

Synnaeve, Gabriel, ja Pierre Bessiere. 2011. “A Bayesian model for RTS units control applied to StarCraft”. Teoksessa *Computational Intelligence and Games (CIG), 2011 IEEE Conference on,* 190–196. IEEE.

Szczepański, Tomasz, ja Agnar Aamodt. 2008. “Case-based reasoning for improved micromanagement in Real-time strategy games.” *Project Report NTNU-IDI.*

Thurau, Christian, Christian Bauckhage ja Gerhard Sagerer. 2004. “Learning human-like movement behavior for computer games”. *From animals to animats* 8:315–323.

Weber, Ben. 2012. “Integrating learning in a multi-scale agent”. Tohtorinväitöskirja, University of California, Santa Cruz.

Weber, Ben G, Michael Mateas ja Arnav Jhala. 2011. “Building human-level ai for real-time strategy games”. Teoksessa *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems,* 329–336. `http://www.aaai.org/ocs/index.php/FSS/FSS11/paper/viewFile/4209/4567`.

Weber, Ben G, ja Santiago Ontanón. 2010. “Using automated replay annotation for case-based planning in games”. Teoksessa *International Conference on Case-based Reasoning Workshop on CBR for Computer Games,* 15–24. Citeseer.

Weber, Ben George, Peter Mawhorter, Michael Mateas ja Arnav Jhala. 2010. “Reactive planning idioms for multi-scale game AI”. Teoksessa *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on,* 115–122. IEEE.

Whitley, L Darrell, ym. 1989. “The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best.” Teoksessa *ICGA,* 116–123.

Yildirim, Sule, ja Sindre Berg Stene. 2008. “A survey on the need and use of AI in game agents”. Teoksessa *Proceedings of the 2008 Spring simulation multiconference,* 124–131. Society for Computer Simulation International.

Liitteet

A Koeajojen versiotarkisteet

Taulukossa 14 olevien versiotarkisteiden (engl. *commit hash*) perusteella versionhallinnasta voidaan hakea lähdekoodista se versio, jota on käytetty kussakin koeajossa.

Taulukko 14: Skenaarioden versiotarkisteet

#	Tarkiste
1	a3a812fd3a678d333fdcb8631c2c4769e26fe07f
2	7fab9e143fde48c37e03400a0e740c1a47610355
3	9e119f4ff4a2957e65205326d557e4ce8a8d88ff
4	29f94efd76fb7227e54f33bc8d78112bbbc0326c
5	bb947b92c2f8f722fb6912d0953dab57e5939124
9	959d2fb96dd8fe1f39ea709fb4be204e59cdc998
7	0c5929be1ece2a3d2d079c7f6d34135935e05f6b

B Käytetyt kirjastojen versiot

Taulukossa 15 esitetään, mitä versioita eri kolmannen osapuolen komponenteista tutkimuksessa käytettiin.

Taulukko 15: Työssä käytettyjen ohjelmistojen ja kirjastojen versiot.

Kirjasto	Versio
ECJ	22
BWAPI	3.7.4
Java	1.7
BWMirror	1.0
StarCraft	1.16

C Lähdekoodi

Joitain alustaviin testauksiin ja kokeiluihin liittyviä tiedostoja on jätetty pois. Versionhallinnan varasto, jossa ovat mukana myös liitteestä pois jätetyt tiedostot on saatavilla osoitteessa `https://github.com/Hemaolle/thesis`.

Listaus 7.1: Lähdekoodin hakemistorakenne

```
+---bot
|       Controller.java
|       PosUtils.java
|       PotentialCalculator.java
|       Visualizer.java
|
+---evolution
|   |   DoubleData.java
|   |   PrintEntirePopulationStatistics.java
|   |   thesis.params
|   |   ThesisProblem.java
|   |
|   \---nodes
|           Add.java
|           BigFloatERC.java
|           Cos.java
|           Cube.java
|           Div.java
|           Exp.java
|           FloatERC.java
|           Iflte.java
|           Log.java
|           Mul.java
|           NegExp.java
|           Pow.java
|           Sin.java
|           Sqrt.java
|           Square.java
|           Sub.java
|           Tan.java
```

```
|           Tanh.java
|           U.java
|           W.java
|           X.java
|           Y.java
|           Z.java
|
\---rmi
        PolicyFileLocator.java
        PotentialFunctionProvider.java
        RemoteBotInterface.java
        RemoteBotRMI.java
        RmiStarter.java
        ThesisProblemRMI.java
```

Listaus 7.2: bot/Controller.java

```
1  package thesis.bot;
2
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.io.PrintWriter;
8  import java.util.ArrayList;
9  import java.util.HashMap;
10 import java.util.List;
11 import java.util.Map.Entry;
12
13 import thesis.rmi.PotentialFunctionProvider;
14 import thesis.rmi.RemoteBotInterface;
15 import bwapi.*;
16 import bwta.*;
17
18 // TODO: Add more objects that generate potential fields.
19 /**
20  * Oskari Leppäaho's master's thesis bot.
21  * <p>
```

```
22   * Player's own units move guided by potential fields generated by the objects
23   * in the game such as enemy units and map edges. When in enemy unit range,
24   * player's units will attack them if their current potential is low enough.
25   * During the cooldown the units continue to move.
26   * <p>
27   * The game will be run in a map that resets the combat scenario after one of
28   * the teams has lost all of it's units. The map will signal the bot about this
29   * by setting the minerals to 500 for a moment. The bot will calculate the score
30   * of the run at that point. (This might not be the best solution. It might be
31   * possible also to just use game.restartGame() to restart the scenario but
32   * JNIBWAPI doesn't currently support that and I'm not sure if that option skips
33   * the menus. Going through the menus takes a lot of time.)
34   * <p>
35   * The client runs in its own thread and getRoundScore method runs in another
36   * thread, polling for the flag that signals the ending of a round.
37   *
38   * @author Oskari Leppäaho
39   */
40  public class Controller extends DefaultBWListener implements Runnable,
41      RemoteBotInterface {
42
43    // TODO: Is there a reason for this to be public?
44    /** Indicates if the round score for the current round has been calculated. */
45    public volatile boolean isRoundResultRetrievable = false;
46
47    /**
48     * Determines if debug information should be displayed. Enables printing
49     * potential values of the points around an unit on the game window. Also
50     * enables graphical visualizations.
51     */
52    final static boolean DEBUG_INFO = false;
53    /**
54     * The maximum distance of the move command from the unit position.
55     */
56    final static int MOVE_DISTANCE = 1;
57    /** Determines the game speed in frames per second. */
58    final static int GAME_SPEED = 0;
```

```
59    /**
60     * Determines the maximum length for a game (in frames). The game will be
61     * restarted if it doesn't end before the maximum number of frames has
62     * passed.
63     */
64    final static int GAME_MAX_LENGTH = 2000;
65    /**
66     * Determines if performance statistics should be written to three files:
67     * victories_%name%.txt, surviving_units_%name%.txt and score_%name%.txt
68     * where %name% is the value of the name variable. For post
69     * evolution evaluation.
70     */
71    final static boolean WRITE_STATS_TO_FILE = true;
72    /**
73     * The name of the bot. Will be displayed on the game window when the bot is
74     * running and in the command window as the bot is started.
75     */
76    final static String BOT_NAME = "Thesis bot v5";
77    /**
78     * Determines if the StarCraft client should draw the game on the screen
79     * every logical game frame. Disabling the GUI should speed the game up.
80     */
81    final static boolean GUI_ON = true;
82    final static int MAX_SCORE = 21150;
83    /**
84     * If the notification about a round end has been registered (the map used
85     * should set minerals to 500 to notify about a round end).
86     */
87    boolean hasBeenRoundEndRegistered = false;
88    /**
89     * Indicates if the isStartingAttack has already changed to true after a
90     * unit has been issued an attack command. The key is the unit ID.
91     */
92    HashMap<Integer, Boolean> isAttackInProgress = new HashMap<Integer, Boolean>()
          ;
93    /**
94     * Used to avoid giving a move order before the attack of a unit has been
```

```
95      * finished. The key is the unit ID.
96      */
97     HashMap<Integer, Boolean> hasAttackOrderBeenGiven = new HashMap<Integer,
           Boolean>();
98     /**
99      * Holds the unit's HP value relative to group's average.
100     */
101    HashMap<Unit, Double> relativeHps = new HashMap<Unit, Double>();
102    /**
103     * If enabled, the game stops in every frame and only proceeds by pressing
104     * enter in the command line.
105     */
106    boolean isStepThroughEnabled = false;
107    // TODO: could maybe be final.
108    /** BroodWar AI runs in this thread. */
109    Thread bwThread;
110    // TODO: Should this be volatile?
111    /**
112     * Fitness for the last round. Calculated in {@link #calculateScore()}.
113     **/
114    int score;
115
116    // TODO: seems to be meaningless ATM, not read anywhere.
117    /**
118     * Unused, indicates if the evolution has started (all threads have been
119     * setup).
120     */
121    boolean hasEvolutionStarted = false;
122    /** Indicates if the StarCraft match has started. */
123    boolean hasMatchStarted = false;
124
125    /** Draws the visualizations on the game window */
126    private Visualizer visualizer;
127    /** Handles potential field calculations */
128    final private PotentialCalculator potentialCalculator;
129    // Moving this much past the point where we want to actually be.
130          // Unit acceleration and breaking causes it to lag behind the
```

```
131         // position of the move command given.
132     /**
133      * Moving this much past the point where we want to actually be.
134      * Unit acceleration and breaking causes it to lag behind the
135      * position of the move command given.
136      */
137     final static int MOVE_COMMAND_OFFSET = 20;
138 
139     /** The BWMirror. Makes communication with BWAPI possible. */
140     private Mirror mirror = new Mirror();
141     /** The BroodWar game from BWMirror */
142     Game game;
143     List<Unit> myUnitsNoRevealers = new ArrayList<Unit>();
144     List<Unit> enemyUnitsNoRevealers = new ArrayList<Unit>();
145     private String name;
146 
147     /**
148      * Creates the AI client and gets results from it indefinitely. First command
149      * line argument is given as the name for the client.
150      *
151      * @throws InterruptedException
152      *         If interrupted.
153      */
154     public static void main(String[] args) throws InterruptedException {
155       String name = "Default";
156       if (args.length > 0) {
157         name = args[0];
158       }
159       Controller client = new Controller(name);
160       int i = 1;
161       while (true) {
162         System.out.println("Round " + i + ". Score: "
163             + client.getRoundScore());
164         ++i;
165       }
166       // client.finish();
167     }
```

```
168
169   /**
170    * Instantiates the JNI-BWAPI interface and connects to BWAPI.
171    */
172   public Controller(String name) {
173     this.name = name;
174     mirror.getModule().setEventListener(this);
175     potentialCalculator = new PotentialCalculator(this);
176     WaitForConnection();
177   }
178
179   /**
180    * Waits for StarCraft to be launched and the match to begin.
181    */
182   public void WaitForConnection() {
183     if (bwThread == null) {
184       bwThread = new Thread(this);
185       bwThread.start();
186     }
187     while (true) {
188       try {
189         Thread.sleep(100);
190       } catch (InterruptedException e) {
191         System.err.println("WaitForConnection of the controller was "
192             + "interrupted");
193         e.printStackTrace();
194       }
195       if (hasMatchStarted) {
196         return;
197       }
198     }
199   }
200
201   /**
202    * TODO: A mechanism that gives the score only after the first full round.
203    * Currently the bot will run using the default potential function until
204    * this version of the getRoundScore is called after which it will return
```

```
205     * the end score of that round event though for the beginning of the round
206     * the default potential function was used.
207     */
208
209    /**
210     * Returns when a single round of fighting has finished. <strike>Also starts
211     * the client if it hasn't been started yet.</strike>
212     *
213     * @param potentialProvider
214     *      The object that will provide the potential function for this
215     *      round.
216     * @return Score for the round.
217     */
218    public int getRoundScore(PotentialFunctionProvider potentialProvider) {
219      hasEvolutionStarted = true;
220      potentialCalculator.setPotentialProvider(potentialProvider);
221      while (true) {
222        try {
223          Thread.sleep(100);
224        } catch (InterruptedException e) {
225          System.err.println("Get round score waiting interrupted:");
226          e.printStackTrace();
227        }
228        if (isRoundResultRetrievable) {
229          isRoundResultRetrievable = false;
230          return score;
231        }
232      }
233    }
234
235    /**
236     * Returns when a single round of fighting has finished. Also starts the
237     * client if it hasn't been started yet.
238     *
239     * @return Score for the round.
240     * @throws InterruptedException
241     *       If interrupted.
```

```
242     */
243   public int getRoundScore() throws InterruptedException {
244     hasEvolutionStarted = true;
245     // Is the starting of the thread required anymore? Looks like it is
246     // already started in {@link #WaitForConnection()}.
247     if (bwThread == null) {
248       bwThread = new Thread(this);
249       bwThread.start();
250     }
251     while (true) {
252       Thread.sleep(100);
253       if (isRoundResultRetrievable) {
254         isRoundResultRetrievable = false;
255         System.out.println("Max score " + MAX_SCORE);
256         System.out.println("Score: " + score);
257         return score;
258       }
259     }
260   }
261
262   /**
263    * Starts the bot.
264    */
265   public void run() {
266     System.out.println("Starting " + BOT_NAME);
267     mirror.startGame();
268   }
269
270   /**
271    * Finishes running the games. Tells the bot to leave the game.
272    */
273   public void finish() {
274     bwThread.interrupt();
275   }
276
277   /**
278    * Called at the beginning of a game. Sets some options for StarCraft and
```

```
279     * gets the reference to BWMirror Game object. Also creates the visualizer.
280     */
281    @Override
282    public void onStart() {
283      game = mirror.getGame();
284      visualizer = new Visualizer(game, this, DEBUG_INFO);
285
286      final int ENABLECODE_PERFECT_INFORMATION = 0;
287      // final int ENABLECODE_USER_INPUT = 1;
288
289      // game.enableFlag(ENABLECODE_USER_INPUT);
290      game.enableFlag(ENABLECODE_PERFECT_INFORMATION);
291      game.setLocalSpeed(GAME_SPEED);
292
293      if (!GUI_ON)
294        game.setGUI(false);
295      hasMatchStarted = true;
296
297      hasBeenRoundEndRegistered = false;
298
299      isAttackInProgress.clear();
300      hasAttackOrderBeenGiven.clear();
301      relativeHps.clear();
302
303      BWTA.readMap();
304      BWTA.analyze();
305    }
306
307    /**
308     * Called each game cycle. Handles stepping frame by frame and leaving the
309     * game when requested. Draws the name of the bot on the screen and
310     * highlights the units. Then loops AI player's units giving them orders.
311     */
312    @Override
313    public void onFrame() {
314      cacheVariables();
315      populateUnitHashMaps();
```

```
316       cacheRelativeHps();
317
318       // Comment: tried to wait for the first request from evolution
319       // while (!isStarted) {
320       // try {
321       // Thread.sleep(1000);
322       // System.out.println("Waiting for the first score request");
323       // } catch (InterruptedException e) {
324       // System.err.println("Match frame interrupted while waiting"
325       // + " for start signal");
326       // e.printStackTrace();
327       // }
328       // }
329       // --------------debug
330       // System.out.println("Frame " + bwapi.getFrameCount());
331       // -------------enddebug
332       if (isStepThroughEnabled) {
333         waitForEnterOnConsole();
334       }
335       if (Thread.currentThread().isInterrupted()) {
336         game.leaveGame();
337       }
338       game.drawTextScreen(0, 20, BOT_NAME);
339       game.drawTextScreen(0, 30, "" + game.getFrameCount());
340       checkForRoundEnd();
341       visualizer.highlightUnits();
342       for (Unit u : myUnitsNoRevealers) {
343         handleUnit(u);
344       }
345     }
346
347     private void cacheRelativeHps() {
348       double averageRelativeHpLeft = 0;
349       double hpLeft = 0;
350       double maxHp = 0;
351       for (Unit u : getMyUnitsNoRevealers()) {
352         hpLeft = 0;
```

```
353         maxHp = 0;
354         hpLeft += u.getHitPoints();
355         hpLeft += u.getShields();
356         maxHp += u.getType().maxHitPoints();
357         maxHp += u.getType().maxShields();
358         averageRelativeHpLeft += hpLeft / maxHp;
359       }
360       averageRelativeHpLeft /= getMyUnitsNoRevealers().size();
361       for (Entry<Unit, Double> entry : relativeHps.entrySet()) {
362         hpLeft = 0;
363         maxHp = 0;
364         hpLeft += entry.getKey().getHitPoints();
365         hpLeft += entry.getKey().getShields();
366         maxHp += entry.getKey().getType().maxHitPoints();
367         maxHp += entry.getKey().getType().maxShields();
368         entry.setValue((hpLeft / maxHp) / averageRelativeHpLeft);
369       }
370     }
371
372     public double getRelativeHp(Unit u) {
373       return relativeHps.get(u);
374     }
375
376     /**
377      * Caches the lists of own and enemy units to avoid calls to BWAPI.
378      */
379     private void cacheVariables() {
380       myUnitsNoRevealers = removeRevealersFromUnitSet(game.self().getUnits());
381       enemyUnitsNoRevealers = removeRevealersFromUnitSet(game.enemy()
382           .getUnits());
383     }
384
385     /**
386      * Checks if the HashMaps containing information about the units' attack
387      * states have been filled for the AI player's units and fills them if they
388      * are empty.
389      */
```

```
390   private void populateUnitHashMaps() {
391     if (hasAttackOrderBeenGiven.isEmpty())
392       for (Unit u : getMyUnitsNoRevealers())
393         hasAttackOrderBeenGiven.put(u.getID(), false);
394
395     if (isAttackInProgress.isEmpty())
396       for (Unit u : getMyUnitsNoRevealers())
397         isAttackInProgress.put(u.getID(), false);
398
399     if (relativeHps.isEmpty())
400       for (Unit u : getMyUnitsNoRevealers())
401         relativeHps.put(u, 1d);
402   }
403
404   /**
405    * Waits for the user to press enter on the console.
406    */
407   private void waitForEnterOnConsole() {
408     try {
409       System.in.read();
410     } catch (IOException e) {
411       System.err.println("Something went wrong when waiting for \n"
412           + "the user to press enter on the console\n"
413           + "(for stepping).");
414       e.printStackTrace();
415     }
416   }
417
418   /**
419    * Check if a round has ended. Handle round ending: set and reset
420    * appropriate flags for the new round and calculate the score. The map
421    * should signal about an end of round by setting the AI player's minerals
422    * to 500. At the start of a new round the minerals will be reset to 0.
423    */
424   private void checkForRoundEnd() {
425     // It takes one frame to enable the perfect information flag so the
426     // enemy units can't be seen in the first frame. If we called
```

```
427     // checkForRoundEnd in the first frame, it would think that the enemy
428     // has been defeated.
429     if (game.getFrameCount() == 0)
430       return;
431     if (!isRoundResultRetrievable
432         && !hasBeenRoundEndRegistered
433         && ((getMyUnitsNoRevealers().size() == 0 || getEnemyUnitsNoRevealers()
434             .size() == 0) || game.getFrameCount() > GAME_MAX_LENGTH)) {
435       hasBeenRoundEndRegistered = true;
436       isRoundResultRetrievable = true;
437
438       isStepThroughEnabled = false;
439
440       calculateScore();
441     }
442
443     // This means that the round has ended and the round result has been
444     // retrieved. Time to start a new round.
445     if (hasBeenRoundEndRegistered && !isRoundResultRetrievable
446         && hasMatchStarted) {
447       game.restartGame();
448       hasMatchStarted = false;
449     }
450   }
451
452   /**
453    * Calculates the score for this round. The score is calculated by adding
454    * shields and hit points of remaining friendly units and subtracting the
455    * shields and hit points of remaining enemy units.
456    */
457   private void calculateScore() {
458     score = 0;
459     score += game.self().getUnitScore();
460     score += game.self().getKillScore();
461     //score -= game.enemy().getKillScore();
462     int totalOwnHitpointsShieldsLeft = 0;
463     if (!(game.getFrameCount() > GAME_MAX_LENGTH)) {
```

```
464         totalOwnHitpointsShieldsLeft = 0;
465         for (Unit u : game.self().getUnits()) {
466           totalOwnHitpointsShieldsLeft += u.getShields();
467           totalOwnHitpointsShieldsLeft += u.getHitPoints();
468         }
469         totalOwnHitpointsShieldsLeft *= 10;
470         score += totalOwnHitpointsShieldsLeft;
471       } else {
472         if (game.self().getKillScore() > 0)
473           appendToFile("endlogs" + File.separator + name + ".txt",
474               "Framecount: " + game.getFrameCount() + " Own units: "
475                   + getMyUnitsNoRevealers().size()
476                   + " Enemy units: "
477                   + getEnemyUnitsNoRevealers().size());
478       }
479       // System.out.println("Unit score: " + game.self().getUnitScore() +
480       // " Enemy kill score: " + game.enemy().getKillScore());
481       // System.out.println("Own unit count: " +
482       // getMyUnitsNoRevealers().size() + " Enemy unit count: " +
483       // getEnemyUnitsNoRevealers().size());
484       // System.out.println("TotalOwnHitpointsShieldsLeft: " +
485       // totalOwnHitpointsShieldsLeft);
486       // System.out.println("Score: " + score);
487       score -= MAX_SCORE;
488       score *= -1;
489       System.out.println("End frames: " + game.getFrameCount());
490       if (WRITE_STATS_TO_FILE)
491         writeStatsToFile();
492     }
493
494     /**
495      * Writes performance statistics to three files: victories_%name%.txt,
496      * surviving_units_%name%.txt and score_%name%.txt where %name% is
497      * the value of the name variable.
498      */
499     private void writeStatsToFile() {
500       appendToFile("victories_" + name + ".txt",
```

```
501         (getEnemyUnitsNoRevealers().size() < getMyUnitsNoRevealers()
502             .size()) ? "1" : "0");
503     appendToFile(
504         "surviving_units_" + name + ".txt",
505         (" Own: " + getMyUnitsNoRevealers().size() + " Enemy: " +
                getEnemyUnitsNoRevealers()
506             .size()));
507     appendToFile("scores_" + name + ".txt", "" + score);
508   }
509
510   /**
511    * Appends a line of text to a file.
512    *
513    * @param filePath
514    *       Path to the file.
515    * @param line
516    *       The line to append.
517    */
518   private void appendToFile(String filePath, String line) {
519     try (PrintWriter out = new PrintWriter(new BufferedWriter(
520         new FileWriter(filePath, true)))) {
521       out.println(line);
522     } catch (IOException e) {
523       System.out.println("Error when printing to myfile.txt: "
524           + e.getMessage());
525     }
526   }
527
528   /**
529    * Return a list of my units. Ignore map revealers.
530    *
531    * @return A list of my units without map revealers.
532    */
533   public List<Unit> getMyUnitsNoRevealers() {
534     return myUnitsNoRevealers;
535   }
536
```

```
537   /**
538    * Return a list of enemy units. Ignore map revealers.
539    *
540    * @return A list of enemy units without map revealers.
541    */
542   public List<Unit> getEnemyUnitsNoRevealers() {
543     return enemyUnitsNoRevealers;
544   }
545
546   /**
547    * Removes map revealer units from a list of units.
548    *
549    * @param unitList
550    *       Unit list to remove map revealers from.
551    * @return Unit list with map revealers removed.
552    */
553   private List<Unit> removeRevealersFromUnitSet(List<Unit> unitList) {
554     List<Unit> units = new ArrayList<Unit>();
555     for (Unit unit : unitList) {
556       if (unit.getType() != UnitType.Terran_SCV)
557         units.add(unit);
558     }
559     return units;
560   }
561
562   /**
563    * Should be Called in each game cycle for every unit. Controls the AI
564    * player's units. If some enemy units are in range and the current
565    * potential is large enough (the unit is fairly comfortable with its
566    * current position), the unit will shoot at some enemy unit that is in
567    * range. Any move orders won't be given before an attack has finished. When
568    * not attacking, moves the units based on each unit's potential fields.
569    *
570    * @param u
571    *       The unit to be handled.
572    */
573   private void handleUnit(Unit u) {
```

```
574     Position moveTo = null;
575     try {
576       double currentPotential;
577       double[] potentials = potentialCalculator.getPotentialsAround(u);
578       Position moveDirection = potentialCalculator
579           .getMoveDirection(potentials);
580       visualizer.drawPotentialValues(potentials);
581       if (isAttacking(u)) {
582         return;
583       }
584       currentPotential = potentialCalculator.getPotential(
585           u.getPosition(), u);
586       if (u.getGroundWeaponCooldown() == 0 && currentPotential > 0) {
587         if (attackFirstEnemy(u)) {
588           return;
589         }
590       }
591       moveTo = potentialCalculator.getHighestPotentialPosition(
592           u.getPosition(), moveDirection, MOVE_DISTANCE, u);
593
594       u.move(offsetPosition(moveTo, moveDirection, MOVE_COMMAND_OFFSET), false);
595     } finally {
596       visualizer.visualizeDestination(u, moveTo,
597           getHasAttackOrderBeenGiven(u) || getIsAttackInProgress(u));
598     }
599   }
600
601   /**
602    * Indicates if a unit has been issued an attack command and hasn't finished
603    * attacking.
604    *
605    * @param u
606    *      Unit to check.
607    * @return True if the unit has been issued an attack command and it hasn't
608    *     finished attacking.
609    */
610   private boolean isAttacking(Unit u) {
```

```
611     /*
612      * After the unit has been issued an attack order it takes a few frames
613      * before its state is isStartingAttack. The order of the unit will
614      * however change immediately. But the order won't change after the
615      * attack has ended. attackInProgress tells if the isStartingAttack has
616      * already changed to true. The attack ends when isAttackFrame and
617      * isStartingAttack are false but attackInProgress is yet true so then
618      * attackInProgress will be set to false.
619      */
620
621     // System.out.println("AttackCheck: "
622     // + (u.isAttackFrame() || u.isStartingAttack() || (!attackInProgress &&
623     // u.getOrder() == OrderType.OrderTypes.AttackUnit))
624     // + " isAttackFrame: " + u.isAttackFrame() + " isStartingAttack: "
625     // + u.isStartingAttack() + " attackInProgress: " + attackInProgress
626     // + " orderType: " + u.getOrder().getName());
627     // //+ " orderIsAttack: " + (u.getOrder() ==
628     // OrderType.OrderTypes.AttackUnit));
629     // if( u.isAttackFrame() || u.isStartingAttack() || (!attackInProgress
630     // && u.getOrder() == OrderType.OrderTypes.AttackUnit)) {
631
632     if (u.isAttackFrame() || u.isStartingAttack()
633         || (!getIsAttackInProgress(u) && getHasAttackOrderBeenGiven(u))) {
634       // Attack order has been given and attack hasn't finished yet.
635       if (!getIsAttackInProgress(u) && (u.isStartingAttack())) {
636         // First frame of the actual attack.
637         isAttackInProgress.put(u.getID(), true);
638         hasAttackOrderBeenGiven.put(u.getID(), false);
639       }
640       return true;
641     } else if (getIsAttackInProgress(u)) {
642       // First frame after the attack.
643       isAttackInProgress.put(u.getID(), false);
644     }
645
646     // No attacking in progress.
647     return false;
```

```
648   }
649
650   /**
651    * Indicates if the unit has an attack in progress.
652    *
653    * @param u
654    *      The unit to check.
655    * @return True if the unit has an attack in progress.
656    */
657   private boolean getIsAttackInProgress(Unit u) {
658     return isAttackInProgress.get(u.getID()).booleanValue();
659   }
660
661   /**
662    * Indicates if the unit has been given an attack order.
663    *
664    * @param u
665    *      The unit to check.
666    * @return True if the unit has been given an attack order.
667    */
668   private boolean getHasAttackOrderBeenGiven(Unit u) {
669     return hasAttackOrderBeenGiven.get(u.getID()).booleanValue();
670   }
671
672   /**
673    * Orders the given unit to attack the first enemy unit that is in its
674    * ground weapon's range.
675    *
676    * @param u
677    *      The unit that will be given the attack order.
678    * @return If an enemy was found in range and was attacked.
679    */
680   private boolean attackFirstEnemy(Unit u) {
681     boolean attack = false;
682
683     for (Unit eu : getEnemyUnitsNoRevealers()) {
684       if (u.isInWeaponRange(eu)) {
```

```
685             hasAttackOrderBeenGiven.put(u.getID(), true);
686             u.attack(eu, false);
687             attack = true;
688             visualizer.setAttackTarget(u, eu);
689           }
690         }
691         return attack;
692       }
693
694       /**
695        * Adds direction * distance to position.
696        *
697        * @param position
698        *      Position to add to.
699        * @param direction
700        *      Direction to add. X and Y must be in [-1, 1]
701        * @param distance
702        *      Distance to add
703        * @return Position + direction * distance
704        *
705        */
706       Position offsetPosition(Position position, Position direction, int distance) {
707         if (direction.getX() < -1 || 1 < direction.getX()
708             || direction.getY() < -1 || 1 < direction.getY())
709           throw new IllegalArgumentException(
710               "Illegal argument direction in offsetPosition: "
711                   + direction);
712         if ((direction.getX() * direction.getX() + direction.getY()
713             * direction.getY()) == 2) {
714           distance = (int) (distance / Math.sqrt(2.0));
715         }
716         Position overMoveVector = new Position((int) Math.round((direction
717             .getX() * distance)),
718             (int) Math.round((direction.getY() * distance)));
719
720         Position newPosition = PosUtils.add(position, overMoveVector);
721         return newPosition;
```

```
722   }
723
724   /**
725    * Finds the highest index in an array of doubles. If there are multiple
726    * highest values and the one in index 4 is one of them, default to index 4.
727    *
728    * @param array
729    *      An array of doubles.
730    * @return The index of the highest value. Default to index 4.
731    */
732   static int findHighestDefaultTo4(double[] array) {
733     int highestIndex = findHighest(array);
734     if (array[highestIndex] == array[4])
735       highestIndex = 4;
736     return highestIndex;
737   }
738
739   /**
740    * Find the index of the highest value in a array of doubles.
741    *
742    * @param array
743    *      An array of doubles.
744    * @return The index of the highest value.
745    */
746   private static int findHighest(double[] array) {
747     double highest = array[0];
748     int highestIndex = 0;
749     for (int i = 0; i < array.length; i++) {
750       if (highest < array[i]) {
751         highest = array[i];
752         highestIndex = i;
753       }
754     }
755     return highestIndex;
756   }
757
758   @Override
```

```
759   public void onUnitDestroy(Unit unit) {
760     isAttackInProgress.remove(unit.getID());
761     hasAttackOrderBeenGiven.remove(unit.getID());
762     relativeHps.remove(unit);
763   }
764 }
```

Listaus 7.3: bot/PosUtils.java

```
1  package thesis.bot;
2
3  import bwapi.Position;
4
5  /**
6   * Provides some vector math operations on bwapi Positions.
7   *
8   * @author Oskari Leppäaho
9   *
10  */
11 public class PosUtils {
12
13   /**
14    * Multiplies a bwapi position by an int.
15    *
16    * @param pos
17    *      The position to multiply.
18    * @param x
19    *      The multiplier.
20    * @return The product.
21    */
22   public static Position multiply(Position pos, int x) {
23     return new Position(pos.getX() * x, pos.getY() * x);
24   }
25
26   /**
27    * Adds two bwapi positions.
28    *
29    * @param x
```

```
30      *      The position to add.
31      * @param y
32      *      Another position to add.
33      * @return The sum.
34      */
35     public static Position add(Position x, Position y) {
36       return new Position(x.getX() + y.getX(), x.getY() + y.getY());
37
38     }
39 }
```

Listaus 7.4: bot/PotentialCalculator.java

```
1  package thesis.bot;
2
3  import java.rmi.RemoteException;
4  import java.util.ArrayList;
5  import java.util.List;
6
7  import thesis.bot.PosUtils;
8  import thesis.rmi.PotentialFunctionProvider;
9  import bwapi.*;
10 import bwta.BWTA;
11
12 /**
13  * Calculates the potential field that guides the movement of the bot. The
14  * default potential field is designed to keep the units at maximum shooting
15  * distance from a single enemy unit.
16  *
17  * The potential field can also be provided by an external component.
18  *
19  * @author Oskari Leppäaho
20  *
21  */
22 public class PotentialCalculator {
23
24   Controller bot;
25   PotentialFunctionProvider potentialProvider; // the component that will
```

```
26                              // provide
27                              // the potential function
28
29      /**
30       * Creates the PotentialCalculator.
31       *
32       * @param bot
33       *      A reference to the bot controller.
34       */
35      public PotentialCalculator(Controller bot) {
36        this.bot = bot;
37      }
38
39      /**
40       * Sets the potential provider.
41       *
42       * @param potentialProvider
43       *      The new potential provider.
44       */
45      public void setPotentialProvider(PotentialFunctionProvider potentialProvider)
            {
46        this.potentialProvider = potentialProvider;
47      }
48
49      /**
50       * Get the potential for a position. Uses
51       * {@link #getPotential(double, double)} to calculate the potential.
52       *
53       * @param pos
54       *      The position
55       * @param u
56       *      The unit for which to calculate the potential.
57       * @return
58       */
59      double getPotential(Position pos, Unit u) {
60        return getPotential(pos.getX(), pos.getY(), u);
61      }
```

```
62
63    /**
64     * Gets the potential for coordinates. Uses the potentialProvider if it was
65     * set. Otherwise uses the potential function defined here (changes
66     * frequently during development).
67     *
68     * @param x
69     *      X coordinate
70     * @param y
71     *      Y coordinate
72     * @return Total potential for the location.
73     */
74    private double getPotential(double x, double y, Unit u) {
75      double ownMaximumShootDistance = u.getType().groundWeapon().maxRange();
76      double potential = 0;
77      double relativeHP = bot.getRelativeHp(u);
78
79      Position nearestPoint = BWTA.getRegion(u.getPosition()).getPolygon()
80          .getNearestPoint(u.getPosition());
81      nearestPoint.getDistance(u.getPosition());
82      double distMapEdge = nearestPoint.getDistance(u.getPosition());
83
84      boolean onCooldown = !(u.getGroundWeaponCooldown() == 0);
85      double[] distancesFromEdges = { distMapEdge };
86      double[] distancesFromEnemies = getEnemyDistances(x, y);
87      double[][] enemyPositionVectors = getEnemyPositionVectors(x, y);
88      double[] distancesFromOwnUnits = getOwnUnitDistances(x, y, u);
89
90      if (potentialProvider == null) {
91
92        // fast game
93        // potential = 0;
94
95        // original
96        // potential += -(0.05 * enemyDistance - 5)
97        // * (0.05 * enemyDistance - 5);
98        double maxPotential = -Double.MAX_VALUE;
```

```
99        double currentPotential;
100       if (!onCooldown)
101         for (int j = 0; j < distancesFromEnemies.length; j++) {
102           currentPotential = enemyPotential(distancesFromEnemies[j],
103               ownMaximumShootDistance, relativeHP, enemyPositionVectors[j][0],
                      enemyPositionVectors[j][1]);
104           if (maxPotential < currentPotential)
105             maxPotential = currentPotential;
106         }
107       else
108         for (int j = 0; j < distancesFromEnemies.length; j++) {
109           currentPotential = enemyPotentialWhenOnCooldown(
110               distancesFromEnemies[j], ownMaximumShootDistance,
111               relativeHP, enemyPositionVectors[j][0], enemyPositionVectors[j
                      ][1]);
112           if (maxPotential < currentPotential)
113             maxPotential = currentPotential;
114         }
115       potential += maxPotential;
116       for (int j = 0; j < distancesFromOwnUnits.length; j++) {
117         potential += ownPotential(distancesFromOwnUnits[j]);
118       }
119       potential += mapEdgePotential(distMapEdge);
120     } else {
121       try {
122         potential += potentialProvider.getPotential(
123             distancesFromEnemies, distancesFromOwnUnits,
124             ownMaximumShootDistance, distancesFromEdges,
125             onCooldown, relativeHP, enemyPositionVectors);
126       } catch (RemoteException e) {
127         System.err.println("Remote potential evaluation failed: ");
128         e.printStackTrace();
129       }
130     }
131
132     return potential;
133   }
```

```
134
135     /**
136      * Gets the distances of the own units from given position. Ignores unit u.
137      *
138      *
139      * @param x
140      *      X coordinate
141      * @param y
142      *      Y coordinate
143      * @param u
144      *      This unit will be ignored
145      * @return Array of own unit distances from given location (without unit u).
146      */
147     private double[] getOwnUnitDistances(double x, double y, Unit ignoreUnit) {
148       List<Unit> myUnits = new ArrayList<Unit>(bot.getMyUnitsNoRevealers());
149       myUnits.remove(ignoreUnit);
150       return getUnitDistances(x, y, myUnits);
151     }
152
153     /**
154      * Gets the distances of the enemy units from given position.
155      *
156      * @param x
157      *      X coordinate
158      * @param y
159      *      Y coordinate
160      * @return Array of enemy unit distances from that location.
161      */
162     private double[] getEnemyDistances(double x, double y) {
163       return getUnitDistances(x, y, bot.getEnemyUnitsNoRevealers());
164     }
165
166     /**
167      * Gets the distances of the enemy units from given position.
168      *
169      * @param x
170      *      X coordinate
```

```
171      * @param y
172      *      Y coordinate
173      * @return Array of enemy unit distances from that location.
174      */
175     private double[][] getEnemyPositionVectors(double x, double y) {
176       List<Unit> enemyUnits = bot.getEnemyUnitsNoRevealers();
177       double[][] positionVectors = new double[enemyUnits.size()][2];
178       for(int i = 0; i < enemyUnits.size(); i++) {
179         Position enemyPosition = enemyUnits.get(i).getPosition();
180         positionVectors[i][0] = enemyPosition.getX() - x;
181         positionVectors[i][1] = enemyPosition.getY() - y;
182       }
183       return positionVectors;
184     }
185
186     /**
187      * Gets the distances of the units from given position.
188      *
189      * @param x
190      *      X coordinate
191      * @param y
192      *      Y coordinate
193      * @param units
194      *      List of units whose distances to get.
195      * @return Array of enemy unit distances from that location.
196      */
197     private double[] getUnitDistances(double x, double y, List<Unit> units) {
198       double[] distances = new double[units.size()];
199       int i = 0;
200       for (Unit u : units) {
201         Position pos = u.getPosition();
202         double xlen = pos.getX() - x;
203         double ylen = pos.getY() - y;
204         double distance = Math.sqrt(xlen * xlen + ylen * ylen);
205         distances[i++] = distance;
206       }
207       return distances;
```

```
208     }
209
210     /**
211      * (Tree 0)
212      * Calculates the potential depending on proximity to an enemy unit.
213      *
214      * @param x Unit's distance from an enemy unit.
215      * @param y Unit's own maximum shooting distance
216      * @param z Unit's remaining hit points relative to the average of all own
             units
217      * @param u X coordinate of vector (enemyPosition - ownPosition)
218      * @param w Y coordinate of vector (enemyPosition - ownPosition)
219      * @return The potential caused by the enemy unit.
220      */
221     public double enemyPotential(double x, double y, double z, double u, double w)
             {
222       return exp(tanh(tanh(tanh(x + -453.0156915247012) / cube(x -
             -175.24203878943587))) - (sin(tan(exp(x - x))) - tanh(tanh(negexp(sin(
             tanh(tanh(tanh(x + -453.0156915247012) / cube(x - -175.24203878943587)))
             ))))));
223     }
224
225     /**
226      * (Tree 1)
227      * Calculates the potential depending on proximity to a map edge.
228      *
229      * @param x
230      *      Unit's distance from map edge
231      * @return The potential caused by the map edge.
232      */
233     public double mapEdgePotential(double x) {
234       return square(0.11725951837008397);
235     }
236
237     /**
238      * (Tree 2)
239      * Calculates the potential depending on proximity to an own unit.
```

```
240      *
241      * @param x
242      *      Unit's distance from an own unit.
243      * @return The potential caused by the own unit.
244      */
245     public double ownPotential(double x) {
246       return 0.1416434362861081;
247     }
248
249     /**
250      * (Tree 3)
251      * Calculates the potential depending on proximity to an enemy unit.
252      *
253      * @param x Unit's distance from an enemy unit.
254      * @param y Unit's own maximum shooting distance
255      * @param z Unit's remaining hit points relative to the average of all own
             units
256      * @param u X coordinate of vector (enemyPosition - ownPosition)
257      * @param w Y coordinate of vector (enemyPosition - ownPosition)
258      * @return The potential caused by the enemy unit.
259      */
260     public double enemyPotentialWhenOnCooldown(double x, double y,
261         double z, double u, double w) {
262       return negexp(iflte(cos(sqrt(cube(cube(cos(0.1403215755394167))))), exp(z),
            cube(y), (((tanh(exp(w)) * (log(x) - (square((tanh(exp(w)) * (log(x) -
            (404.49523422431037 + -13.15162410727089))) / sqrt(cube(cos
            (0.1403215755394167)))) * square(tan(-320.6165247044389))))) / sqrt(cube
            (cos(0.1403215755394167)))) * (log(x) - (404.49523422431037 +
            -13.15162410727089))) / sqrt(cube(cos(0.1403215755394167)))));
263     }
264
265     /**
266      * Gets the potentials around an unit in 8 directions and the unit's current
267      * position. The potentials are measured in 1 pixel's distance from the
268      * unit's current position (diagonals included).
269      *
270      * @param u
```

```
271     *      The unit around which to measure the potentials.
272     * @return An array with the potentials around the unit. The order is from
273     *     top left to bottom right, row by row.
274     */
275   double[] getPotentialsAround(Unit u) {
276     double[] potentials = new double[9];
277     int k = 0;
278     for (int n = -1; n <= 1; n++) {
279       for (int m = -1; m <= 1; m++) {
280         double i = m;
281
282         double j = n;
283
284         /*
285          * It's OK that i and j will be < 1, because getPotential can
286          * handle doubles. It's important that the diagonal potentials
287          * are checked in positions whose distance from the unit is the
288          * same as with the other directions. Otherwise the diagonal
289          * directions might get favored more than other directions.
290          *
291          * But remember not to divide by zero ;)
292          */
293         if (i != 0)
294           i /= Math.sqrt(i * i + j * j);
295         if (j != 0)
296           j /= Math.sqrt(i * i + j * j);
297
298         double posX = u.getPosition().getX() + i;
299         double posY = u.getPosition().getY() + j;
300         potentials[k] = getPotential(posX, posY, u);
301         k++;
302       }
303     }
304     return potentials;
305   }
306
307   /**
```

```
308     * Gets the move direction based on an array of potentials. The move
309     * direction will be the direction with the highest potential. Default value
310     * is (0,0: no movement).
311     *
312     * @param potentials
313     *      An array of potentials around an unit. The order is from top
314     *      left to bottom right, row by row.
315     * @return The direction corresponding to the highest potential.
316     */
317    Position getMoveDirection(double[] potentials) {
318      int highestIndex = Controller.findHighestDefaultTo4(potentials);
319      // Indices:
320      // 0 1 2 -1,-1 0,-1 1,-1
321      // 3 4 5 -1, 0 0, 0 1 ,0
322      // 6 7 8 -1, 1 0, 1 1, 1
323      int x = highestIndex % 3 - 1;
324      int y = ((highestIndex) / 3 - 1);
325      return new Position(x, y);
326    }
327
328    /**
329     * Checks for the highest potential in each pixel coordinate from a position
330     * in a direction for a distance. If the potential is the same in all
331     * distances (or highest potentials are at the from position and some
332     * direction) returns the from position.
333     *
334     * @param from
335     *      The position from which to start checking.
336     * @param dir
337     *      The direction in which to check. Should be a position with
338     *      coordinates between -1 and 1.
339     * @param distance
340     *      Distance for which to check.
341     * @param u
342     *      The unit for which to check.
343     * @return The position with the highest potential. Default is the current
344     *     position.
```

```
345     */
346    Position getHighestPotentialPosition(Position from, Position dir,
347        int distance, Unit u) throws IllegalArgumentException {
348      if (dir.getX() < -1 || 1 < dir.getX() || dir.getY() < -1
349          || 1 < dir.getY())
350        throw new IllegalArgumentException("Direction should "
351            + "have coordinates between -1 and 1. Was " + dir.getX()
352            + " and " + dir.getY());
353
354      double highestPotential = getPotential(from, u);
355      Position highestPosition = from;
356      if ((dir.getX() * dir.getX() + dir.getY() * dir.getY()) == 2) {
357        distance = (int) (distance / Math.sqrt(2.0));
358      }
359
360      // -------------debug
361      // Position enemyPos = new Position(0,0);
362      // for (Unit u : bot.getEnemyUnitsNoRevealers()) {
363      // enemyPos = u.getPosition();
364      // }
365      //
366      // -------------enddebug
367
368      for (int i = 0; i <= distance; i++) {
369
370        Position offset = PosUtils.multiply(dir, i);
371
372        double currentPotential = getPotential(PosUtils.add(from, offset),
373            u);
374
375        // -------------debug
376        //
377        // System.out.println("Distance: " +
378        // enemyPos.getPDistance(from.plus(offset)) + " potential: " +
379        // currentPotential);
380        // -------------enddebug
381
```

```
382         if (currentPotential > highestPotential) {
383           highestPosition = PosUtils.add(from, offset);
384           highestPotential = currentPotential;
385         }
386       }
387       return highestPosition;
388     }
389
390     double sin(double x) {
391       return Math.sin(x);
392     }
393
394     double cos(double x) {
395       return Math.cos(x);
396     }
397
398     double cube(double x) {
399       return x*x*x;
400     }
401
402     double exp(double x) {
403       return Math.exp(x);
404     }
405
406     double iflte(double condition1, double condition2, double returnValue1, double
          returnValue2) {
407       if(condition1 < condition2)
408         return returnValue1;
409       else
410         return returnValue2;
411     }
412
413     double log(double x) {
414       if (x == 0)
415         return 0;
416       else
417         return Math.log(Math.abs(x));
```

```
418     }
419
420     double negexp(double x) {
421       return Math.exp(0 - x);
422     }
423
424     double pow(double x, double power) {
425       double result = Math.pow(Math.abs(x), power);
426       if (result == Double.NaN || result == Double.POSITIVE_INFINITY || result ==
              Double.NEGATIVE_INFINITY )
427         result = 0;
428       return result;
429     }
430
431     double sqrt(double x) {
432       return Math.sqrt(Math.abs(x));
433     }
434
435     double square(double x) {
436       return x*x;
437     }
438
439     double tan(double x) {
440       double result = Math.tan(x);
441       if (result == Double.NaN || result == Double.POSITIVE_INFINITY || result ==
              Double.NEGATIVE_INFINITY )
442         result = 0;
443       return result;
444     }
445
446     double tanh(double x) {
447       return Math.tanh(x);
448     }
449 }
```

Listaus 7.5: bot/Visualizer.java

```
1 package thesis.bot;
```

```
2
3  import java.util.HashMap;
4  import java.util.List;
5
6  import bwapi.*;
7  import bwta.Region;
8  import bwta.*;
9
10 /**
11  * Handles the visualizations on screen during the gameplay for the AI client.
12  *
13  * @author Oskari Leppäaho
14  *
15  */
16 public class Visualizer {
17
18   final Game game;
19   final Controller bot;
20   final boolean isVisualizationOn;
21
22   HashMap<Unit, Unit> attackTargets = new HashMap<Unit,Unit>(); // Key is the
         attacker, value is the
23                       // target.
24
25   public Visualizer(Game game, Controller bot, boolean isVisualizationOn) {
26     this.game = game;
27     this.bot = bot;
28     this.isVisualizationOn = isVisualizationOn;
29   }
30
31   /**
32    * Draw a circle around both own and enemy units to highlight them.
33    */
34   void highlightUnits() {
35     if (isVisualizationOn) {
36       int enemyCircleRadius = 128; // Maximum shooting distance for a
37                         // dragoon.
```

```
38        int ownCircleRadius = 2;
39        // Position enemyPosition = null;
40        // Position myPosition = null;
41        for (Unit u : bot.getEnemyUnitsNoRevealers()) {
42          game.drawCircleMap(u.getPosition().getX(), u.getPosition()
43              .getY(), enemyCircleRadius, Color.Orange, false);
44          // bwapi.drawCircle(u.getPosition(), 120, BWColor.Orange, false,
45          // false);
46          // bwapi.drawCircle(u.getPosition(), 136, BWColor.Orange, false,
47          // false);
48          // System.out.println("Enemy position: " + u.getPosition());
49          // enemyPosition = u.getPosition();
50        }
51        for (Unit u : bot.getMyUnitsNoRevealers()) {
52          game.drawCircleMap(u.getPosition().getX(), u.getPosition()
53              .getY(), ownCircleRadius, Color.Blue, false);
54          // bwapi.drawCircle(u.getPosition(), 136, BWColor.Orange, false,
55          // false);
56          // System.out.println("Enemy position: " + u.getPosition());
57          // myPosition = u.getPosition();
58        }
59
60        // System.out.println("Distance between units: " +
61        // enemyPosition.getPDistance(myPosition));
62      }
63    }
64
65    /**
66     * Visualize the destination/target of the unit. If the unit is attacking,
67     * draw a red line to its target. If it is moving, draw a green line to its
68     * destination.
69     *
70     * @param u
71     *      The unit to visualize.
72     * @param moveTo
73     */
74    void visualizeDestination(Unit u, Position moveTo, boolean isAttacking) {
```

```
75      if (isVisualizationOn) {
76        if (isAttacking) {
77          Unit attackTarget = attackTargets.get(u);
78          game.drawLineMap(u.getPosition().getX(),
79              u.getPosition().getY(), attackTarget.getPosition()
80                  .getX(), attackTarget.getPosition().getY(),
81              Color.Red);
82        } else if (moveTo != null) {
83          game.drawLineMap(u.getPosition().getX(),
84              u.getPosition().getY(), moveTo.getX(), moveTo.getY(),
85              Color.Green);
86        }
87      }
88    }
89
90    /**
91     * Draws the potential values around an unit on the screen for
92     * visualization/debugging purposes.
93     *
94     * @param potentials
95     *      Array of the potential values.
96     */
97    void drawPotentialValues(double[] potentials) {
98      if (isVisualizationOn) {
99        String text;
100       for (int i = 0; i < potentials.length; i++) {
101         text = String.format("%.5f", potentials[i]);
102         game.drawTextScreen(100 * (i % 3), 40 + 10 * ((i) / 3), text);
103       }
104       int highestIndex = Controller.findHighestDefaultTo4(potentials);
105       text = String.format("%c%.5f", 0x06, potentials[highestIndex]);
106       game.drawTextScreen(100 * (highestIndex % 3),
107           40 + 10 * ((highestIndex) / 3), text);
108     }
109   }
110
111   /**
```

```
112     * Sets the attack target for a unit.
113     *
114     * @param attacker
115     *      The attacker.
116     * @param target
117     *      The target.
118     */
119    public void setAttackTarget(Unit attacker, Unit target) {
120      attackTargets.put(attacker, target);
121    }
122  }
```

Listaus 7.6: evolution/DoubleData.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution;
8
9  import ec.gp.*;
10
11 /**
12  * GPData representing a double precision floating point number.
13  *
14  * @author Sean Luke
15  *
16  */
17 public class DoubleData extends GPData {
18    public double x; // return value
19
20    public void copyTo(final GPData gpd) // copy my stuff to another DoubleData
21    {
22      ((DoubleData) gpd).x = x;
23    }
24 }
```

Listaus 7.7: evolution/PrintEntirePopulationStatistics.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution;
9  import ec.*;
10 import ec.simple.SimpleProblemForm;
11 import ec.steadystate.*;
12
13 import java.io.IOException;
14
15 import ec.util.*;
16
17 import java.io.File;
18
19 /*
20  * SimpleStatistics.java
21  *
22  * Created: Tue Aug 10 21:10:48 1999
23  * By: Sean Luke
24  */
25
26 /**
27  * A basic Statistics class suitable for simple problem applications.
28  *
29  * SimpleStatistics prints out the best individual, per subpopulation,
30  * each generation. At the end of a run, it also prints out the best
31  * individual of the run. SimpleStatistics outputs this data to a log
32  * which may either be a provided file or stdout. Compressed files will
33  * be overridden on restart from checkpoint; uncompressed files will be
34  * appended on restart.
35  *
36  * <p>SimpleStatistics implements a simple version of steady-state statistics:
```

```
37  * if it quits before a generation boundary,
38  * it will include the best individual discovered, even if the individual was
        discovered
39  * after the last boundary. This is done by using individualsEvaluatedStatistics
        (...)
40  * to update best-individual-of-generation in addition to doing it in
41  * postEvaluationStatistics(...).
42  *
43  * <p><b>Parameters</b><br>
44  * <table>
45  * <tr><td valign=top><i>base.</i><tt>gzip</tt><br>
46  * <font size=-1>boolean</font></td>
47  * <td valign=top>(whether or not to compress the file (.gz suffix added)</td></tr
        >
48  * <tr><td valign=top><i>base.</i><tt>file</tt><br>
49  * <font size=-1>String (a filename), or nonexistant (signifies stdout)</font></td
        >
50  * <td valign=top>(the log for statistics)</td></tr>
51  * </table>
52  *
53  *
54  * @author Sean Luke
55  * @version 1.0
56  */
57
58 public class PrintEntirePopulationStatistics extends Statistics implements
        SteadyStateStatisticsForm //, ec.eval.ProvidesBestSoFar
59  {
60  public Individual[] getBestSoFar() { return best_of_run; }
61
62  /** log file parameter */
63  public static final String P_STATISTICS_FILE = "file";
64
65  /** compress? */
66  public static final String P_COMPRESS = "gzip";
67
68  public static final String P_DO_FINAL = "do-final";
```

```
69    public static final String P_DO_GENERATION = "do-generation";
70    public static final String P_DO_MESSAGE = "do-message";
71    public static final String P_DO_DESCRIPTION = "do-description";
72    public static final String P_DO_PER_GENERATION_DESCRIPTION = "do-per-
        generation-description";
73
74    /** The Statistics' log */
75    public int statisticslog = 0; // stdout
76
77    /** The best individual we've found so far */
78    public Individual[] best_of_run = null;
79
80    /** Should we compress the file? */
81    public boolean compress;
82    public boolean doFinal;
83    public boolean doGeneration;
84    public boolean doMessage;
85    public boolean doDescription;
86    public boolean doPerGenerationDescription;
87
88    public void setup(final EvolutionState state, final Parameter base)
89      {
90      super.setup(state,base);
91
92      compress = state.parameters.getBoolean(base.push(P_COMPRESS),null,false);
93
94      File statisticsFile = state.parameters.getFile(
95        base.push(P_STATISTICS_FILE),null);
96
97      doFinal = state.parameters.getBoolean(base.push(P_DO_FINAL),null,true);
98      doGeneration = state.parameters.getBoolean(base.push(P_DO_GENERATION),null,
       true);
99      doMessage = state.parameters.getBoolean(base.push(P_DO_MESSAGE),null,true);
100     doDescription = state.parameters.getBoolean(base.push(P_DO_DESCRIPTION),null
       ,true);
101     doPerGenerationDescription = state.parameters.getBoolean(base.push(
       P_DO_PER_GENERATION_DESCRIPTION),null,false);
```

```
102
103     if (silentFile)
104       {
105       statisticslog = Output.NO_LOGS;
106       }
107     else if (statisticsFile!=null)
108       {
109       try
110         {
111         statisticslog = state.output.addLog(statisticsFile, !compress, compress)
    ;
112         }
113       catch (IOException i)
114         {
115         state.output.fatal("An IOException occurred while trying to create the
    log " + statisticsFile + ":\n" + i);
116         }
117       }
118     else state.output.warning("No statistics file specified, printing to stdout
    at end.", base.push(P_STATISTICS_FILE));
119     }
120
121   public void postInitializationStatistics(final EvolutionState state)
122     {
123     super.postInitializationStatistics(state);
124
125     // set up our best_of_run array -- can't do this in setup, because
126     // we don't know if the number of subpopulations has been determined yet
127     best_of_run = new Individual[state.population.subpops.length];
128     }
129
130   /** Logs the best individual of the generation. */
131   public void postEvaluationStatistics(final EvolutionState state)
132     {
133     super.postEvaluationStatistics(state);
134
135     // print the best-of-generation individual
```

```
136     if (doGeneration) state.output.println("\nGeneration: " + state.generation,
        statisticslog);
137
138     // for now we just print the best fitness per subpopulation.
139     Individual[] best_i = new Individual[state.population.subpops.length]; //
        quiets compiler complaints
140     for(int x=0;x<state.population.subpops.length;x++)
141       {
142       best_i[x] = state.population.subpops[x].individuals[0];
143       for(int y=1;y<state.population.subpops[x].individuals.length;y++)
144         if (state.population.subpops[x].individuals[y].fitness.betterThan(best_i
        [x].fitness))
145           best_i[x] = state.population.subpops[x].individuals[y];
146
147       // now test to see if it's the new best_of_run
148       if (best_of_run[x]==null || best_i[x].fitness.betterThan(best_of_run[x].
        fitness))
149         best_of_run[x] = (Individual)(best_i[x].clone());
150       }
151
152     for(int x=0;x<state.population.subpops.length;x++)
153       {
154       if (doGeneration) state.output.println("Subpopulation " + x + ":",
        statisticslog);
155       for(int y=0;y<state.population.subpops[x].individuals.length;y++)
156       {
157         state.output.println("", statisticslog);
158         state.output.println("Generation" + state.generation + ", Individual " +
         y, statisticslog);
159         state.population.subpops[x].individuals[y].printIndividualForHumans(
        state,statisticslog);
160       }
161       if (doMessage && !silentPrint) state.output.message("Subpop " + x + " best
         fitness of generation" +
162         (best_i[x].evaluated ? " " : " (evaluated flag not set): ") +
163         best_i[x].fitness.fitnessToStringForHumans());
164
```

```
165         // describe the winner if there is a description
166         if (doGeneration && doPerGenerationDescription)
167           {
168           if (state.evaluator.p_problem instanceof SimpleProblemForm)
169             ((SimpleProblemForm)(state.evaluator.p_problem.clone())).describe(
    state, best_i[x], x, 0, statisticslog);
170           }
171         }
172       }
173
174     /** Allows MultiObjectiveStatistics etc. to call super.super.finalStatistics
      (...) without
175       calling super.finalStatistics(...) */
176     protected void bypassFinalStatistics(EvolutionState state, int result)
177       { super.finalStatistics(state, result); }
178
179     /** Logs the best individual of the run. */
180     public void finalStatistics(final EvolutionState state, final int result)
181       {
182       super.finalStatistics(state,result);
183
184       // for now we just print the best fitness
185
186       if (doFinal) state.output.println("\nBest Individual of Run:",statisticslog)
    ;
187       for(int x=0;x<state.population.subpops.length;x++ )
188         {
189         if (doFinal) state.output.println("Subpopulation " + x + ":",statisticslog
    );
190         if (doFinal) best_of_run[x].printIndividualForHumans(state,statisticslog);
191         if (doMessage && !silentPrint) state.output.message("Subpop " + x + " best
     fitness of run: " + best_of_run[x].fitness.fitnessToStringForHumans());
192
193         // finally describe the winner if there is a description
194         if (doFinal && doDescription)
195           if (state.evaluator.p_problem instanceof SimpleProblemForm)
```

```
196             ((SimpleProblemForm)(state.evaluator.p_problem.clone())).describe(
    state, best_of_run[x], x, 0, statisticslog);
197         }
198       }
199     }
```

Listaus 7.8: evolution/thesis.params

```
1  # Copyright 2006 by Sean Luke and George Mason University
2  # Licensed under the Academic Free License version 3.0
3  # See the file "LICENSE" for more information
4
5  parent.0 = E:/Tiedostot/Workspace/OneDrive/Gradu/3rd party libraries/ecj/params/
       ec/gp/koza/koza.params
6
7  # the next four items are already defined in koza.params, but we
8  # put them here to be clear.
9
10 gp.tc.size = 4
11 gp.tc.0 = ec.gp.GPTreeConstraints
12 gp.tc.1 = ec.gp.GPTreeConstraints
13 gp.tc.2 = ec.gp.GPTreeConstraints
14 gp.tc.3 = ec.gp.GPTreeConstraints
15 gp.tc.0.returns = nil
16 gp.tc.0.init = ec.gp.koza.HalfBuilder
17 gp.tc.1.returns = nil
18 gp.tc.1.init = ec.gp.koza.HalfBuilder
19 gp.tc.2.returns = nil
20 gp.tc.2.init = ec.gp.koza.HalfBuilder
21 gp.tc.3.returns = nil
22 gp.tc.3.init = ec.gp.koza.HalfBuilder
23
24 gp.tc.0.name = tc0
25 gp.tc.0.fset = f1
26 gp.tc.1.name = tc1
27 gp.tc.1.fset = f0
28 gp.tc.2.name = tc2
29 gp.tc.2.fset = f0
```

```
gp.tc.3.name = tc3
gp.tc.3.fset = f1

# We have one function set, of class GPFunctionSet
gp.fs.size = 2

gp.fs.0 = ec.gp.GPFunctionSet
# We'll call the function set "f0".
gp.fs.0.name = f0

# We have six functions in the function set. They are:
gp.fs.0.size = 18
gp.fs.0.func.0 = thesis.evolution.nodes.X
gp.fs.0.func.0.nc = nc0
gp.fs.0.func.1 = thesis.evolution.nodes.Add
gp.fs.0.func.1.nc = nc2
gp.fs.0.func.2 = thesis.evolution.nodes.Sub
gp.fs.0.func.2.nc = nc2
gp.fs.0.func.3 = thesis.evolution.nodes.Mul
gp.fs.0.func.3.nc = nc2
gp.fs.0.func.4 = thesis.evolution.nodes.Div
gp.fs.0.func.4.nc = nc2
gp.fs.0.func.5 = thesis.evolution.nodes.FloatERC
gp.fs.0.func.5.nc = nc0
gp.fs.0.func.6 = thesis.evolution.nodes.BigFloatERC
gp.fs.0.func.6.nc = nc0
gp.fs.0.func.7 = thesis.evolution.nodes.Cos
gp.fs.0.func.7.nc = nc1
gp.fs.0.func.8 = thesis.evolution.nodes.Cube
gp.fs.0.func.8.nc = nc1
gp.fs.0.func.9 = thesis.evolution.nodes.Exp
gp.fs.0.func.9.nc = nc1
gp.fs.0.func.10 = thesis.evolution.nodes.Iflte
gp.fs.0.func.10.nc = nc4
gp.fs.0.func.11 = thesis.evolution.nodes.Log
gp.fs.0.func.11.nc = nc1
gp.fs.0.func.12 = thesis.evolution.nodes.NegExp
```

```
gp.fs.0.func.12.nc = nc1
gp.fs.0.func.13 = thesis.evolution.nodes.Sin
gp.fs.0.func.13.nc = nc1
gp.fs.0.func.14 = thesis.evolution.nodes.Sqrt
gp.fs.0.func.14.nc = nc1
gp.fs.0.func.15 = thesis.evolution.nodes.Square
gp.fs.0.func.15.nc = nc1
gp.fs.0.func.16 = thesis.evolution.nodes.Tan
gp.fs.0.func.16.nc = nc1
gp.fs.0.func.17 = thesis.evolution.nodes.Tanh
gp.fs.0.func.17.nc = nc1

gp.fs.1 = ec.gp.GPFunctionSet
gp.fs.1.name = f1

# We have six functions in the function set. They are:
gp.fs.1.size = 21
gp.fs.1.func.0 = thesis.evolution.nodes.X
gp.fs.1.func.0.nc = nc0
gp.fs.1.func.1 = thesis.evolution.nodes.Add
gp.fs.1.func.1.nc = nc2
gp.fs.1.func.2 = thesis.evolution.nodes.Sub
gp.fs.1.func.2.nc = nc2
gp.fs.1.func.3 = thesis.evolution.nodes.Mul
gp.fs.1.func.3.nc = nc2
gp.fs.1.func.4 = thesis.evolution.nodes.Div
gp.fs.1.func.4.nc = nc2
gp.fs.1.func.5 = thesis.evolution.nodes.FloatERC
gp.fs.1.func.5.nc = nc0
gp.fs.1.func.6 = thesis.evolution.nodes.Y
gp.fs.1.func.6.nc = nc0
gp.fs.1.func.7 = thesis.evolution.nodes.Z
gp.fs.1.func.7.nc = nc0
gp.fs.1.func.8 = thesis.evolution.nodes.BigFloatERC
gp.fs.1.func.8.nc = nc0
gp.fs.1.func.9 = thesis.evolution.nodes.Cos
gp.fs.1.func.9.nc = nc1
```

```
104 gp.fs.1.func.10 = thesis.evolution.nodes.Cube
105 gp.fs.1.func.10.nc = nc1
106 gp.fs.1.func.11 = thesis.evolution.nodes.Exp
107 gp.fs.1.func.11.nc = nc1
108 gp.fs.1.func.12 = thesis.evolution.nodes.Iflte
109 gp.fs.1.func.12.nc = nc4
110 gp.fs.1.func.13 = thesis.evolution.nodes.Log
111 gp.fs.1.func.13.nc = nc1
112 gp.fs.1.func.14 = thesis.evolution.nodes.NegExp
113 gp.fs.1.func.14.nc = nc1
114 gp.fs.1.func.15 = thesis.evolution.nodes.Sin
115 gp.fs.1.func.15.nc = nc1
116 gp.fs.1.func.16 = thesis.evolution.nodes.Sqrt
117 gp.fs.1.func.16.nc = nc1
118 gp.fs.1.func.17 = thesis.evolution.nodes.Square
119 gp.fs.1.func.17.nc = nc1
120 gp.fs.1.func.18 = thesis.evolution.nodes.Tan
121 gp.fs.1.func.18.nc = nc1
122 gp.fs.1.func.19 = thesis.evolution.nodes.Tanh
123 gp.fs.1.func.19.nc = nc1
124 gp.fs.1.func.20 = thesis.evolution.nodes.W
125 gp.fs.1.func.20.nc = nc0
126
127
128 eval.problem = thesis.evolution.ThesisProblem
129 eval.problem.data = thesis.evolution.DoubleData
130 # How many times evaluation of an individual should be repeated
131 # Using the average fitness.
132 eval.problem.repeated-evaluations = 10
133
134 # own additions
135 pop.subpop.0.size =     500
136 generations =       26
137 print-params = true
138
139 evalthreads = 8
140
```

```
141 seed.0 = time
142 seed.1 = time
143 seed.2 = time
144 seed.3 = time
145 seed.4 = time
146 seed.5 = time
147 seed.6 = time
148 seed.7 = time
149
150 pop.subpop.0.species.ind.numtrees = 4
151 pop.subpop.0.species.ind.tree.0 = ec.gp.GPTree
152 pop.subpop.0.species.ind.tree.0.tc = tc0
153 pop.subpop.0.species.ind.tree.1 = ec.gp.GPTree
154 pop.subpop.0.species.ind.tree.1.tc = tc1
155 pop.subpop.0.species.ind.tree.2 = ec.gp.GPTree
156 pop.subpop.0.species.ind.tree.2.tc = tc2
157 pop.subpop.0.species.ind.tree.3 = ec.gp.GPTree
158 pop.subpop.0.species.ind.tree.3.tc = tc3
159
160 # Print programs in results in c style rather than
161 # lisp style.
162 gp.tree.print-style=c
163
164 # More statistics than just the SimpleStatistics.
165 stat = ec.gp.koza.KozaShortStatistics
166 stat.do-size = true
167 stat.do-time = true
168
169 stat.num-children = 2
170 stat.child.0 = ec.simple.SimpleStatistics
171 stat.child.0.file = $out2.stat
172 stat.child.1 = thesis.evolution.PrintEntirePopulationStatistics
173 stat.child.1.file = $out3.stat
174
175 checkpoint = true
176 checkpoint-modulo = 1
177 checkpoint-directory = /../../../checkpoints/
```

```
178
179 gp.koza.half.min-depth = 2
180 gp.koza.half.max-depth = 8
```

Listaus 7.9: evolution/ThesisProblem.java

```
1  package thesis.evolution;
2
3  import java.io.BufferedReader;
4  import java.io.InputStreamReader;
5  import java.rmi.RemoteException;
6  import java.util.ArrayList;
7
8  import ec.util.*;
9  import ec.*;
10 import ec.gp.*;
11 import ec.gp.koza.*;
12 import ec.simple.*;
13 import thesis.rmi.PotentialFunctionProvider;
14 import thesis.rmi.ThesisProblemRMI;
15 import thesis.rmi.RemoteBotInterface;
16
17 /**
18  * The evolution problem of the master's thesis.
19  *
20  * The object is to evolve a potential function that results in the best
21  * possible performance in a small scale combat scenario in StarCraft Broodwar
22  * RTS game.
23  *
24  * Different trees in the individuals are ment to calculate different potential
25  * fields: Tree 0: Enemy units Tree 1: Map edges
26  *
27  * @author Oskari Leppäaho
28  *
29  */
30 public class ThesisProblem extends GPProblem implements SimpleProblemForm,
31     PotentialFunctionProvider {
32   private static final long serialVersionUID = 1;
```

```
33
34      /**
35       * Parameter name for how many times the evaluation of a single individual
36       * should be repeated
37       */
38      public static final String P_REPEATEDEVALUATIONS = "repeated-evaluations";
39
40      public double currentX;
41      public double currentY;
42      public double currentZ;
43      public double currentW;
44      public double currentU;
45      /** The broodwar clients used in evaluation */
46      ArrayList<RemoteBotInterface> bwClients;
47
48      // Information regarding the current individual
49      EvolutionState state;
50      int threadnum;
51      Individual ind;
52      DoubleData localInput;
53
54      int repeatedEvaluations; // How many times a single individual should be
55                               // evaluated?
56
57      int currentGen = 0;
58      int individualssEvaluatedThisGen = 0;
59
60      /**
61       * Setup the evolution. Read how many times an individual should be
62       * evaluated. Connect to the remote BroodWar clients.
63       */
64      public void setup(final EvolutionState state, final Parameter base) {
65        super.setup(state, base);
66
67        Parameter def = defaultBase();
68
69        repeatedEvaluations = state.parameters.getIntWithDefault(
```

```
70          base.push(P_REPEATEDEVALUATIONS),
71          def.push(P_REPEATEDEVALUATIONS), 1);
72
73      System.out.println("Repeated evaluations: " + repeatedEvaluations);
74
75      // verify our input is the right class (or subclasses from it)
76      if (!(input instanceof DoubleData))
77        state.output.fatal("GPData class must subclass from "
78            + DoubleData.class, base.push(P_DATA), null);
79      bwClients = new ArrayList<RemoteBotInterface>();
80      System.out.println(state.evalthreads + " evalthreads");
81
82      ThesisProblemRMI starter = new ThesisProblemRMI();
83
84      BufferedReader buffer = new BufferedReader(new InputStreamReader(
85          System.in));
86      for (int i = 0; i < state.evalthreads; i++) {
87        System.out.println("Start client " + i);
88        System.out
89            .println("Press enter when the client has been started >");
90        try {
91          buffer.readLine();
92          bwClients.add(starter.connectClient(i + ""));
93          System.out.println("Connected to remote bot " + i);
94        } catch (Exception e) {
95          System.err.println("Something went wrong when waiting for \n"
96              + "the user to start the client\n" + "(for stepping).");
97          e.printStackTrace();
98        }
99      }
100   }
101
102     /** Called to reinitialize remote evaluation network contacts when the run
        is restarted from checkpoint. By default does nothing. */
103   public void reinitializeContacts( EvolutionState state )
104   {
105     bwClients = new ArrayList<RemoteBotInterface>();
```

```
106     System.out.println(state.evalthreads + " evalthreads");
107
108     ThesisProblemRMI starter = new ThesisProblemRMI();
109
110     BufferedReader buffer = new BufferedReader(new InputStreamReader(
111         System.in));
112     for (int i = 0; i < state.evalthreads; i++) {
113       System.out.println("Start client " + i);
114       System.out
115           .println("Press enter when the client has been started >");
116       try {
117         buffer.readLine();
118         bwClients.add(starter.connectClient(i + ""));
119         System.out.println("Connected to remote bot " + i);
120       } catch (Exception e) {
121         System.err.println("Something went wrong when waiting for \n"
122             + "the user to start the client\n" + "(for stepping).");
123         e.printStackTrace();
124       }
125     }
126   }
127
128
129   /**
130    * Evaluate the fitness of an individual. The fitness is the average of
131    * round scores of BroodWar matches run using the current individual as the
132    * potential function.
133    */
134   public void evaluate(final EvolutionState state, final Individual ind,
135       final int subpopulation, final int threadnum) {
136     if (!ind.evaluated) // don't bother reevaluating
137     {
138       this.ind = ind;
139       this.state = state;
140       this.threadnum = threadnum;
141
142       localInput = (DoubleData) (this.input);
```

```
143
144         int hits = 0;
145
146         double fitness = 0;
147         double fitnessSum = 0;
148
149         try {
150           for (int i = 0; i < repeatedEvaluations; i++) {
151             fitnessSum += bwClients.get(threadnum).getRoundScore(this);
152           }
153           fitness = fitnessSum / repeatedEvaluations;
154         } catch (RemoteException e) {
155           System.err.println("A remote exception while evaluating: ");
156           e.printStackTrace();
157           throw new Error();
158         }
159
160         if (fitness < 0.001)
161           hits++;
162
163         // the fitness better be KozaFitness!
164         KozaFitness f = ((KozaFitness) ind.fitness);
165         f.setStandardizedFitness(state, fitness);
166         f.hits = hits;
167         ind.evaluated = true;
168
169         printProgressInfo(subpopulation, threadnum);
170       }
171     }
172
173     /**
174      * Print information about the evolution progress. Prints the current thread
175      * number and the progress of the current evolution.
176      *
177      * @param subpopulation
178      *      Index of the current subpopulation
179      * @param threadnum
```

```
180      *      Index of the current thread.
181      */
182     private void printProgressInfo(int subpopulation, int threadnum) {
183       if (state.generation != currentGen) {
184         currentGen = state.generation;
185         individualssEvaluatedThisGen = 0;
186       }
187       individualssEvaluatedThisGen++;
188       System.out.println("Evaluated in thread " + threadnum);
189       System.out.println("Generation progress: "
190           + ((float) individualssEvaluatedThisGen)
191           / state.population.subpops[subpopulation].individuals.length
192           * 100 + " %");
193     }
194
195     /**
196      * Returns the potential for an enemy unit. This part of the potential
197      * function is evolved in tree 0.
198      *
199      * @param distanceFromEnemy
200      *      Player unit's distance from the enemy unit
201      * @param ownMaximumShootDistance
202      *      The own unit's maximum shooting distance.
203      * @param relativeHP
204      * @param enemyPositionVector
205      * @return The potential for the enemy.
206      */
207     private double getEnemyPotential(double distanceFromEnemy,
208         double ownMaximumShootDistance, double relativeHP,
209         double[] enemyPositionVector) {
210       currentX = distanceFromEnemy;
211       currentY = ownMaximumShootDistance;
212       currentZ = relativeHP;
213       currentU = enemyPositionVector[0];
214       currentW = enemyPositionVector[1];
215       ((GPIndividual) ind).trees[0].child.eval(state, threadnum, localInput,
216           stack, ((GPIndividual) ind), this);
```

```
217       return localInput.x;
218     }
219
220     /**
221      * Returns the potential for an enemy unit when the own unit is on cooldown.
222      * This part of the potential function is evolved in tree 3.
223      *
224      * @param distanceFromEnemy
225      *      Player unit's distance from the enemy unit
226      * @param ownMaximumShootDistance
227      *      The own unit's maximum shooting distance.
228      * @param relativeHP
229      * @param enemyPositionVectors
230      * @return The potential for the enemy.
231      */
232     private double getenemyPotentialWhenOnCooldown(double distanceFromEnemy,
233         double ownMaximumShootDistance, double relativeHP,
234         double[] enemyPositionVector) {
235       currentX = distanceFromEnemy;
236       currentY = ownMaximumShootDistance;
237       currentZ = relativeHP;
238       currentU = enemyPositionVector[0];
239       currentW = enemyPositionVector[1];
240       ((GPIndividual) ind).trees[3].child.eval(state, threadnum, localInput,
241           stack, ((GPIndividual) ind), this);
242       return localInput.x;
243     }
244
245     /**
246      * Returns the potential for an own unit. This part of the potential
247      * function is evolved in tree 2.
248      *
249      * @param distance
250      *      Player unit's distance from an own unit
251      * @return The potential for the enemy.
252      */
253     private double getOwnUnitPotential(double distance) {
```

```
254       currentX = distance;
255       ((GPIndividual) ind).trees[2].child.eval(state, threadnum, localInput,
256           stack, ((GPIndividual) ind), this);
257       return localInput.x;
258     }
259
260     /**
261      * Returns the potential for the map edges. This part of the potential
262      * function is evolved in tree 1.
263      *
264      * @param distancesFromEdges
265      *      Distances from the 4 map edges.
266      * @return The potential for the map edges.
267      */
268     private double getMapEdgePotential(double distanceFromEdge) {
269       double potential = 0;
270       currentX = distanceFromEdge;
271       ((GPIndividual) ind).trees[1].child.eval(state, threadnum, localInput,
272           stack, ((GPIndividual) ind), this);
273       potential += localInput.x;
274
275       return potential;
276     }
277
278     /**
279      * {@inheritDoc}
280      *
281      * Using the function from the individual currently being evaluated.
282      *
283      */
284     public double getPotential(double[] distancesFromEnemies,
285         double[] distancesFromOwnUnits, double ownMaximumShootDistance,
286         double[] distancesFromEdges, boolean onCooldown, double relativeHP,
287         double[][] enemyPositionVectors) {
288       double potential = 0;
289       double maxPotential = -Double.MAX_VALUE;
290       double currentPotential;
```

```
291     if (!onCooldown)
292       for (int i = 0; i < distancesFromEnemies.length; i++) {
293         currentPotential = getEnemyPotential(distancesFromEnemies[i],
294             ownMaximumShootDistance, relativeHP,
295             enemyPositionVectors[i]);
296         if (maxPotential < currentPotential)
297           maxPotential = currentPotential;
298       }
299     else {
300       for (int i = 0; i < distancesFromEnemies.length; i++) {
301         currentPotential = getenemyPotentialWhenOnCooldown(
302             distancesFromEnemies[i], ownMaximumShootDistance,
303             relativeHP, enemyPositionVectors[i]);
304         if (maxPotential < currentPotential)
305           maxPotential = currentPotential;
306       }
307     }
308     potential += maxPotential;
309     for (int i = 0; i < distancesFromOwnUnits.length; i++) {
310       potential += getOwnUnitPotential(distancesFromOwnUnits[i]);
311     }
312     for (int i = 0; i < distancesFromEdges.length; i++) {
313       potential += getMapEdgePotential(distancesFromEdges[i]);
314     }
315     return potential;
316   }
317 }
```

Listaus 7.10: evolution/nodes/Add.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
```

```
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 /**
15  * A node representing addition in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class Add extends GPNode {
22
23   public String toString() {
24     return "+";
25   }
26
27   /*
28    * public void checkConstraints(final EvolutionState state, final int tree,
29    * final GPIndividual typicalIndividual, final Parameter individualBase) {
30    * super.checkConstraints(state,tree,typicalIndividual,individualBase); if
31    * (children.length!=2)
32    * state.output.error("Incorrect number of children for node " +
33    * toStringForError() + " at " + individualBase); }
34    */
35   public int expectedChildren() {
36     return 2;
37   }
38
39   public void eval(final EvolutionState state, final int thread,
40       final GPData input, final ADFStack stack,
41       final GPIndividual individual, final Problem problem) {
42     double result;
43     DoubleData rd = ((DoubleData) (input));
44
45     children[0].eval(state, thread, input, stack, individual, problem);
```

```
46      result = rd.x;
47
48      children[1].eval(state, thread, input, stack, individual, problem);
49      rd.x = result + rd.x;
50    }
51  }
```

Listaus 7.11: evolution/nodes/BigFloatERC.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import ec.*;
10 import ec.app.regression.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 import java.io.*;
15
16
17 /*
18  * RegERC.java
19  *
20  * Created: Wed Nov 3 18:26:37 1999
21  * By: Sean Luke
22  */
23
24 /**
25  * @author Sean Luke
26  * @version 1.0
27  */
28
29 public class BigFloatERC extends FloatERC
```

```
30    {
31     public String name() { return "KornsERC"; }
32
33      public void resetNode(final EvolutionState state, final int thread)
34         { value = state.random[thread].nextDouble() * 1000.0 - 500.0; }
35    }
```

Listaus 7.12: evolution/nodes/Cos.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 /**
15  * A node representing cosine function in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class Cos extends GPNode {
22
23   public String toString() {
24     return "cos";
25   }
26
27   public int expectedChildren() {
28     return 1;
29   }
```

```
30
31   public void eval(final EvolutionState state, final int thread,
32       final GPData input, final ADFStack stack,
33       final GPIndividual individual, final Problem problem) {
34     double result;
35     DoubleData rd = ((DoubleData) (input));
36
37     children[0].eval(state, thread, input, stack, individual, problem);
38     rd.x = Math.cos(rd.x);
39   }
40 }
```

Listaus 7.13: evolution/nodes/Cube.java

```
1  /*
2   Copyright 2012 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.app.regression.*;
12 import ec.gp.*;
13 import ec.util.*;
14
15 /*
16  * Cube.java
17  *
18  * Created: Mon Apr 23 17:15:35 EDT 2012
19  * By: Sean Luke
20
21  <p>This function appears in the Korns function set, and is just x * x * x
22  <p>M. F. Korns. Accuracy in Symbolic Regression. In <i>Proc. GPTP.</i> 2011.
23
24 */
```

```
25
26 /**
27  * @author Sean Luke
28  * @version 1.0
29  */
30
31 public class Cube extends GPNode
32   {
33   public String toString() { return "cube"; }
34
35   public int expectedChildren() { return 1; }
36
37   public void eval(final EvolutionState state,
38     final int thread,
39     final GPData input,
40     final ADFStack stack,
41     final GPIndividual individual,
42     final Problem problem)
43     {
44     DoubleData rd = ((DoubleData)(input));
45
46     children[0].eval(state,thread,input,stack,individual,problem);
47     rd.x = rd.x * rd.x * rd.x;
48     }
49   }
```

Listaus 7.14: evolution/nodes/Div.java

```
1 /*
2  Copyright 2006 by Sean Luke
3  Licensed under the Academic Free License version 3.0
4  See the file "LICENSE" for more information
5 */
6
7
8 package thesis.evolution.nodes;
9 import thesis.evolution.DoubleData;
10 import ec.*;
```

```
11 import ec.gp.*;
12
13 /**
14  * A node representing division in a program tree generated by genetic
15  * programming.
16  *
17  * @author Sean Luke
18  *
19  */
20 public class Div extends GPNode
21   {
22
23   public String toString() { return "/"; }
24
25 /*
26  public void checkConstraints(final EvolutionState state,
27  final int tree,
28  final GPIndividual typicalIndividual,
29  final Parameter individualBase)
30  {
31  super.checkConstraints(state,tree,typicalIndividual,individualBase);
32  if (children.length!=2)
33  state.output.error("Incorrect number of children for node " +
34  toStringForError() + " at " +
35  individualBase);
36  }
37 */
38   public int expectedChildren() { return 2; }
39
40   public void eval(final EvolutionState state,
41     final int thread,
42     final GPData input,
43     final ADFStack stack,
44     final GPIndividual individual,
45     final Problem problem)
46     {
47     double result;
```

```
48      DoubleData rd = ((DoubleData)(input));
49
50      children[0].eval(state,thread,input,stack,individual,problem);
51      result = rd.x;
52
53      children[1].eval(state,thread,input,stack,individual,problem);
54      if (-0.001 < rd.x && rd.x < 0.001)
55        rd.x = 1.0;
56      else
57        rd.x = result / rd.x;
58      }
59    }
```

Listaus 7.15: evolution/nodes/Exp.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.app.regression.*;
12 import ec.gp.*;
13 import ec.util.*;
14
15 /*
16  * Exp.java
17  *
18  * Created: Wed Nov 3 18:26:37 1999
19  * By: Sean Luke
20  */
21
22 /**
23  * @author Sean Luke
```

```
24   * @version 1.0
25   */
26
27 public class Exp extends GPNode
28   {
29   public String toString() { return "exp"; }
30
31 /*
32  public void checkConstraints(final EvolutionState state,
33  final int tree,
34  final GPIndividual typicalIndividual,
35  final Parameter individualBase)
36  {
37  super.checkConstraints(state,tree,typicalIndividual,individualBase);
38  if (children.length!=1)
39  state.output.error("Incorrect number of children for node " +
40  toStringForError() + " at " +
41  individualBase);
42  }
43 */
44   public int expectedChildren() { return 1; }
45
46   public void eval(final EvolutionState state,
47     final int thread,
48     final GPData input,
49     final ADFStack stack,
50     final GPIndividual individual,
51     final Problem problem)
52     {
53     DoubleData rd = ((DoubleData)(input));
54
55     children[0].eval(state,thread,input,stack,individual,problem);
56     rd.x = /*Strict*/Math.exp(rd.x);
57     }
58   }
```

Listaus 7.16: evolution/nodes/FloatERC.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import ec.*;
10 import ec.app.regression.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 import java.io.*;
15
16 import thesis.evolution.DoubleData;
17
18
19 /*
20  * RegERC.java
21  *
22  * Created: Wed Nov 3 18:26:37 1999
23  * By: Sean Luke
24  */
25
26 /**
27  * @author Sean Luke
28  * @version 1.0
29  */
30
31 public class FloatERC extends ERC
32    {
33    public double value;
34
35    // Koza claimed to be generating from [-1.0, 1.0] but he wasn't,
36    // given the published simple-lisp code. It was [-1.0, 1.0). This is
37    // pretty minor, but we're going to go with the code rather than the
```

```
38    // published specs in the books. If you want to go with [-1.0, 1.0],
39    // just change nextDouble() to nextDouble(true, true)
40
41    public void resetNode(final EvolutionState state, final int thread)
42      { value = state.random[thread].nextDouble() * 2.0 - 1.0; }
43
44    public int nodeHashCode()
45      {
46      // a reasonable hash code
47      long l = Double.doubleToLongBits(value);
48      int iUpper = (int)(l & 0x00000000FFFFFFFF);
49      int iLower = (int)(l >>> 32);
50      return this.getClass().hashCode() + iUpper + iLower;
51      }
52
53    public boolean nodeEquals(final GPNode node)
54      {
55      // check first to see if we're the same kind of ERC --
56      // won't work for subclasses; in that case you'll need
57      // to change this to isAssignableTo(...)
58      if (this.getClass() != node.getClass()) return false;
59      // now check to see if the ERCs hold the same value
60      return (((FloatERC)node).value == value);
61      }
62
63    public void readNode(final EvolutionState state, final DataInput dataInput)
       throws IOException
64      {
65      value = dataInput.readDouble();
66      }
67
68    public void writeNode(final EvolutionState state, final DataOutput dataOutput)
       throws IOException
69      {
70      dataOutput.writeDouble(value);
71      }
72
```

```
73    public String encode()
74      { return Code.encode(value); }
75
76    public boolean decode(DecodeReturn dret)
77      {
78      // store the position and the string in case they
79      // get modified by Code.java
80      int pos = dret.pos;
81      String data = dret.data;
82
83      // decode
84      Code.decode(dret);
85
86      if (dret.type != DecodeReturn.T_DOUBLE) // uh oh!
87        {
88        // restore the position and the string; it was an error
89        dret.data = data;
90        dret.pos = pos;
91        return false;
92        }
93
94      // store the data
95      value = dret.d;
96      return true;
97      }
98
99    public String toStringForHumans()
100     { return "" + value; }
101
102   public void eval(final EvolutionState state,
103     final int thread,
104     final GPData input,
105     final ADFStack stack,
106     final GPIndividual individual,
107     final Problem problem)
108     {
109     DoubleData rd = ((DoubleData)(input));
```

```
110     rd.x = value;
111     }
112   }
```

Listaus 7.17: evolution/nodes/Iflte.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 /**
15  * A node representing if less than, else function in a program tree generated
       by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class Iflte extends GPNode {
22
23   public String toString() {
24     return "iflte";
25   }
26
27   public int expectedChildren() {
28     return 4;
29   }
30
31   public void eval(final EvolutionState state, final int thread,
```

```
32        final GPData input, final ADFStack stack,
33        final GPIndividual individual, final Problem problem) {
34      double condition1, condition2;
35      DoubleData rd = ((DoubleData) (input));
36
37      children[0].eval(state, thread, input, stack, individual, problem);
38      condition1 = rd.x;
39      children[1].eval(state, thread, input, stack, individual, problem);
40      condition2 = rd.x;
41      if (condition1 < condition2)
42        children[2].eval(state, thread, input, stack, individual, problem);
43      else
44        children[3].eval(state, thread, input, stack, individual, problem);
45    }
46  }
```

Listaus 7.18: evolution/nodes/Log.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 /**
15  * A node representing logarithm function in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
```

```
21 public class Log extends GPNode {
22
23   public String toString() {
24     return "log";
25   }
26
27   public int expectedChildren() {
28     return 1;
29   }
30
31   public void eval(final EvolutionState state, final int thread,
32       final GPData input, final ADFStack stack,
33       final GPIndividual individual, final Problem problem) {
34     double result;
35     DoubleData rd = ((DoubleData) (input));
36
37     children[0].eval(state, thread, input, stack, individual, problem);
38     result = rd.x;
39     if (result == 0)
40       result = 0;
41     else
42       result = Math.log(Math.abs(result));
43     rd.x = result;
44   }
45 }
```

Listaus 7.19: evolution/nodes/Mul.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
9  import thesis.evolution.DoubleData;
10 import ec.*;
```

```java
import ec.gp.*;

/**
 * A node representing multiplication in a program tree generated by genetic
 * programming.
 *
 * @author Sean Luke
 *
 */
public class Mul extends GPNode {

  public String toString() {
    return "*";
  }

  /*
   * public void checkConstraints(final EvolutionState state, final int tree,
   * final GPIndividual typicalIndividual, final Parameter individualBase) {
   * super.checkConstraints(state,tree,typicalIndividual,individualBase); if
   * (children.length!=2)
   * state.output.error("Incorrect number of children for node " +
   * toStringForError() + " at " + individualBase); }
   */
  public int expectedChildren() {
    return 2;
  }

  public void eval(final EvolutionState state, final int thread,
      final GPData input, final ADFStack stack,
      final GPIndividual individual, final Problem problem) {
    double result;
    DoubleData rd = ((DoubleData) (input));

    children[0].eval(state, thread, input, stack, individual, problem);
    result = rd.x;

    children[1].eval(state, thread, input, stack, individual, problem);
```

```
48      rd.x = result * rd.x;
49    }
50  }
```

Listaus 7.20: evolution/nodes/NegExp.java

```
1  /*
2   Copyright 2012 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.app.regression.*;
12 import ec.gp.*;
13 import ec.util.*;
14
15 /*
16  * NegExp.java
17  *
18  * Created: Mon Apr 23 17:15:35 EDT 2012
19  * By: Sean Luke
20
21  <p>This function appears in the Vladislavleva-B and Vladislavleva-C function
       sets, and is e^(-x)
22  <p>E. Vladislavleva, G. Smits, and D. Den Hertog. Order of Nonlinearity as a
       Complexity Measure for Models Generated by Symbolic Regression via Pareto
       Genetic Programming. <i>IEEE Trans EC,</i> 13(2):333-349, 2009. */
23
24 /**
25  * @author Sean Luke
26  * @version 1.0
27  */
28
29 public class NegExp extends GPNode
```

```
30    {
31    public String toString() { return "negexp"; }
32
33    public int expectedChildren() { return 1; }
34
35    public void eval(final EvolutionState state,
36      final int thread,
37      final GPData input,
38      final ADFStack stack,
39      final GPIndividual individual,
40      final Problem problem)
41      {
42      DoubleData rd = ((DoubleData)(input));
43
44      children[0].eval(state,thread,input,stack,individual,problem);
45      rd.x = Math.exp(0 - rd.x);
46      }
47    }
```

Listaus 7.21: evolution/nodes/Pow.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 /**
15  * A node representing power function in a program tree generated by genetic
16  * programming.
17  *
```

```
18   * @author Sean Luke
19   *
20   */
21  public class Pow extends GPNode {
22
23    public String toString() {
24      return "^";
25    }
26
27    public int expectedChildren() {
28      return 2;
29    }
30
31    public void eval(final EvolutionState state, final int thread,
32        final GPData input, final ADFStack stack,
33        final GPIndividual individual, final Problem problem) {
34      double result;
35      DoubleData rd = ((DoubleData) (input));
36
37      children[0].eval(state, thread, input, stack, individual, problem);
38      result = rd.x;
39      children[1].eval(state, thread, input, stack, individual, problem);
40      result = Math.pow(Math.abs(result), rd.x);
41      if (result == Double.NaN || result == Double.POSITIVE_INFINITY || result ==
            Double.NEGATIVE_INFINITY )
42        result = 0;
43      rd.x = result;
44    }
45  }
```

Listaus 7.22: evolution/nodes/Sin.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
```

```java
package thesis.evolution.nodes;

import thesis.evolution.DoubleData;
import ec.*;
import ec.gp.*;
import ec.util.*;

/**
 * A node representing sine function in a program tree generated by genetic
 * programming.
 *
 * @author Sean Luke
 *
 */
public class Sin extends GPNode {

  public String toString() {
    return "sin";
  }

  public int expectedChildren() {
    return 1;
  }

  public void eval(final EvolutionState state, final int thread,
      final GPData input, final ADFStack stack,
      final GPIndividual individual, final Problem problem) {
    double result;
    DoubleData rd = ((DoubleData) (input));

    children[0].eval(state, thread, input, stack, individual, problem);
    rd.x = Math.sin(rd.x);
  }
}
```

Listaus 7.23: evolution/nodes/Sqrt.java

```java
/*
```

```
2   Copyright 2012 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.app.regression.*;
12 import ec.gp.*;
13 import ec.util.*;
14
15 /*
16  * Sqrt.java
17  *
18  * Created: Mon Apr 23 17:15:35 EDT 2012
19  * By: Sean Luke
20
21  <p>This function appears in the Korns and Keijzer function sets, and is sqrt(x)
22  <p>M. F. Korns. Accuracy in Symbolic Regression. In <i>Proc. GPTP.</i> 2011.
23  <p>M. Keijzer. Improving Symbolic Regression with Interval Arithmetic and
        Linear Scaling. In <i>Proc. EuroGP.</i> 2003.
24
25 */
26
27 /**
28  * @author Sean Luke
29  * @version 1.0
30  */
31
32 public class Sqrt extends GPNode
33   {
34   public String toString() { return "sqrt"; }
35
36   public int expectedChildren() { return 1; }
37
```

```
38   public void eval(final EvolutionState state,
39     final int thread,
40     final GPData input,
41     final ADFStack stack,
42     final GPIndividual individual,
43     final Problem problem)
44     {
45     DoubleData rd = ((DoubleData)(input));
46
47     children[0].eval(state,thread,input,stack,individual,problem);
48     rd.x = Math.sqrt(Math.abs(rd.x));
49     }
50   }
```

Listaus 7.24: evolution/nodes/Square.java

```
1  /*
2   Copyright 2012 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.app.regression.*;
12 import ec.gp.*;
13 import ec.util.*;
14
15 /*
16  * Square.java
17  *
18  * Created: Mon Apr 23 17:15:35 EDT 2012
19  * By: Sean Luke
20
21  <p>This function appears in the Korns and all three Vladislavleva function sets
       , and is x * x
```

```
22   <p>M. F. Korns. Accuracy in Symbolic Regression. In <i>Proc. GPTP.</i> 2011.
23   <p>E. Vladislavleva, G. Smits, and D. Den Hertog. Order of Nonlinearity as a
        Complexity Measure for Models Generated by Symbolic Regression via Pareto
        Genetic Programming. <i>IEEE Trans EC,</i> 13(2):333-349, 2009.
24   */
25
26   /**
27    * @author Sean Luke
28    * @version 1.0
29    */
30
31   public class Square extends GPNode
32     {
33     public String toString() { return "square"; }
34
35     public int expectedChildren() { return 1; }
36
37     public void eval(final EvolutionState state,
38       final int thread,
39       final GPData input,
40       final ADFStack stack,
41       final GPIndividual individual,
42       final Problem problem)
43       {
44       DoubleData rd = ((DoubleData)(input));
45
46       children[0].eval(state,thread,input,stack,individual,problem);
47       rd.x = rd.x * rd.x;
48       }
49     }
```

Listaus 7.25: evolution/nodes/Sub.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
```

```
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.gp.*;
12
13 /**
14  * A node representing addition in a program tree generated by genetic
15  * programming.
16  *
17  * @author Sean Luke
18  *
19  */
20 public class Sub extends GPNode
21   {
22
23   public String toString() { return "-"; }
24
25 /*
26  public void checkConstraints(final EvolutionState state,
27  final int tree,
28  final GPIndividual typicalIndividual,
29  final Parameter individualBase)
30  {
31  super.checkConstraints(state,tree,typicalIndividual,individualBase);
32  if (children.length!=2)
33  state.output.error("Incorrect number of children for node " +
34  toStringForError() + " at " +
35  individualBase);
36  }
37 */
38   public int expectedChildren() { return 2; }
39
40   public void eval(final EvolutionState state,
41     final int thread,
42     final GPData input,
```

```
43      final ADFStack stack,
44      final GPIndividual individual,
45      final Problem problem)
46      {
47      double result;
48      DoubleData rd = ((DoubleData)(input));
49
50      children[0].eval(state,thread,input,stack,individual,problem);
51      result = rd.x;
52
53      children[1].eval(state,thread,input,stack,individual,problem);
54      rd.x = result - rd.x;
55      }
56    }
```

Listaus 7.26: evolution/nodes/Tan.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.gp.*;
12 import ec.util.*;
13
14 /**
15  * A node representing tangent function in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class Tan extends GPNode {
```

```
22
23    public String toString() {
24      return "tan";
25    }
26
27    public int expectedChildren() {
28      return 1;
29    }
30
31    public void eval(final EvolutionState state, final int thread,
32        final GPData input, final ADFStack stack,
33        final GPIndividual individual, final Problem problem) {
34      double result;
35      DoubleData rd = ((DoubleData) (input));
36
37      children[0].eval(state, thread, input, stack, individual, problem);
38      result = Math.tan(rd.x);
39      if (result == Double.NaN || result == Double.POSITIVE_INFINITY || result ==
            Double.NEGATIVE_INFINITY )
40        result = 0;
41      rd.x = result;
42    }
43  }
```

Listaus 7.27: evolution/nodes/Tanh.java

```
1  /*
2   Copyright 2012 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import ec.*;
11 import ec.app.regression.*;
12 import ec.gp.*;
```

```
13 import ec.util.*;
14
15 /*
16  * Tanh.java
17  *
18  * Created: Mon Apr 23 17:15:35 EDT 2012
19  * By: Sean Luke
20
21
22  <p>This function appears in the Korns function set, and is just tanh(x)
23  <p>M. F. Korns. Accuracy in Symbolic Regression. In <i>Proc. GPTP.</i> 2011.
24
25 */
26
27 /**
28  * @author Sean Luke
29  * @version 1.0
30  */
31
32 public class Tanh extends GPNode
33   {
34   public String toString() { return "tanh"; }
35
36   public int expectedChildren() { return 1; }
37
38   public void eval(final EvolutionState state,
39     final int thread,
40     final GPData input,
41     final ADFStack stack,
42     final GPIndividual individual,
43     final Problem problem)
44     {
45     DoubleData rd = ((DoubleData)(input));
46
47     children[0].eval(state,thread,input,stack,individual,problem);
48     rd.x = Math.tanh(rd.x);
49     }
```

```
50    }
```

Listaus 7.28: evolution/nodes/U.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import thesis.evolution.ThesisProblem;
11 import ec.*;
12 import ec.gp.*;
13
14 /**
15  * A node representing an input parameter in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class U extends GPNode
22    {
23
24    public String toString() { return "z"; }
25
26 /*
27  public void checkConstraints(final EvolutionState state,
28  final int tree,
29  final GPIndividual typicalIndividual,
30  final Parameter individualBase)
31  {
32  super.checkConstraints(state,tree,typicalIndividual,individualBase);
33  if (children.length!=0)
34  state.output.error("Incorrect number of children for node " +
```

```
35   toStringForError() + " at " +
36   individualBase);
37   }
38 */
39    public int expectedChildren() { return 0; }
40
41    public void eval(final EvolutionState state,
42      final int thread,
43      final GPData input,
44      final ADFStack stack,
45      final GPIndividual individual,
46      final Problem problem)
47      {
48      DoubleData rd = ((DoubleData)(input));
49      rd.x = ((ThesisProblem)problem).currentU;
50      }
51    }
```

Listaus 7.29: evolution/nodes/W.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import thesis.evolution.ThesisProblem;
11 import ec.*;
12 import ec.gp.*;
13
14 /**
15  * A node representing an input parameter in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
```

```
19   *
20   */
21  public class W extends GPNode
22    {
23
24    public String toString() { return "w"; }
25
26  /*
27   public void checkConstraints(final EvolutionState state,
28   final int tree,
29   final GPIndividual typicalIndividual,
30   final Parameter individualBase)
31   {
32   super.checkConstraints(state,tree,typicalIndividual,individualBase);
33   if (children.length!=0)
34   state.output.error("Incorrect number of children for node " +
35   toStringForError() + " at " +
36   individualBase);
37   }
38  */
39    public int expectedChildren() { return 0; }
40
41    public void eval(final EvolutionState state,
42      final int thread,
43      final GPData input,
44      final ADFStack stack,
45      final GPIndividual individual,
46      final Problem problem)
47      {
48      DoubleData rd = ((DoubleData)(input));
49      rd.x = ((ThesisProblem)problem).currentW;
50      }
51    }
```

Listaus 7.30: evolution/nodes/X.java

```
1  /*
2   Copyright 2006 by Sean Luke
```

```
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5   */
6
7  package thesis.evolution.nodes;
8
9  import thesis.evolution.DoubleData;
10 import thesis.evolution.ThesisProblem;
11 import ec.*;
12 import ec.gp.*;
13
14 /**
15  * A node representing an input parameter in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class X extends GPNode {
22
23   public String toString() {
24     return "x";
25   }
26
27   /*
28    * public void checkConstraints(final EvolutionState state, final int tree,
29    * final GPIndividual typicalIndividual, final Parameter individualBase) {
30    * super.checkConstraints(state,tree,typicalIndividual,individualBase); if
31    * (children.length!=0)
32    * state.output.error("Incorrect number of children for node " +
33    * toStringForError() + " at " + individualBase); }
34    */
35   public int expectedChildren() {
36     return 0;
37   }
38
39   public void eval(final EvolutionState state, final int thread,
```

```
40        final GPData input, final ADFStack stack,
41        final GPIndividual individual, final Problem problem) {
42      DoubleData rd = ((DoubleData) (input));
43      rd.x = ((ThesisProblem) problem).currentX;
44    }
45  }
```

Listaus 7.31: evolution/nodes/Y.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import thesis.evolution.ThesisProblem;
11 import ec.*;
12 import ec.gp.*;
13
14 /**
15  * A node representing an input parameter in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class Y extends GPNode
22   {
23
24   public String toString() { return "y"; }
25
26 /*
27  public void checkConstraints(final EvolutionState state,
28  final int tree,
29  final GPIndividual typicalIndividual,
```

```
30   final Parameter individualBase)
31   {
32   super.checkConstraints(state,tree,typicalIndividual,individualBase);
33   if (children.length!=0)
34   state.output.error("Incorrect number of children for node " +
35   toStringForError() + " at " +
36   individualBase);
37   }
38  */
39    public int expectedChildren() { return 0; }
40
41    public void eval(final EvolutionState state,
42      final int thread,
43      final GPData input,
44      final ADFStack stack,
45      final GPIndividual individual,
46      final Problem problem)
47      {
48      DoubleData rd = ((DoubleData)(input));
49      rd.x = ((ThesisProblem)problem).currentY;
50      }
51    }
```

Listaus 7.32: evolution/nodes/Z.java

```
1  /*
2   Copyright 2006 by Sean Luke
3   Licensed under the Academic Free License version 3.0
4   See the file "LICENSE" for more information
5  */
6
7
8  package thesis.evolution.nodes;
9  import thesis.evolution.DoubleData;
10 import thesis.evolution.ThesisProblem;
11 import ec.*;
12 import ec.gp.*;
13
```

```
14 /**
15  * A node representing an input parameter in a program tree generated by genetic
16  * programming.
17  *
18  * @author Sean Luke
19  *
20  */
21 public class Z extends GPNode
22   {
23
24   public String toString() { return "z"; }
25
26 /*
27  public void checkConstraints(final EvolutionState state,
28  final int tree,
29  final GPIndividual typicalIndividual,
30  final Parameter individualBase)
31  {
32  super.checkConstraints(state,tree,typicalIndividual,individualBase);
33  if (children.length!=0)
34  state.output.error("Incorrect number of children for node " +
35  toStringForError() + " at " +
36  individualBase);
37  }
38 */
39   public int expectedChildren() { return 0; }
40
41   public void eval(final EvolutionState state,
42     final int thread,
43     final GPData input,
44     final ADFStack stack,
45     final GPIndividual individual,
46     final Problem problem)
47     {
48     DoubleData rd = ((DoubleData)(input));
49     rd.x = ((ThesisProblem)problem).currentZ;
50     }
```

```
51    }
```

Listaus 7.33: rmi/PolicyFileLocator.java

```
1  package thesis.rmi;
2
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.io.InputStream;
8
9  /**
10  * class to locate our most "secure" policy file
11  *
12  * @author srasul (http://code.nomad-labs.com/)
13  *
14  */
15 public class PolicyFileLocator {
16
17   public static final String POLICY_FILE_NAME = "/allow_all.policy";
18
19   public static String getLocationOfPolicyFile() {
20     try {
21       File tempFile = File.createTempFile("rmi-base", ".policy");
22       InputStream is = PolicyFileLocator.class.getResourceAsStream(
     POLICY_FILE_NAME);
23       BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));
24       int read = 0;
25       while((read = is.read()) != -1) {
26         writer.write(read);
27       }
28       writer.close();
29       tempFile.deleteOnExit();
30       return tempFile.getAbsolutePath();
31     }
32     catch(IOException e) {
33       throw new RuntimeException(e);
```

```
34      }
35    }
36  }
```

Listaus 7.34: rmi/PotentialFunctionProvider.java

```
1  package thesis.rmi;
2
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5
6  /**
7   * The interface for a potential function provider. The potential function is a
8   * function that guides the AI player's units. The potential field values are
9   * calculated in the locations nearest to the unit and the unit then moves in
10  * direction of the highest potential next to it.
11  *
12  * @author Oskari Leppäaho
13  *
14  */
15 public interface PotentialFunctionProvider extends Remote {
16    /**
17     * The name of the remote service.
18     */
19    public static final String SERVICE_NAME = "PotentialFunctionService";
20
21    /**
22     * Returns the potential based on the distance from one enemy unit and the
23     * map edges.
24     *
25     * TODO: Modify to use multiple enemy units.
26     *
27     * @param distancesFromOwnUnits
28     *
29     * @param distanceFromEnemy
30     *      Distance between the AI player's unit that the potential is
31     *      being calculated for and the enemy unit.
32     * @param distancesFromEdges
```

```
33      *      Current unit's distance from the 4 map edges.
34      * @param onCooldown
35      *      Indicates if the unit is on cooldown.
36      * @param relativeHP
37      *      Unit's HP amount relative to other own units.
38      * @param enemyPositionVectors
39      * @return The potential value.
40      * @throws RemoteException
41      *       If something goes wrong with the remote function call.
42      */
43     public double getPotential(double distancesFromEnemies[],
44         double[] distancesFromOwnUnits, double ownMaximumShootDistance,
45         double[] distancesFromEdges, boolean onCooldown, double relativeHP, double
            [][] enemyPositionVectors)
46         throws RemoteException;
47 }
```

Listaus 7.35: rmi/RemoteBotInterface.java

```
1  package thesis.rmi;
2
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5
6  /**
7   * The interface for a remote bot.
8   *
9   * @author Oskari Leppäaho
10  *
11  */
12 public interface RemoteBotInterface extends Remote {
13    public static final String SERVICE_NAME = "BotService";
14
15    /**
16     * Gets the score of a single round of fighting.
17     *
18     * @return Score for the round.
19     * @throws InterruptedException
```

```
20      *       If interrupted.
21      */
22     public int getRoundScore(PotentialFunctionProvider problem) throws
           RemoteException;
23 }
```

Listaus 7.36: rmi/RemoteBotRMI.java

```
1  package thesis.rmi;
2
3  import java.rmi.registry.LocateRegistry;
4  import java.rmi.registry.Registry;
5  import java.rmi.server.UnicastRemoteObject;
6
7  import thesis.bot.Controller;
8
9  /**
10  * Starter for running a bot that can be accessed remotely. Takes one command
11  * line argument that will be the unique part of the name that can be used to
12  * access the started bot.
13  *
14  * @author Oskari Leppäaho
15  *
16  */
17 public class RemoteBotRMI extends RmiStarter {
18
19   static String currentName;
20
21   public RemoteBotRMI() {
22     super(RemoteBotInterface.class);
23   }
24
25   public static void main(String[] args) {
26     currentName = args[0];
27     new RemoteBotRMI();
28   }
29
30   /**
```

```
31      * Adds the bot to the RMI registry.
32      */
33     @Override
34     public void doCustomRmiHandling() {
35       try {
36         Registry registry = LocateRegistry.getRegistry();
37         RemoteBotInterface remoteBot = new Controller(currentName);
38         RemoteBotInterface remoteBotStub = (RemoteBotInterface)
               UnicastRemoteObject
39             .exportObject(remoteBot, 0);
40
41         registry.rebind(RemoteBotInterface.SERVICE_NAME + currentName,
42             remoteBotStub);
43         System.out.println("Bind " + RemoteBotInterface.SERVICE_NAME
44             + currentName);
45       } catch (Exception e) {
46         e.printStackTrace();
47       }
48
49     }
50
51 }
```

Listaus 7.37: rmi/RmiStarter.java

```
1  package thesis.rmi;
2
3
4  /**
5   * class to do some common things for client & server to get RMI working
6   *
7   * @author srasul (http://code.nomad-labs.com/)
8   *
9   */
10 public abstract class RmiStarter {
11
12   /**
13    *
```

```
14      * @param clazzToAddToServerCodebase
15      *       a class that should be in the java.rmi.server.codebase
16      *       property.
17      */
18     public RmiStarter(Class clazzToAddToServerCodebase) {
19
20       System.setProperty("java.rmi.server.codebase",
21           clazzToAddToServerCodebase.getProtectionDomain()
22               .getCodeSource().getLocation().toString());
23
24       System.setProperty("java.security.policy",
25           PolicyFileLocator.getLocationOfPolicyFile());
26
27       if (System.getSecurityManager() == null) {
28         System.setSecurityManager(new SecurityManager());
29       }
30
31       doCustomRmiHandling();
32     }
33
34     /**
35      * extend this class and do RMI handling here
36      */
37     public abstract void doCustomRmiHandling();
38
39 }
```

Listaus 7.38: rmi/ThesisProblemRMI.java

```
1  package thesis.rmi;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.rmi.RemoteException;
7  import java.rmi.registry.LocateRegistry;
8  import java.rmi.registry.Registry;
9  import java.rmi.server.UnicastRemoteObject;
```

```
10
11 import test.TestPotentialFunctionProvider;
12
13 /**
14  * For testing the RMI connection.
15  *
16  * @author Oskari Leppäaho
17  */
18
19 public class ThesisProblemRMI extends RmiStarter {
20
21   Registry registry;
22   PotentialFunctionProvider provider;
23 //  ArrayList<RemoteBotInterface> bots;
24
25   public ThesisProblemRMI() {
26     super(PotentialFunctionProvider.class);
27   }
28
29   @Override
30   public void doCustomRmiHandling() {
31     try {
32       provider = new TestPotentialFunctionProvider();
33       PotentialFunctionProvider providerStub = (PotentialFunctionProvider)
             UnicastRemoteObject.exportObject(provider, 0);
34
35       registry = LocateRegistry.getRegistry();
36       registry.rebind(PotentialFunctionProvider.SERVICE_NAME, providerStub);
37     }
38     catch(Exception e ){
39       e.printStackTrace();
40     }
41   }
42
43   public RemoteBotInterface connectClient(String name) {
44     try {
```

```
45          return (RemoteBotInterface)registry.lookup(RemoteBotInterface.SERVICE_NAME
                + name);
46        } catch (Exception e ) {
47          // TODO Auto-generated catch block
48          e.printStackTrace();
49        }
50        return null;
51      }
52
53      public static void main(String[] args) throws RemoteException {
54        ThesisProblemRMI starter = new ThesisProblemRMI();
55
56        RemoteBotInterface client1;
57        //RemoteBotInterface client2;
58
59        BufferedReader buffer = new BufferedReader(new InputStreamReader(System.in))
              ;
60
61        System.out.println("Press enter when ready to initialize next client >");
62        try {
63          buffer.readLine();
64        } catch (IOException e) {
65          System.err.println("Something went wrong when waiting for \n"
66              + "the user to press enter on the console\n"
67              + "(for stepping).");
68          e.printStackTrace();
69        }
70        client1 = starter.connectClient("0");
71  //      System.out.println("Press enter when ready to initialize next client >");
72  //      try {
73  //        buffer.readLine();
74  //      } catch (IOException e) {
75  //        System.err.println("Something went wrong when waiting for \n"
76  //            + "the user to press enter on the console\n"
77  //            + "(for stepping).");
78  //        e.printStackTrace();
79  //      }
```

```
80 //     starter.connectClient("2");
81
82     System.out.println("Get score from client 1");
83     System.out.println(client1.getRoundScore(starter.provider));
84   }
85 }
```