POHJOIS-KARJALAN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Juuso Hietala

TEKOÄLY JA SEN SOVELLUTUKSET PELEISSÄ CASE STUDY: DROIDWARS

Opinnäytetyö
Marraskuu 2012

POHJOIS-KARJALAN
AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Marraskuu 2012
Tietojenkäsittelyn koulutusohjelma

Länsikatu
80110 JOENSUU
p. 050 311 6310

Tekijä
Juuso Hietala

Nimeke
Tekoälyn sovellutukset peleissä. Case Study: DroidWars

Toimeksiantaja
-

Tiivistelmä

Opinnäytetyössä tarkastellaan tekoälyä ja sen sovellutuksia peleissä. Tutkimuskysymykset olivat, pystynkö tekemään jo olemassa olevan pelin kaltaisen pelin Androidille, pystynkö suunnittelemaan peliin toimivan tekoälyvastustajan ja miten pelin toiminnot pitää suunnitella että niitä pystyy käyttämään sekä tekoäly- että ihmispelaaja.

Tekoälyä tarkasteltiin ensin yleisenä terminä. Vahva tekoäly tarkoittaa konetta joka pystyy oppimaan ja soveltamaan asioita itsenäisesti. Heikko tekoäly tarkoittaa konetta, joka osaa ratkoa ongelmia sille annettujen sääntöjen mukaisesti. Tekoälyä tarkasteltiin myös pelien näkökulmasta ja esiteltiin tekoälyn käytön historiaa peleissä sekä tekoälyn hoitamia tehtäviä peleissä, jotka sisältävät muun muassa reitinetsinnän ja tekoälyvastustajan hallinnan.

Tekoälyn sovellutusta tutkittiin toteuttamalla DroidWars -peli Android -alustalle ja toteuttamalla siihen toimiva tekoälyvastustaja. Tarkastelun kohteena olivat DroidWarsin mekaniikat, tekoälyn tehtävät ja tekoälyvastustajan toimintalogiikka. Tarkastelun kohteena olivat myös pelimoottorivaihtoehdot toteutusta varten sekä toteutukseen käytetyt työkalut. Työssä käsitellään myös toteutusprosessia sekä sen eri vaiheita.

Opinnäytetyön tuloksena syntyi DroidWars-peli, jossa on mahdollisuus pelata joko tekoälyä tai ihmispelaajaa vastaan.

Kieli	Sivuja 40
suomi	Liitteet 2

Asiasanat
Tekoäly, peliohjelmointi, Android

NORTH KARELIA
UNIVERSITY OF APPLIED SCIENCES

THESIS
November 2012
Degree Programme in Business Information Technology

Länsikatu 15
80200 JOENSUU
FINLAND
Tel. +358-50 311 6310

Author
Juuso Hietala

Title
Application of Artificial Intelligence in Games. Case Study: DroidWars

Commissioned by
-

Abstract

The thesis focuses on artificial intelligence and its usage in games. The research questions were as follows: Can I make a game on Android-platform that resembles an already existing game, can I design a working artificial intelligence controlled opponent and how the functions of the game need to be designed so that both human and artificial intelligence players can use them.

In this thesis, artificial intelligence was first examined as a general term and it was found that strong artificial intelligence refers to a machine that can learn and adapt things by itself. A weak artificial intelligence is a machine that can solve problems according to the given rules/instructions. Artificial intelligence was also examined from the viewpoint of games. First the history of artificial intelligence usage in games and the tasks that artificial intelligence performs, for example, path finding and the controlling an opponent were presented.

The application of artificial intelligence was studied by developing DroidWars for anroid-platform and developing a working artificial intelligence opponent. The subjects examined were the mechanics of DroidWars, the tasks of the artificial intelligence and the logic behind the artificial intelligence opponent.

The project resulted in a game called DroidWars in which you can either play against a another human player or an artificial intelligence controlled opponent.

Language
Finnish

Pages 40

Appendices 2

Keywords
Artificial Intelligence, game programming, Android

Sisältö

1 Johdanto 5
2 Tekoälyn teoriaa 6
2.1 Vahva tekoäly 6
2.2 Heikko tekoäly 8
2.3 Tekoälyn historia peleissä 9
2.4 Tekoälyn tehtävistä peleissä 11
2.5 Reitinetsintäalgoritmit 15
2.6 Vuoropohjaisen strategiapelin tekoäly 17
3 Case: Droidwars 19
3.1 Droidwarsin mekaniikat 19
3.2 Droidwars ja tekoäly 21
3.2.1 Tekoälyn tehtävät Droidwarsissa 22
3.2.2 Droidwarsin tekoälyvastustaja 23
3.3 Pelimoottorin valinta 29
3.3.1 Unity3D 30
3.3.2 AndEngine 30
3.3.3 Itse tehty pelimoottori 31
3.3.4 Pelimoottorin Valinta 31
3.4 Käytetyt ohjelmat 32
3.5 Toteutus 33
4. Pohdinta 38
4.1 Sisällön ja tulosten tarkastelu 38
4.2 Toteutuksen ja menetelmän tarkastelu 38
4.3 Oppimisprosessi 40
4.4 Kehittämisideat 40
Lähteet 42

Liitteet

Liite 1 A* Reitinetsinnän pseudokoodi
Liite 2 Tekoälyn toiminnan pseudokoodi

1 Johdanto

Kiinnostukseni tekoälyyn heräsi harjoitteluni aikana, kun sain työkseni toteuttaa harjoittelufirmani peliin vihollisten tekoälyn. Työ osoittautui hyvin mielenkiintoiseksi ja tekoälyn toteutus omaan tarkoitukseen jäikin mietintään. Opinnäytetyötä suunnitellessani sain ajatuksen tehdä strategiapelin ja strategiapeleissä tekoälyä tarvitaankin useaan eri tehtävään. Otin mallikseni jo olemassa olevan strategiapelin. Pelin mekaniikoiden toteuttamisen lisäksi halusin lisätä peliin myös tekoälyvastustajan. Tarkoitukseni on selvittää, miten suunnittelen ja toteutan Android-alustalla toimivan pelin ja tekoälyvastustajan sekä miten pelin toiminnot pitää suunnitella että niitä pystyy käyttämään sekä ihmispelaaja että tekoälyvastustaja.

Toisessa luvussa pureudutaan tekoälyn teoriapuoleen. Siinä selvitetään, mitä tekoäly on, mihin eri tyyppeihin se jaetaan ja miten tekoälyä testataan. Tarkastelen myös tekoälyä pelien kehittämisen näkökulmasta. Milloin tekoälyä käytettiin ensimmäisen kerran peleissä, millaisia tehtäviä tekoäly hoitaa peleissä ja läpi käydään myös muutama hyvin tunnettu reitinetsintäalgoritmi.

Kolmannessa luvussa siirrytään Droidwarsin tapaustutkimukseen. Aluksi esitellään pelin mekaniikat eli millaisten sääntöjen puitteissa peliä pelataan. Mekaniikkojen jälkeen käsitellään tekoälyn ja miten sitä sovelletaan droidwarsissa. Esitellyssä ovat eri tehtävät ja miten droidwarsin tekoälyvastustaja toimii.

Mekaniikoiden ja tekoälyn jälkeen siirrytään pelin toteuttamispuoleen. Ensiksi arvioidaan erialisia vaihtoehtoja pelimoottoriksi sekä niiden hyviä ja huonoja puolia pelini toteutuksen kannalta. Viimeisenä yhteenvetona ovat kehityksessä käytetyt ohjelmat.

2 Tekoälyn teoriaa

Tekoälystä puhutaan keinotekoisen olennon osoittamana älynä. Yleensä sen oletetaan olevan tietokoneohjelma, joka pyrkii suorittamaan ihmismäistä ajattelua. Tekoälyllä voidaan tarkoittaa myös tietokonetieteen haaraa, joka pyrkii sen luomiseen. Kyseinen haara perustettiin todistamaan väittämää, jonka mukaan ihmisten älykkyys voidaan määritellä niin tarkasti, että kone kykenee simuloimaan sitä. (Poole, Markworth, Goebel 1998.) Mekaanista päättelyä, joka on tekoälyn perusta, on kehitetty jo antiikin ajoista asti (McCorduck 2004). Tämän päättelytavan tutkimus johti lopulta nykyisen tietokoneen kehittämiseen Alan Turningin ja muiden asiaa tutkineiden työn pohjalta.

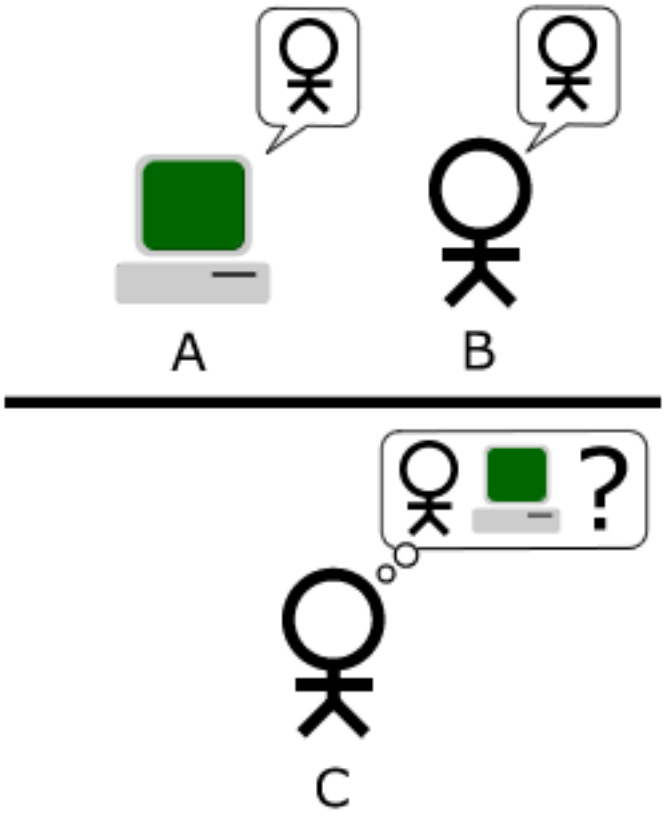
On tärkeää määritellä, puhutaanko vahvasta vai heikosta tekoälystä. Molemmat toimivat samoilla periaatteilla mutta niillä on toisistaan poikkeava olemus. Seuraavissa luvuissa käydään läpi sekä heikko että vahva tekoäly ja tarkastellaan niiden määritelmiä, tarkoituksia ja eroja.

2.1 Vahva tekoäly

Vahva tekoäly -termillä tarkoitetaan konetta, joka pystyy samoihin toimintoihin kun ihmisen aivot. Suurin ongelma vahvan tekoälyn kanssa on sen määriteltävyys. Tällä hetkellä koneiden ”älykkyyden” määritelmästä ei ole päästy yksimielisyyteen. Tekoälytermin keksinyt John McCarthy (2007) totesi asiasta: ” Ongelma on se, ettemme voi vielä karakterisoida millaisia laskennallisia menetelmiä haluamme kutsua älykkäiksi. Me ymmärrämme joitain älykkyyden mekanismeja muttemme joitain muita.”

Älykkyyttä mittaamaan on tehty kuitenkin useita testejä. Näistä tunnetuin on Alan Turingin kehittämä Turingin testi. Kuvassa 1 näkyy Turingin testi, joka sisältää henkilöt B ja C ja tietokoneen A. Henkilö C on kuulustelija, joka tuntee kuulusteltavat vain A:na ja B:nä. Hän kuulustelee A:ta B:tä kirjoitettujen viestien välityksellä. Kuulustelija voi tehdä kuulusteltaville esimerkiksi seuraavanlaisia kysymyksiä: "Voisiko A kertoa minulle pelaako hän shakkia?" Tietokoneen tehtävä testissä on saada kuulustelija luulemaan, että se on ihminen. Kuulusteltavan ihmisen tehtävä on auttaa kuulustelijaa tieto-

koneen tunnistamisessa. Testin lopussa kuulustelija tekee kuulustelunsa perusteella päätöksen siitä, kumpi A:sta ja B:stä on tietokone ja kumpi ihminen. Tämä on vain yksi versio Turingin testistä, mutta tätä pidetään sen normaalina tulkintana (Oppy, Graham, Dowe, David 2011).



Kuva 1. Turingin testi (Kuva: Wikimedia Commons 2008).

Muita Turingin testin versiota ovat mm. käänteinen Turingin testi, jossa tietokone yrittää selvittää, keskusteleeko se ihmisen vai tietokoneen kanssa. Erästä käänteisen Turingin testin muotoa, CAPTCHA:aa, käytetään aktiivisesti esimerkiksi keskustelufoorumeille rekisteröidyttäessä. Näyttämällä vääristettyä tekstiä ja pyytämällä käyttäjää kirjoittamaan se, voidaan olla lähes varmoja että käyttäjä on ihminen. Toinen esimerkki muokatusta Turingin testistä on Feigenbaumin testi, jossa kuulusteltavan tietokoneen täytyy esiintyä jonkin asia-alueen asiantuntijana (Feigenbaum 2003).

Muitakin testejä on olemassa mutta niistä yksikään, Turningin testi mukaan lukien, ei ole onnistunut vakuuttamaan kaikkia tutkijoita. Käsitykset, jotka tekevät tekoälystä vahvan, vaihtelevat paljon sen mukaan keneltä asiaa kysytään (Goertzel 2007). Yksi vahvan tekoälyn tutkimussuunta on ”Artificial General Intelligence (AGI)”, joka eroaa normaalista tekoälytutkimuksesta siinä, että kyseisellä tavalla kehitetty tekoäly ei ole

erikoistunut mihinkään tiettyyn asiaan vaan kone voi oppia aivan mitä vain ja oppimansa avulla ratkaista mitä tahansa ongelmia tai suorittaa mitä tahansa tehtäviä. Kone pystyy siis itsenäisesti oppimaan asioita joko niitä tarkastelemalla tai asioiden kanssa itsenäisesti toimimalla. Oppiminen on kumulatiivista, joten kone oppii uusia asioita ja saattaa muuttaa muistissaan olevia asioita oppimiensa asioiden perusteella. Kone osaa myös toimia sille annetun tehtävän mukaan ja kerätä vain sellaista tietoa, joka on tarpeellista annetun tehtävän kannalta. AGI:n mukaisen tekoälyyn ei ole tarkoitus ohjelmoida tietoa, vaan sen on tarkoitus hankkia, kehittää ja käyttää tietoa täysin itsenäisesti. AGI on saanut kritiikkiä osakseen useilta eri tahoilta, koska jotkut pitävät tämän suuntauksen pyrkimyksiä mahdottomina. (Goertzel 2007.)

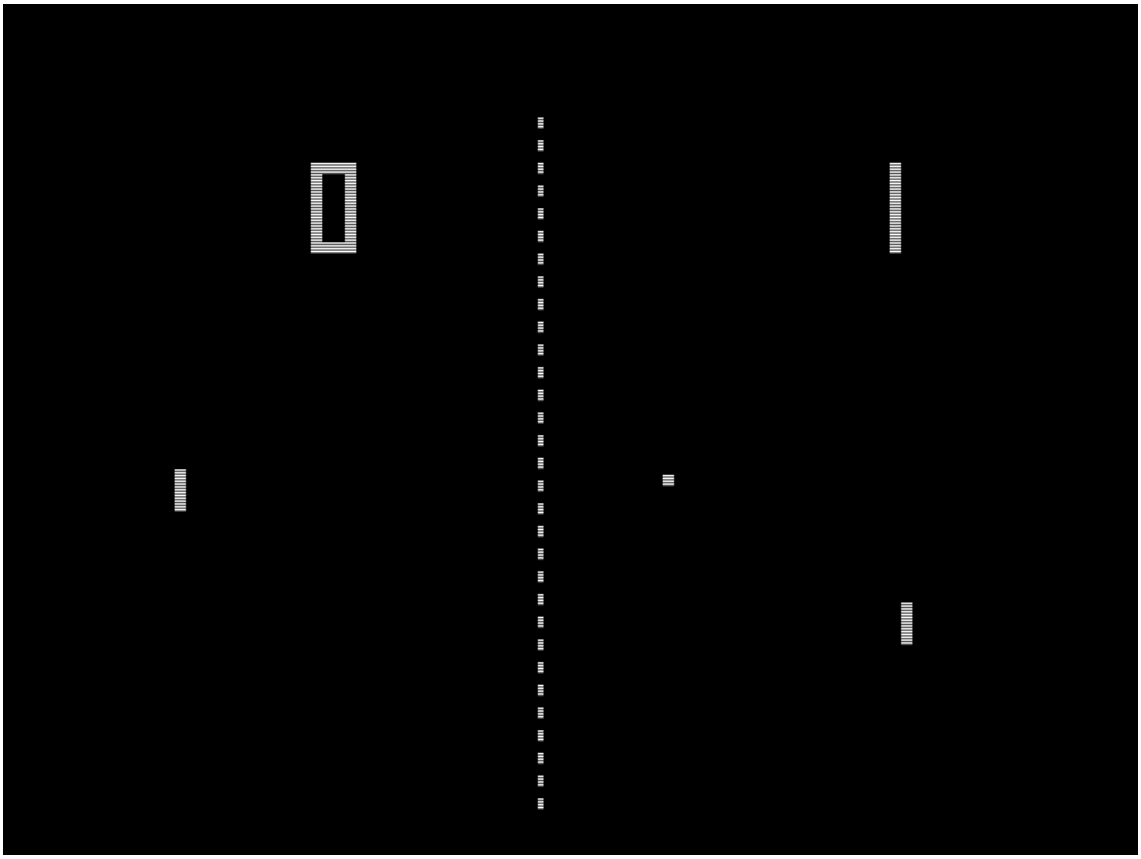
2.2 Heikko tekoäly

Heikko tekoäly toimii samoilla periaatteilla kuin vahva tekoäly, mutta määritelmä on todella erilainen. Kone, jolla on heikko tekoäly, osaa tarkastella ja ratkoa ongelmia, mutta kone ei hoida ongelmanratkontaa samalla tavalla kuin ihminen, vaan se ratkoo ongelman sille ohjelmoitujen sääntöjen mukaisesti. Hyvänä esimerkkinä heikosta tekoälystä on IBM:n kehittämä Deep Blue joka on shakin peluuseen erikoistunut tietokone. Shakkia pidetään yleisesti pelinä, joka vaatii paljon suunnittelua, ajattelua ja taktikointia. Ihminen tekee siirtojaan kokemuksensa ja mielikuvituksensa perusteella, kun Deep Blue taas analysoi koko pelilaudan sen hetkisen tilanteen ja simuloi kaikki mahdolliset liikkeet**.** Se valitsee niistä parhaan eliminoimalla liikkeitä, kunnes jäljelle jäi vain paras mahdollinen liike**.** Kone ei tietenkään ollut itse oppinut, mitkä shakin säännöt ovat tai mikä on hyvä ja mikä on huono liike, vaan se määritteli kyseiset asiat sille ohjelmoitujen sääntöjen mukaan. Toinen esimerkki heikosta tekoälystä on tekstinkäsittelyohjelman oikeinkirjoituksen tarkistus. Kone tai ohjelma ei huomaa väärin kirjoitettua samalla tavalla kuin esimerkiksi opettaja koulussa, vaan se vertaa aina kirjoitettua sanaa tuntemiinsa sanoihin.

Yhteenvetona voidaan todeta, että heikko tekoäly eroaa vahvasta siinä, että heikolle tekoälylle annetaan tietty määrä tietoja ja säännöt, joiden puitteissa se suorittaa sille annettua tehtävää käyttäen näitä annettuja tietoja. Vahva tekoäly saattaa taas oppia uusia asioita itsenäisesti ja saattaa osata monia eri asioita.

2.3 Tekoälyn historia peleissä

Peleissä oleva tekoäly on nykyään hyvinkin pitkälle kehittynyttä ja suurimmassa osassa pelejä on jonkinlainen tekoäly. Pelien alkuaikoina tekoäly ei kuitenkaan ollut itsestäänselvyys. Ensimmäinen peli julkaistiin 1958 (oskilloskoopilla pelattava Tennis For Two) ja ensimmäinen tietokonepeli 1961 (Spacewar PDP-1 tietokoneelle). Vuonna 1968 julkaistu Space invaders–kolikkopeli astui jo hieman tekoälyn suuntaan vihollisten yksinkertaisilla liikkumisradoilla. Tämä ei kuitenkaan ole ”todellista” tekoälyä, sillä viholliset eivät reagoi mitenkään ympäristöönsä tai pelaajaan, vaan liikkuvat ennalta määrättyä liikerataa. Kuvassa 2 on mahdollisesti ensimmäinen ”todellinen” tekoäly löytyy 1972 vuonna tehdystä Pong-kolikkopelistä. (Wexler 2002.)



Kuva 2. Pong-peli (Kuva: Wikimedia Commons 2006).

Pong on yksinkertainen pöytätennistä simuloiva peli, jossa ruudulla on vain mailoja esittävät 2 palkkia, palloa esittävä neliö sekä pisteet. Peliä voidaan pelata joko kahdella pelaajalla tai tietokonevastusta vastaan. Tietokonevastuksen liike perustui yksinkertai-

seen laskutoimitukseen, joka laski missä kohtaa pallo ylittäisi mailan liikkumislinjan. Laskutoimituksen jälkeen maila siirtyi kyseiseen paikkaan. Jos tekoälyn annettaisiin pelata täydellisesti, olisi sen voittaminen käytännössä mahdotonta, joten siihen piti tehdä muutoksia, jotta sitä vastaan pelaaminen olisi mielekästä. Tämä saatiin aikaiseksi antamalla mailalle tietty liikkumisnopeus, jotta maila ei vain suoraan siirtyisi oikeaan paikkaan. Mailan nopeus voitiin myös sitoa eri vaikeustasoihin, jolloin tekoäly tarjoaa haastetta sekä peliä ensimmäistä kertaa pelaavalle kuin myös peliä paljon pelanneelle. Helpoimmalla vaikeustasolla mailalle annettiin myös tietty todennäköisyys liikkua väärään paikkaan lasketun oikean paikan sijasta. Tällä saatiin erehtymätön tietokone käyttäytymään ihmismäisemmin. (Wexler 2002.)

Tekoäly on kehittynyt tasaista tahtia pelien rinnalla. Jotkin pelit ovat kumminkin tunnettuja siitä, kuinka niiden tekoäly pystyi tekemään jotain aivan uutta, jota muista sen ajan peleistä ei vielä löytynyt. Esimerkiksi vuonna 1980 julkaistu Pac-Man oli ensimmäinen peli, jonka tekoälyvastustajat osasivat navigoida labyrintissa. Vastustajilla oli myös toisistaan eriävät persoonallisuudet jotka vaikuttivat niiden liikkumisratoihin (Mateas 2003.)

Ensimmäinen Sim City, joka julkaistiin vuonna 1989 oli ensimmäisiä pelejä, joissa pelaaja pääsi kontrolloimaan monimutkaista tekoälyn ohjaamaa simulaatiota. Kaupungin toiminta on mallinnettu hyvin tarkasti vastaamaan oikeaa kaupunkia. Se oli ominaisuus, jota tämän ajan peleissä ei oltu nähty aikaisemmin. (Champandard 2007.)

Vuonna 1996 julkaistu Creatures on peli, jossa pelaaja kasvattaa munista kuoriutuvia pieniä eläimiä. Mikä teki tästä pelistä uraa uurtavan, on se että pelaaja pystyy opettamaan näitä pieniä eläimiä. Voit esimerkiksi poimia kursorillasi hedelmän puusta ja näyttää sitä ja kirjoittaa sanan "hedelmä". Tällöin eläin ymmärtää, että hedelmä on hedelmä, ja tämä tieto välitetään kaikille muille elossa oleville elämille neuraaliverkon kautta. Eläimiä voi myös palkita tai rankaista, mikä vaikuttaa niiden toimintoihin. Tällainen opettamisen mahdollisuus oli peleissä tähän aikaan jotain aivan täysin uutta ja peliä pidetäänkin läpimurtona keinotekoisen elämän tutkinnassa. Ensimmäisen persoonan räiskintäpeleissä isoimmat tekoälyn uranuurtajat ovat vuonna 1998 julkaistu Half-Life, vuonna 2001 julkaistu Halo: Combat evolved sekä vuonna 2005 julkaistu F.E.A.R. (Champandard 2007.)

Half-Life oli ensimmäisiä pelejä, joissa "välianimaatiot" olivat täysin interaktiivisia. Pelaajan mukana kulkee myös tekoälyn ohjaama vartija joka avustaa pelaajaa pelin alkuvaiheilla. Pelin loppupuolella olevat viholliset käyttävät joukkotaktiikoita tehokkaasti. Halo vei vihollisten tekoälyä vielä pidemmälle ja pelissä viholliset osasivat käyttää suojia tehokkaasti hyödykseen sekä heittää kranaatteja takaisin. Täysin uutta tekoälyrintamalla oli myös joukkojen karkuun lähteminen niiden johtajan kuollessa. (Champandard 2007.)

F.E.A.R vei vihollisten tekoälyä vielä askeleen pitemmälle. Yksiköt osaavat käyttää joukkotaktiikoita kuten aikaisemmin mainitut pelitkin, mutta peli osaa generoida erilaisia taktiikoita ottamalla sen hetkisen ympäristön huomioon. Vihollisyksiköt osaavat myös käyttää ympäristöään vieläkin tehokkaammin hyödyksi kaatamalla pöytiä suojakseen sekä hyökkäämällä pelaajan kimppuun ikkunoista (Champandard 2007.) Edellä mainitut pelit olivat vain pieni otanta suuresta määrästä pelejä, jotka ovat jokainen vieneet pelien tekoälyä eteenpäin omalla tavallaan.

2.4 Tekoälyn tehtävistä peleissä

Tekoäly hoitaa peleissä useita eri tehtäviä usein sellaisiakin joita ei itse pelatessa tule ajatelleeksikaan. Jos pelissä on yksinpeli tai moninpeli, jossa on mahdollisuus pelata tietokonevastustajia vastaan, on tekoälyn tehtävänä kontrolloida tietokonepelaajan yksiköitä tai yksinpelin tietokoneohjattuja hahmoja. Tietokoneohjattujen hahmojen ja tekoälyvihollisten toiminta voidaan tiivistää ”äärellisen automaatin” malliin. Äärellinen automaatti voidaan ajatella kaaviona, jossa on x–määrä tiloja, joiden välillä siirrytään kun tietyt ehdot täyttyvät. Esimerkiksi tietokoneen ohjaaman hahmon on tarkoitus kävellä oviaukon läpi. Kyseisessä oviaukossa on kumminkin suljettu ovi eikä hahmo voi liikkua sen läpi. Tekoälyn pitää ensin avata ovi ja vasta sen jälkeen kulkea oviaukosta. Kuvan 3 kaavio kuvaa edellä mainitun automaatin toimintaa.

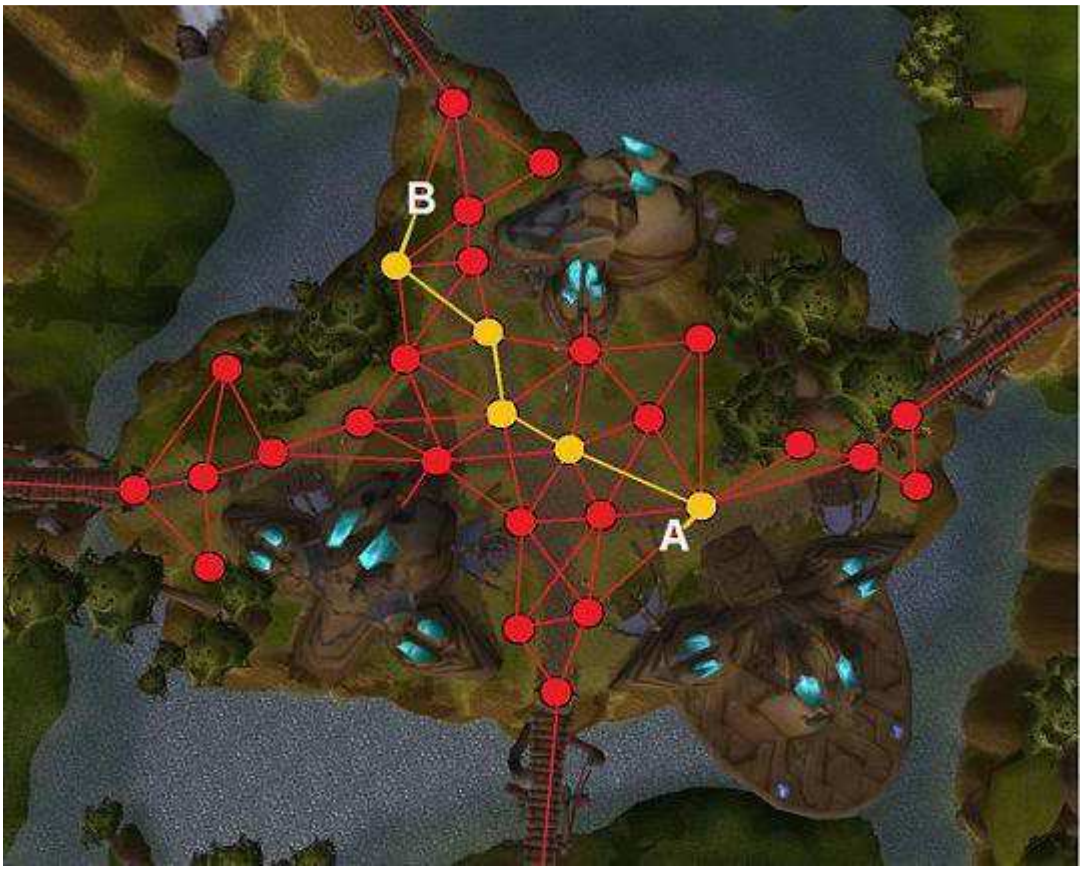
Kuva 3. Äärellisen automaatin toimintaa kuvastava tilasiirtymäkaavio.

Äsken mainittu malli on karkea malli tekoälyn toiminnan mallintamiseen, mutta se pitää lähes kaikissa tekoälyn toteutuksissa paikkansa. Tekoälyn toimintaan saatetaan lisätä muuttujia tai sattumanvaraisuuksia, joilla saadaan tekoäly toimimaan hieman eri tavalla tai sille saadaan määrättyä erilaisia ”persoonallisuuksia”. Näistä persoonallisuuksista voidaan antaa esimerkiksi strategiapelissä hyökkäävä tai puolustava vihollinen, armeijaan tai ekonomiaan keskittyvä vihollinen ja niin edelleen. Tekoälyn tärkeimpiä tehtäviä on myös reitinetsintä olettaen tietenkin, että kyseinen peli sellaista tarvitsee. Lähes kaikki pelit tarvitsevat reitinetsintää jossakin määrin. Reitinetsintää voidaan toteuttaa monella eri tavalla sen mukaan millä tavalla peli on toteutettu. Kuvassa 4 näemme World of Warcraft -pelin "Halaa" -nimisen kaupungin ja siihen on merkittynä lähtöpiste A sekä loppupiste B. Tekoälyn on tarkoitus kulkea pisteestä A pisteeseen B ja tämä liikkuminen voidaan toteuttaa muutamalla eri tavalla.

Kuva 4. Halaa -kaupunki (Kuva: Paul Tozorou 2008, käytetty tekijän luvalla).

Ensimmäinen vaihtoehto on toteuttaa tekoälyn liikkeet välietappisysteemillä. Pelin karttaan on siis sijoiteltu pelaajalle näkymättömiä pisteitä, joiden kautta tietokonepelaajat voivat kulkea. Tekoäly laskee siis lyhimmän reitin nykyisestä paikasta välietappien kautta paikkaan, johon tekoäly haluaa mennä. Tämä tapa reitinhakuun on suhteellisen kevyt, mutta aiheuttaa tietenkin sen, että tekoälyllä on vain tietty määrä pisteitä, joihin se voi liikkua. Tämän takia reitit eivät ole välttämättä aivan suoria ja liikkuminen saattaa olla siksakkaavaa (Tozorou 2008.)

Kuvassa 5 edellä mainittuun Halaa-kaupunkiin on sijoiteltu välietapit sekä reitti niiden kautta pisteestä A pisteeseen B. Reitti on suhteellisen suora mutta siinä on silti pieniä mutkia, joita ei oikeassa elämässä tehtäisi.



Kuva 5. Välietapit Halaa-kaupungissa (Kuva: Paul Tozorou 2008, käytetty tekijän luvalla).

Reitinhaussa 3D-maailmassa voidaan hoitaa myös huomattavasi järkevämmin ja tehokkaammin ”navigaatioverkon” avulla. Alueet, joilla tekoäly voi liikkua, peitetään yksinkertaiseen polygoniverkkoon ja tekoälyn liikkuessa sen tarvitsee vain tarkistaa, onko haettu reitti kyseisen verkon sisällä. Tällä tavalla vältetään välietappimallista johtuvaa pientä siksakkausliikettä, liikkumisesta tulee sulavampaa ja vaikka pelissä olisi esimerkiksi ympäriinsä lentäviä esineitä, osaisi tekoäly liikkumisverkon takia kiertää ne, niin kauan kuin siihen on tilaa (Tozorou 2008.)

Kuvassa 6 Halaa-kaupunkiin on asetettu navigointiverkko, joka kuvassa näkyy valkoisena alueena, jonka kulmissa on siniset ympyrät. Kaikki tämän alueen sisälle jäävä pelimaailma on siis tekoälylle mahdollista kulkureittiä. Kulkureitti on luotisuora koska koko alue A ja B pisteen välissä sattuu olemaan navigointiverkon sisäpuolella.

Kuva 6. Navigaatioverkko Halaa-kaupungissa (Kuva: Paul Tozorou 2008, käytetty tekijän luvalla).

2.5 Reitinetsintäalgoritmit

Reitinetsinnässä käytettävät algoritmit ovat todellisuudessa sovellettuja graafien läpikäyntiin tarkoitettuja algoritmeja. Niistä tunnetuin on Dijkstran algoritmi. Kehittäjänsä Edsger Dijkstran mukaan nimetty algoritmi etsii graafille lyhimmän reitin yhdestä pisteestä mihin tahansa muuhun graafissa olevaan pisteeseen. Pisteiden välillä olevien viivojen painojen pitää kumminkin olla ei-negatiivisia. (Black 2006.)

Dijikstran algoritmia on helpoin ajatella siten, että pisteet ovat kaupunkeja ja viivat ovat teitä niiden välillä. Jokaisella tiellä on myös painonsa eli pituus. Algoritmi ensiksi asettaa etäisyyden lähtökaupunkiin nollaksi (koska kaupungissa ollaan tällä hetkellä, joten sinne ei ole matkaa) ja kaikkien muiden kaupunkien etäisyydeksi äärettömyyden. Äärettömyys tämän algoritmin konseptissa tarkoittaa sitä, ettei kaupungissa ole ”käyty” vielä.

Tämän jälkeen aloituskaupunkista aletaan katsoa matkoja kaikkiin siihen suoraan yhteydessä oleviin kaupunkeihin. Algoritmi etsii kaupungin, joka on lyhimmän matkan päässä. Etsinnän aikana jokaiselle tarkastellulle kaupungille annetaan arvoksi sen etäisyys alkukaupungista. Kun lyhimmän matkan päässä ollut kaupunki on löytynyt, merkitään se tämänhetkiseksi kaupungiksi sekä käydyksi kaupungiksi. Tästä uudesta kaupungista aletaan sama lähimmän kaupungin etsintä. Matkat ovat summautuvia eli tämän tarkistuksen aikana tarkasteltujen kaupunkien arvoksi tulevat niiden etäisyys tämänhetkisestä kaupungista plus matka, joka on kuljettu että tämänhetkiseen kaupunkiin on päästy (Sniedovich 2012.)

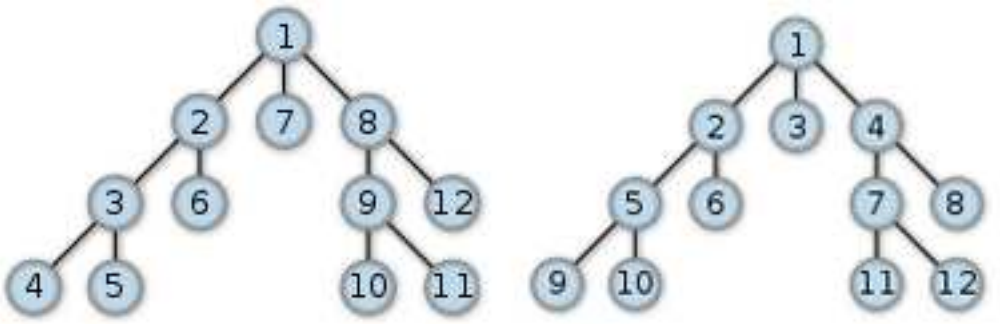
Uusi arvo annetaan kaupungille vain, jos se on pienempi kuin sille ennemmin annettu arvo. Tärkeätä on myös se, että jo käytyjä kaupunkeja ei tarkastella koskaan. Kaupunkiin johtavan matkan tarkastelu ei tee siitä käytyä, vaan kaupunki merkitään käydyksi vain, jos algoritmi on valinnut sen osaksi lyhyintä reittiä. Tätä silmukkaa jatketaan niin kauan kunnes päästään haluttuun kaupunkiin. Halutun kaupungin ympäriltä tarkastetaan vielä kaupungit, jos niistä jonkin antaisi vielä lyhyemmän reitin. Kun ympärillä olevat kaupungit on tarkastettu, valitaan kaikkein lyhyin reitti. Algoritmi keskeytyy myös, jos tämänhetkisen kaupungin ympärillä olevat kaupungit on kaikki merkitty käydyiksi, jolloin reittiä ei ole ollut mahdollista löytää (Sniedovich 2012.)

Vuosien varrella huomattiin että Dijkstran algoritmia olisi mahdollista optimoida ja yksi näistä optimoinneista on pelikäytössä varsinkin paljon käytössä oleva A* algoritmi. Kyseessä oleva algoritmi optimoi Dijkstran algoritmia lisäämällä siihen heureettisen arvioinnin matkasta, joka pitää kulkea tämänhetkisestä pisteestä maaliin. Heureettinen arvio on toteuttajasta riippuva matemaattisesti laskettu arvo joka arvio kahden eri pisteen väliä. Tällä tavalla algoritmin kulkua saadaan ohjattua paremmin suoraan kohti maalia eli se ei turhaan levitä etsintäänsä laajalle alalle vaan suuntaa maalia kohti tehokkaammin. (Paterl 2012.)

Liite 1 kuvaa A* reitinetsintäalgoritmia. Mainittu heureettinen arvio voitaisiin esimerkiksi laskea vähentämällä tämänhetkisen pisteen x- ja y-koordinaatit päätepisteen vastaavista koordinaateista ja laskemalla tulokset yhteen. Pienillä muutoksilla kyseinen pseudokoodi pätee myös dijikstran algoritmiin. Ainoa muutos on heureettisen arvion laskentaan, jonka tulos muutetaan olemaan aina 0. Tämä johtuu siitä, että dijikstran al-

goritmissa ei ole heureettista arviota ja muuttamalla sen tulos nollaan saadaan sen vaikutus poistettua.

Kaksi yksinkertaisempaa reitinetsintäalgoritmia ovat Depth-first search (DFS) sekä Breadth-first search(BFS) -algoritmit. Nämä kaksi algoritmia ovat huomattavasti yksinkertaisempia kuin aikaisemmin mainitut algoritmit. Ne eivät ota huomioon pisteiden välistä matkaa laisinkaan eivätkä ne ohjaa itseään lyhimpää reittiä kohti, vaan käyvät graafia läpi systemaattisesti. Ne eroavat toisistaan siinä, missä järjestyksessä ne käyvät kyseistä graafia läpi. DFS yrittää nimensä mukaisesti päästä mahdollisimman "syvälle" graafiin, ennen kuin se alkaa käydä läpi muita mahdollisia reittejä. BFS taas käy graafia läpi yksi "kerros" kerrallaan. Kuvassa 7 vasemmalla on DFS algoritmin läpikäyntijärjestystä kuvaava graafi, oikealla BFS algoritmin vastaava graafi.



Kuva 7. Depth-first search- ja Breadth-first search -algoritmien graafin läpikäyntijärjestykset (Alexander Drichel / Wikimedia Commons 2008).

2.6 Vuoropohjaisen strategiapelin tekoäly

Tekoälyn toiminta vaihtelee paljon pelityypin mukaan. Esimerkiksi toimintapeleissä tekoälyllä on ylivoimainen ihmispelaajaa vastaan, sillä tekoälypelaajan tähtäys on täydellistä sekä sen refleksit ovat monta kertaa ihmistä nopeammat. Suurin haaste näissä peleissä onkin tehdä tekoälyvastustajasta huonompi kun se voisi oikeasti olla, jotta ihmispelaajilla on mahdollisuus sitä vastaan. Vuoropohjaisessa strategiapelissä ihmispelaaja on etulyöntiasemassa. Ihmispelaajan suunnittelutaidot ja intuitio useimmiten voittavat tekoälyvastustajan tiettyihin sääntöihin perustuvan pelityylin.

Kun suunnitellaan tekoälyä vuoropohjaiseen strategiapeliin täytyy ottaa huomioon pelityypin hyvät ja huonot puolet tekoälyn kannalta. Vuoropohjaisessa pelissä juurikin vuorot ovat suuri hyöty. Pelaajan vuorolla tekoälyvastustajan ei tarvitse olla edes aktiivisena, joten sen toimintoihin ei tarvitse kuluttaa laskentatehoa laisinkaan. Tekoälypelaajan vuorolla taas kaikki laskentateho voidaan omistaa tekoälyn käyttöön ja tekoäly saa tehdä tarvittavat laskutoimitukset ja niistä seuraavat toimenpiteet rauhassa ja antaa vuoron jälleen pelaajalle.

Vuoropohjaisen strategiapelin tekoälyä tehdessä pitää ottaa huomioon myös se, ettei tekoäly tekisi asioita jokaisella pelikerralla samalla tavalla. Tällaista tekoälyvastustajaa vastaan pelaaminen käy nopeasti tylsäksi ja ennalta arvattavaksi. Tämän välttämiseksi tekoälyyn voidaan lisätä satunnaismuuttujia tai persoonallisuusmuuttujia jotka vaikuttavat tekoälyn toimintaan. Esimerkiksi jokin tekoälyvastustaja saattaa olla muita aggressiivisempi, jokin hieman passiivisempi jne.

Yksi tapa toteuttaa vuoropohjaisten strategiapelin tekoäly on eräänlainen prioriteettisysteemi. Eri toiminnoille pelissä asetetaan erilaisia prioriteettiarvoja ja tämän avulla tekoälyn ohjastamille yksiköille määritetään parhaiten sopiva tehtävä. Tehtävät vaihtelevat pelin idean mukaisesti mutta esimerkkinä voidaan antaa seuraavanlainen prioriteettilistaus:

1. Tukikohtien puolustus
2. Vihollisten tukikohtien kimppuun hyökkääminen
3. Uusien tukikohtien perustaminen
4. Vihollisten yksiköiden kimppuun hyökkääminen
5. Vahingoittuneiden yksiköiden korjaus
6. Uusien alueiden tutkiminen

Listassa tärkein tehtävä on ensimmäisenä, vähiten tärkein viimeisenä. Prioriteettilistausta voidaan käyttää pelin toteutuspuolella siten, että vuoron alussa kerätään tietoa pelin sen hetkisestä vaiheesta, lasketaan vaiheen perusteella tarpeelliset tehtävät ja määrätään niille parhaiten sopivat suorittajat. Yksikön sopivuutta määritellessä otetaan huomioon sen kunto sekä matka, jonka yksikön tarvitsisi matkustaa suorittaakseen tehtävän.

Tämän tyylinen järjestelmä on toteutuspuolella suhteellisen yksinkertainen, mutta sen tuottama tekoäly on kumminkin suhteellisen haastava sekä se reagoi hyvin muuttuviin tilanteisiin (Welch 2007.)

3 Case: Droidwars

Tässä luvussa käsitellään Droidwarsia, sen mekaniikoita, sen suhdetta tekoälyyn ja tekoälyn toteutukseen sekä käydään läpi sen toteutukseen käytetyt työkalut. Droidwars on yksin ohjelmoimani Advance Warsiin pohjautuva vuoropohjainen strategiapeli Android-alustalle.

3.1 Droidwarsin mekaniikat

Ennen kuin peliin lähdetään edes tekemään tekoälyä, on tärkeää olla selvillä pelin toiminnasta ja sen mekaniikoista. Jos peliin halutaan tekoäly, saatetaan sen mekaniikoiden suunnitteluvaiheessa ottaa huomioon tekoälyn tuomat rajoitukset ja vaatimukset.

Droidwars on strategiapeli, jossa päämääränä on vastustajan voittaminen niin kuin muissa monissakin strategiapeleissä. Pelin voi voittaa kahdella eri tavalla, joko valloittamalla vastustajan päämajan tai saamalla vastustajan luovuttamaan. Luovuttaakseen pelaajan pitää valita vuorollaan pelin valikosta ”luovuta” valinta, jolloin peli varmistaa, että pelaaja haluaa luovuttaa ja näyttää tämän jälkeen loppuruudun.

Droidwarsin keskeisin pelimekaniikka on rakennukset. Pelissä on neljänlaisia rakennuksia, joista jokainen palvelee omaa tehtäväänsä. Molemmilla pelaajilla on päämaja, jonka menettäminen tarkoittaa pelin häviämistä. Kartalla on myös kaupunkeja, joita pelaajat voivat käyttää yksikköjensä korjaamiseen. Kaupungit tuottavat myös rahaa omistajalleen joka vuoron alussa. Kaksi viimeistä rakennusta ovat tuotantorakennuksia, joita pelaajat käyttävät yksikköjen tekemiseen. Tehtaasta tuotetaan kaikki pelin maayksiköt ja lentokentältä kaikki lentävät yksiköt. Kaikki rakennukset päämajoja lukuun ottamatta voivat olla joko jommankumman joukkueen hallussa tai neutraaleja. Neutraalit kaupun-

git eivät tuota rahaa kummallekaan pelaajalle ja neutraalit tuotantorakennukset eivät ole laisinkaan käytettävissä. Rakennuksia vallataan liikuttamalla niiden päälle jalkaväkiyksikkö ja valitsemalla ”valloita”. Rakennuksilla on 200 valloituspistettä ja tästä luvusta vähennetään valloittavan yksikön terveyspistemäärä (1-100). Kun valloituspisteet menevät nollaan tai sen ali, on rakennus valloitettu.

Yksiköiden valmistukseen tarvitaan Droidwarsissa rahaa, jota saadaan pelaajan valloittamista kaupungeista. Pelin alkaessa pelaajalla on 10 000 rahaa ja pelaaja saa 1000 rahaa jokaisesta valloittamastaan kaupungista joka vuoron alussa. Rahaa käytetään sekä yksiköiden valmistamiseen tehtaissa ja lentokentillä sekä niiden korjaamiseen kaupungeissa. Korjaus tapahtuu aina pelaajan vuoron alussa ja se maksaa yhden kymmenesosan yksikön hinnasta kymmentä terveyspistettä kohti. Yksikköä voidaan korjata kerrallaan maksimissaan 20 terveyspistettä.

Iso osa Droidwarsia on myös vastakkaisten puolien yksiköiden taistelu. Yksiköiden hyökkäystapoja on kahdenlaisia. Kahta yksikköä lukuun ottamatta yksiköiden pitää olla vastustajan yksikön vieressä hyökätäkseen. Tähän sääntöön poikkeuksen tekevät tykistö ja ilmatorjuntaohjusyksiköt, joiden hyökkäys on epäsuora. Ne voivat ampua useamman ruudun päähän, jolloin hyökättävä yksikkö ei voi ampua takaisin, jollei se satu olemaan myös epäsuorasti hyökkäävä yksikkö. Yksiköillä saattaa olla myös rajoitteita yksiköiden suhteen, joita vastaan ne voivat hyökätä. Tykistöyksiköt voivat hyökätä vain muita maakohteista vastaan ja ilmatorjuntaohjusyksiköt voivat hyökätä vain ilmakohteita vastaan. Miehistönkuljetusvaunu ei taas voi hyökätä laisinkaan. Yksiköt myös tekevät eri määriä vahinkoa eri yksiköihin. Konekiväärillä varustettu tiedustelujeeppi ei esimerkiksi tee kovinkaan paljon vahinkoa tankkiin, mutta on hyvinkin tehokas jalkaväkeä vastaan. Yksikön hyökätessä sen aiheuttamaan vahinkoon vaikuttaa hyökkäävän yksikön tyyppi ja terveyspisteet, puolustavan yksikön tyyppi sekä puolustavan yksikön ympäristön antava suoja. Kun aiheutetun vahingon määrä on laskettu, se vähennetään puolustavan yksikön terveyspisteistä. Jos puolustavan yksikön terveyspisteet tipahtavat nollaan tai sen ali, yksikkö tuhoutuu. Jos puolustava yksikkö ei tuhoudu ja se voi hyökätä takaisin, tehdään sama vahingonlaskenta mutta yksiköiden paikat vaihtuvat. Tällä kertaa hyökkäävän yksikön terveyspisteet ovat toki huomattavasti alemmat joten ensin hyökkäävä yksikkö saa taisteluissa yleensä edun.

Droidwarsin maasto koostuu kolmesta erilaisesta maastotyypistä rakennuksien lisäksi. Perusmaasto on ruohikkoa, joka tarjoaa todella vähän suojaa. Seuraavana on metsikkö, joka hidastaa ajoneuvoja, jotka ajavat sen läpi, mutta se tarjoaa hieman parempaa suojaa kuin ruohikko. Viimeisenä maastotyyppinä on vuoristo, jota ajoneuvot eivät voi ylittää laisinkaan, mutta se tarjoaa pelin parhaan suojan. Lentävät yksiköt eivät hyödy mitenkään maastosta, jonka yllä ne ovat. Rakennukset tarjoavat myös suojaa. Tehtaat, lentokentät sekä kaupungit tarjoavat hieman metsää parempaa suojaa, ja päämaja tarjoaa yhtä hyvän suojan kun vuoret.

Droidwarsissa yksiköt voivat liikkua saman puolen yksiköiden läpi mutteivät vihollispuolen yksiköiden läpi. Yhdessä ruudussa voi olla vain yksi yksikkö, joten samaan ruutuun ei voi kasata useampaa yksikköä. Jokainen yksikkö voi liikkua kerran vuorossa. Kun yksikkö on liikkunut käskettyyn paikkaan, voi sille valita erilaisia toimintoja. Yksikkö voi joko

1. Hyökätä vihollisyksikön kimppuun.
2. Valloittaa rakennuksen (vain jalkaväkiyksiköt).
3. Odottaa.
4. Lastautua kuljetusajoneuvoon (vain jalkaväkiyksiköt).
5. Jättää yksikön kyydistä (vain kuljetusajoneuvot).

Kaikki yksiköt eivät voi hyökätä liikuttuaan, sillä sekä tykistö että ilmatorjuntaohjusyksiköt voivat vain joko hyökätä tai liikkua. Jalkaväkiyksiköt voivat päättää liikkumisensa kuljetusyksikköjen päälle, jos niissä ei ole jo toista jalkaväkiyksikköä kuljetettavana ja voivat lastautua kyytiin. Kun jalkaväkiyksikkö lastautuu kuljetusajoneuvoon, ei kuljetusajoneuvo voi enää liikkua samalla vuorolla.

3.2 Droidwars ja tekoäly

Droidwars on Androidille tehty ruutukaavalla toimiva vuoropohjainen strategiapeli. Sekä pelin alusta että pelin tyyppi asettaa tekoälylle tiettyjä vaatimuksia sekä rajoitteita. Android-alusta tarkoittaa sitä, että peli tulee olemaan mobiilipeli. Tämä tarkoittaa että laskentatehoa ei ole käyttävissä saman verran kun esimerkiksi tietokoneella. Vuoropoh-

jaisuus kumminkin auttaa tähän ongelmaan. Pelaaja saa tehdä omat siirtonsa ensin ja tekoäly saa tehdä siirtonsa omalla vuorollaan, joten viety laskentateho ei vaikuta pelaajan pelikokemukseen, jollei se ala vaikuttamaan pelin sulavaan pyörimiseen.

3.2.1 Tekoälyn tehtävät Droidwarsissa

Droidwars käyttää tekoälyä pääasiassa kahdella eri tavalla. Ensimmäinen tapa on pelin käyttämät A* -reitinetsintä, jota kaikki pelin yksiköt käyttävät navigointiin ruutukaavalla. Kyseinen reitinetsintä on käytetyn Andenginen ominaisuus, joten pelin tekijän ei tarvinnut käyttää aikaa logiikan kirjoittamiseen. Reitinetsintään piti tehdä kumminkin muokkauksia, kun yksiköille määriteltiin yksikkötyyppikohtaiset liikkumissäännöt. Esimerkiksi ajoneuvot eivät voi kiivetä vuorille ja liikkuvat normaalia hitaammin metsässä. Jalkaväki taas voi kiivetä vuorille mutta se vie huomattavasti liikkumista. Lentokoneet ja helikopterit taas eivät välitä maastosta lainkaan. Reitinetsintää käytetään myös yksikön liikkumismahdollisuuksien näyttämiseen pelaajalle.

Toteutin tämän käymällä läpi kaikki yksikön ympärillä olevat ruudut, joihin kyseinen yksikkö voisi liikkua ja osoittamalla reitinetsinnän etsimään reitin kyseiseen ruutuun. Jos ruutuun liikkuminen oli mahdollista, lisättiin kyseiseen kohtaan mahdollista liikkumista osoittava merkki ja jatkettiin seuraavaan teoreettisesti mahdolliseen ruutuun. Kun kaikki teoreettisesti mahdolliset ruudut oli käyty läpi, näytettiin kaikki liikkumisvaihtoehdot pelaajalle. Myös yksiköiden hyökkäämisalueen näyttäminen tehtiin kyseistä tekniikkaa käyttäen. Kävin samalla tavalla läpi kaikki mahdolliset liikkumisruudut ja katsoin jokaisen ruudun kohdalla yksikköä ympäröivät ruudut. Jokainen näistä ruuduista merkittiin mahdolliseksi ruuduksi hyökkäystä varten.

Toinen tekoälyn tehtävä Droidwarsissa on tekoälyn ohjaaman vastustajan päätöksenteko. Tämän tehtävän vaatimuksiin pureudutaan tarkemmin seuraavassa kappaleessa.

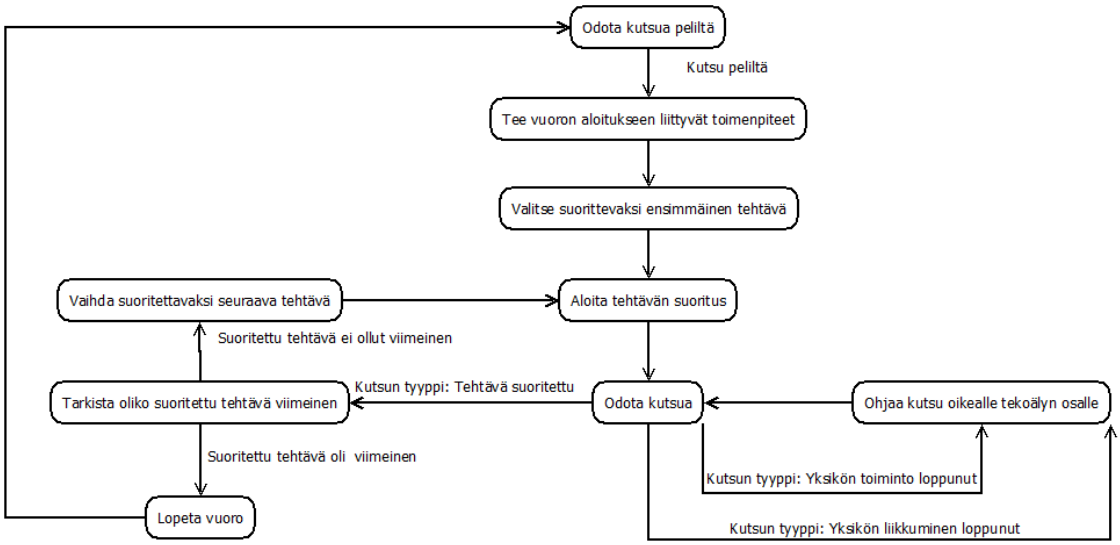
3.2.2 Droidwarsin tekoälyvastustaja

Droidwarsissa on mahdollista pelata tekoälyn ohjaamaa vastustajaa vastaan. Tämän vastustajan päätöksenteko perustuu prioriteettisysteemiin jossa eri tehtäville on asetettu eri prioriteetteja. Suurimalla prioriteetilla on toki oman päämajan puolustaminen ja alimpana on kartan tutkiminen. Prioriteettilistaa käydään läpi kohta kerrallaan ja sitä suorittamaan määrätään yksiköt jotka täyttävät kyseisen prioriteetin vaatimukset. Prioriteettilista koostuu seuraavista yksiköistä:

1. Tukikohdan suojelu
2. Rakennusten valloitus (vain jalkaväkiyksiköt)
3. Lähellä olevien vastustajien yksiköiden eliminointi
4. Suojan etsiminen
5. Kartan tutkiminen/vihollisen tukikohtaa päin liikkuminen
6. Uusien yksiköiden tuottaminen (tuotantorakennukset).

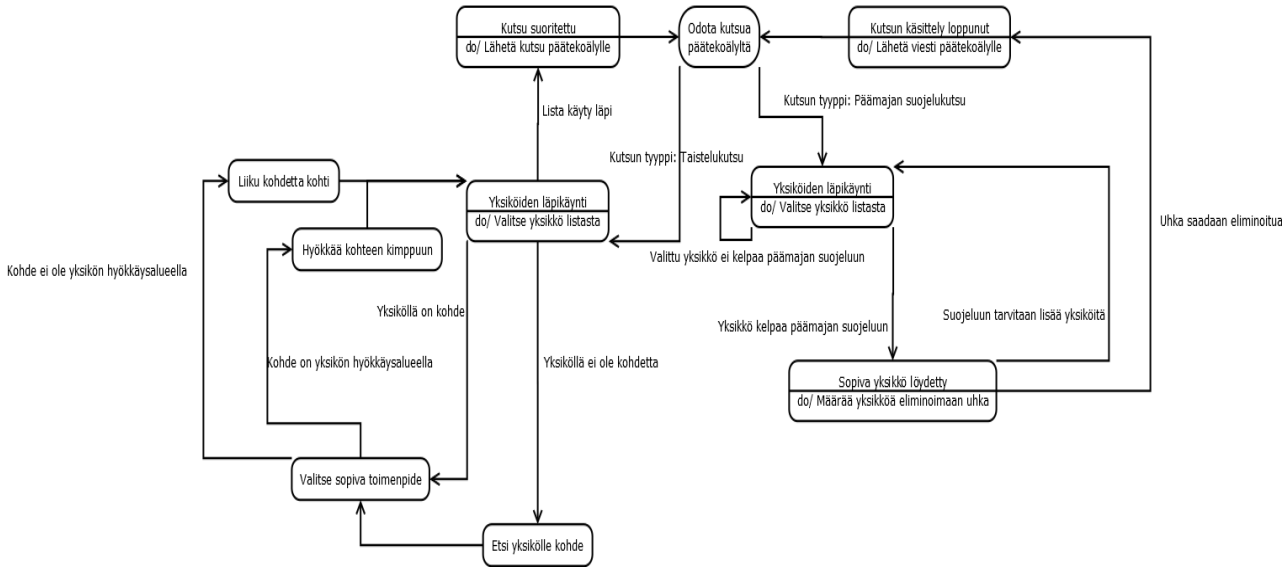
Tekoäly on sekä teoria että toteutuspuolella jaettu useampaan eri kategoriaan. Kyseiset kategoriat ovat pää-, taistelu-, valloitus-, liikkumis-, prioriteetti-, tuotanto- sekä yksikkötekoäly. Näistä jokaisella on oma tehtävänsä ja ne läpikäydään alla.

Päätekoäly, jonka toimintaa kuvastetaan kuvassa 8, kontrolloi muiden tekoälyosien suoritusjärjestystä sekä toimii kommunikointiväylänä tekoälyn ja muun pelin välillä. Esimerkiksi kun yksikkö lopettaa liikkumisensa, peli ilmoittaa siitä ensin päätekoälylle ja päätekoäly ohjaa kyseisen viestin oikealle tekoälyn osalle, joka päättää mitä seuraavaksi pitää tehdä. Päätekoäly sisältää myös listan kaikista tekoälyn yksiköistä sekä niille annetuista tehtävistä.



Kuva 8. Päätekoälyn tilasiirtymäkaavio.

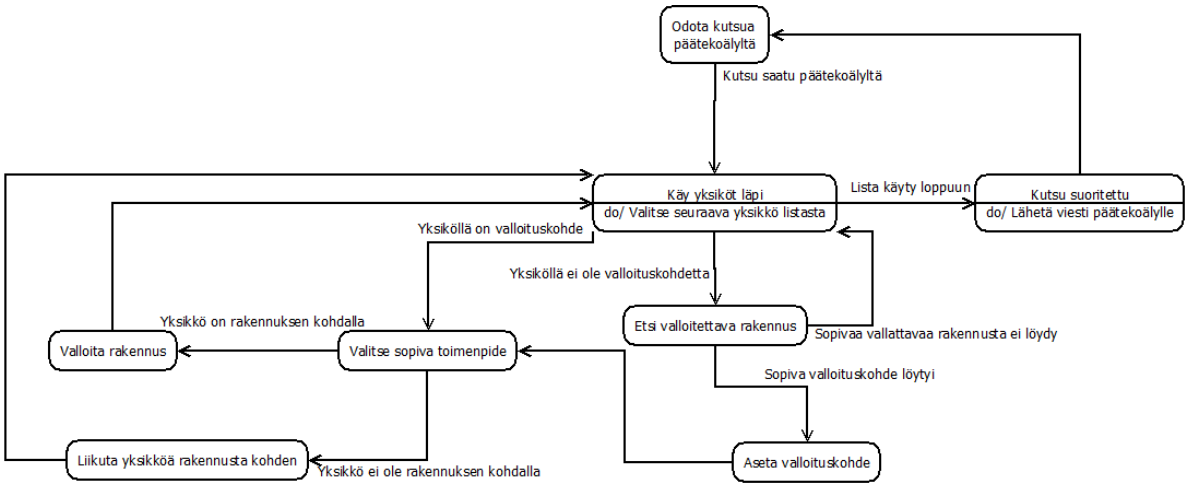
Kuvassa 9 kuvattu taistelutekoäly katsoo, onko yksikön lähellä vihollisen yksiköitä. Jos on, tekoäly ennustaa taistelun tuloksen ja jos ennustus on suotuisa, hyökkää vihollisyksikön kimppuun. Taistelutekoäly sisältää tunnistelistan yksiköistä, jotka on määrätty taistelemaan. Tunnistelistan lisäksi tekoäly sisältää myös listan jokaiselle yksikölle määrätyistä kohteista. Taistelutekoäly hoitaa tarvittaessa myös päämajan suojelun.



Kuva 9. Taistelutekoälyn tilasiirtymäkaavio.

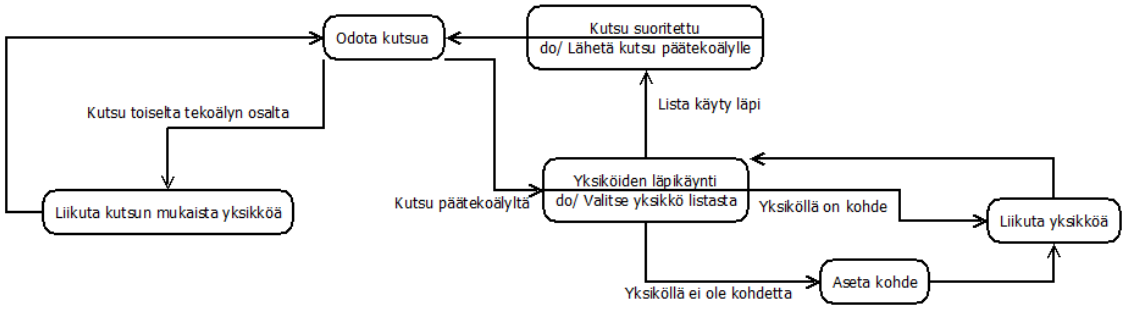
Valloitustekoäly toimii vain jalkaväkiyksiköillä ja se tarkistaa, onko kartalla valloitettavia rakennuksia. Jos on, se ohjeistaa jalkaväkiyksikköä liikkumaan niitä kohti tai valloittamaan niitä. Tuotantorakennuksilla on suurempi prioriteetti kuin kaupungeilla, mut-

ta kaikkein korkein prioriteetti on vastustajan päämajalla. Yksiköt eivät kumminkaan lähde heti vastustajan päämajaa kohti vain välimatka otetaan huomioon sekä valloitettavaa rakennusta ympäröivät vihollisen yksiköt. Valloitustekoäly sisältää taistelutekoälyn lailla tunnistelistan valloitustehtäviin määrätyistä yksiköistä sekä toisen listan yksiköiden valloituskohteista. Kohdelistaa käytetään uuden yksikön valloituskohdetta valittaessa, jottei kahdella yksiköllä ole samaa valloituskohdetta. Valloitustekoälyn toimintaa havainnollistaa kuva 10.



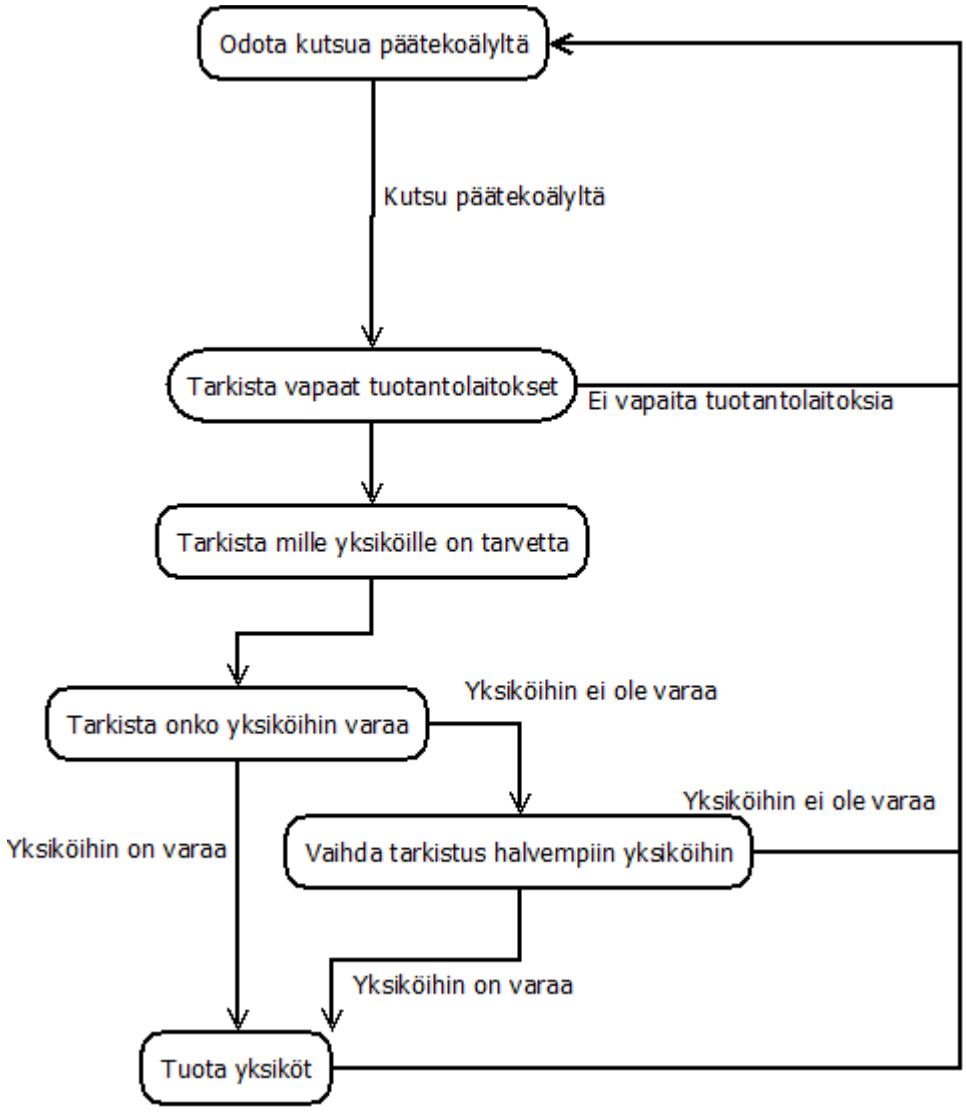
Kuva 10. Valloitustekoälyn tilasiirtymäkaavio.

Liikkumistekoäly (kuva 11) huolehtii tekoälyvastustajan yksiköiden liikuttelusta. Tekoäly tutkii mihin suuntaan yksikköä kannattaa liikuttaa ottaen huomioon maastot, jota yksikkö ei voi ylittää, vihollisen paikat jne. Tämäkin tekoäly sisältää listan pelkkään liikkumistehtävään määrätyistä yksiköistä sekä listan näiden yksiköiden kohdepaikoista. Liikkumistekoälyn kautta hoidetaan myös muiden tekoälyjen liikkumiskäskyt. Mahdollisia liikkumiskäskyjä on kahdenlaisia, on yksinkertainen liikkumiskäsky sekä ”liiku kohti kohdetta” -käsky. Ensimmäinen liikkumiskäsky olettaa, että annettu kohdepiste liikkumiselle on yksikön liikkumisalueen sisäpuolella. Liiku kohti kohdetta -käskylle voidaan antaa mikä tahansa piste kartalta ja käsky liikuttaa yksikköä sen maksimiliikkumismäärän kyseistä kohdetta kohti.



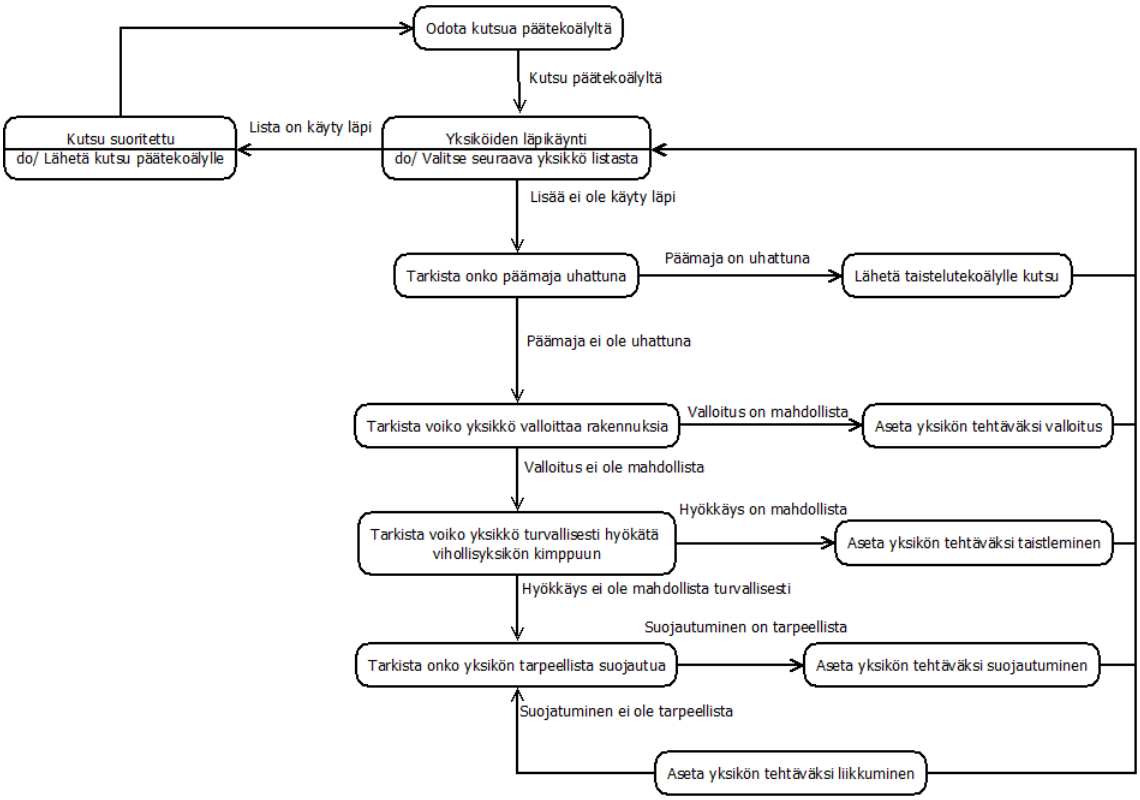
Kuva 11. Liikkumistekoälyn tilasiirtymäkaavio.

Tuotantotekoäly hoitaa yksiköiden tuottamisen tekoälyvastustajan omistamista tuotantorakennuksista. Tuotettavan yksikön tyyppi riippuu tarpeista, esimeriksi, jos tekoälypelaajalla ei ole yhtäkään jalkaväkiyksikköä, kannattaa niitä tuottaa, sillä ilman niitä rakennusten valloitus on mahdotonta. Tekoäly huomioi myös, mitä yksiköitä vastustajalla on. Jos vastustajalla on paljon ilmayksiköitä, kannattaa tekoälyn valmistaa yksiköitä jotka ovat vahvoja ilmayksiköitä vastaan. Tekoälyn täytyy myös ottaa huomioon tekoälypelaajan varat ja käyttää varat sen mukaisesti. Tuotantotekoälyn toimintaa kuvaa kaavio kuvassa 12.



Kuva 12. Tuotantotekoälyn tilasiirtymäkaavio.

Prioriteettitekoäly huolehtii prioriteettilistan läpikäynnistä tekoälypelaajan vuoron alussa. Jos yksikölle ei ole vielä määrätty tehtävää tai se on suorittanut tehtävänsä viime vuoron aikana, prioriteettitekoäly käy prioriteettilistan läpi ja määrää yksiköille niille sopivat tehtävät. Prioriteettitekoälyn läpikäyntijärjestys on koottu kuvaan 13.



Kuva 13. Prioriteettitekoälyn tilasiirtymäkaavio.

Tekoäly kokonaisuudessaan toimii prioriteettilistaa mukailevalla tavalla. Pelin lähettäessä päätekoälylle viestin tekoälypelaajan vuoron alkamisesta prioriteettitekoäly käy ensimmäisenä läpi tekoälyn yksiköt ja määrää tehtävättömille yksiköille niille sopivat tehtävät. Tämän jälkeen päätekoäly aloittaa prioriteettilistan tapaisen tehtävälistan läpikäymisen. Ensimmäisenä kutsutaan päämajan suojeluun tehtyä tekoälyä, joka jakelee päämajan suojeluun määrätyille yksiköille tarkemmat määräykset ja suorittaa ne. Kun suojelutekoäly on suorittanut kaikki tehtävänsä, se kutsuu päätekoälyn ”suorita seuraava” -funktiota, joka siirtää suoritusvaiheeseen seuraavan tehtävän joka tässä tapauksessa on hyökkääminen. Päätekoäly käskee siis hyökkäystekoälyä aloittamaan tehtävänsä, joka suorittamisen jälkeen kutsuu jälleen päätekoälyn ”suorita seuraava” -funktiota. Tätä silmukkaa suoritetaan, kunnes päästään viimeiseen tehtävään. Viimeisen tehtävän suorituksen jälkeen tekoäly päättää vuoronsa. Liitteessä 2 on pseudokoodiversio tekoälyn toiminnasta.

Kun tekoäly luo tuotantorakennuksista uusia yksiköitä, kutsutaan päätekoälyn yksikönlisäysfunktiota, joka lisää tuotetun yksikön tekoälyn kontrolloimien yksiköiden listaan. Yksikköjen tehtävälistaan lisätään myös tyhjä kohta, jotta tekoäly tietää seuraavan vuoron alussa määrätä yksikölle tehtävän. Kun prioriteettilista määrää yksikölle tehtävän, se

lisätään kyseistä tehtävää hoitavan tekoälyn viitelistalle. Kun yksikkö suorittaa tehtävänsä, se poistetaan viitelistalta ja sen tehtävä päätekoälyssä määritellään jälleen tyhjäksi ja sama silmukka alkaa jälleen alusta. Jos yksikkö tuhoutuu, peli kutsuu päätekoälyn "poista yksikkö"-funktiota, joka poistaa yksikön ensin kyseisen yksikön tehtävän viitelistasta ja sitten päätekoälyn yksikkölistasta sekä yksikön tehtävän yksikkötehtävälistasta.

3.3 Pelimoottorin valinta

Pelinkehityksen ensimmäinen vaihe on pelin ideointi. Kun alkuvaiheen ideointi saadaan valmiiksi, pitäisi selvillä olla minkälaisia ominaisuuksia peliin tulee. Tämän jälkeen voidaan alkaa etsiä idealle sopivia pelimoottoreita. Ohjelmoija voi toki kirjoittaa kokonaan oman pelimoottorinsa, mutta moottorin ohjelmointi vaatii jo todella suurta tietotaitoa ja kuluttaa paljon aikaa jo ennen kuin itse pelin ohjelmointiin päästään kiinni. Tarjolla on kumminkin useita erilaisia valmiita pelimoottoreita. Näistä pelimoottoreista osa on monelle alustalle soveltuvia, osa taas on erikoistunut vain yhdelle alustalle. Oma pelini oli suunniteltu alusta alkaen Androidille, jonka takia minun piti etsiä kyseisellä alustalla toimivia pelimoottoreita. Pelimoottorin valintaan vaikuttavat alustan lisäksi käytettävä ohjelmointikieli, mitä ominaisuuksia pelimoottorissa on ja mitä ominaisuuksia tehtävä peli tarvitsee. Jos pelin tekijä on tehnyt pelejä aikaisemmin, saattaa moottorin valintaan vaikuttaa myös aikaisempi kokemus tiettyjen moottoreiden kanssa.

Moottorin valintaan vaikuttavat kriteerit tässä tapauksessa olivat:
- Kehittäminen Android-alustalle pitää olla mahdollista
- Kielenä mielellään joko Java tai C#
- Moottorin täytyy olla ilmainen
- Mielellään hyvin dokumentoitu

Sopivia moottoreita etsiessäni tutustuin useampaan erilaiseen pelimoottoriin ja otin niistä tarkempaan tarkasteluun muutaman.

3.3.1 Unity3D

Unity3D on Unity Technologiesin kehittämä 3D-pelimoottori. Moottorin mukana tulee myös Unityn oma editori, jonka kautta pelin rakentaminen on helppoa. Editori sisältää työkalut pelin ympäristöjen ja asioiden helppoon rakentamiseen sekä niiden testaamiseen heti. Taulukosta 1 selviää, editori osaa myös käyttää muilla ohjelmilla tehtyjä 3D-malleja, tekstuureita ja ääniä suoraan ilman sen erikoisempaa säätämistä. Mukana tulee myös kaksi erilaista ohjelmointiympäristöä: Unityn editorin itsensä sisällä oleva suppea tekstieditori ja erillisenä ohjelmana oleva MonoDevelop, joka integroituu kätevästi Unityn editorin kanssa. Unityllä voi tehdä pelejä PC:lle, MAC:ille, Linuxille, selaimiin, uusimille konsoleille sekä mobiilialustoille. Mobiilialustoille ja konsoleille kehitettäessä ohjelmaan pitää ostaa erillinen lisenssi jokaista alustaa varten. Moottori on ensisijaisesti suunniteltu 3D-pelejä varten, mutta sillä voi kehittää helposti myös 2D-pelejä.

Taulukko 1. Unity3D:n hyvät ja huonot puolet.

+ Hyvä dokumentaatio	- Tarvitsee lisenssin Androidia varten
+ Helppokäyttöinen editori	- Useita pelilleni turhia ominaisuuksia
+ Mahdolliset kielet: JavaScript, C#, Boo	
+ Tukee tiedostoja muista ohjelmista (mm. Photoshop)	
+ Helppo asentaa ja ottaa käyttöön	

3.3.2 AndEngine

AndEngine on Nicolas Gramlichin kehittämä ilmainen vapaan lähdekoodin pelimoottori. Andengine lisätään Eclipseen kirjastoprojektina ja moottoria käytetään tätä kautta. Moottorin ominaisuuksiin kuuluu muun muassa tuki TMX-kartoille (Taulukko 2). Moottoriin on olemassa ilmaisena myös lisäpaketteja, jotka lisäävät tuen muun muassa Box2D -ominaisuuksille, moninpelille sekä monikosketukselle. Moottorin suurimpia heikkouksia on sen dokumentoinnin puute.

Taulukko 2. AndEnginen hyvät ja huonot puolet.

+ Ilmainen	- Ei kunnolista dokumentaatiota
+ Valmis A* reitinetsintä	- Ei erillistä editoria
+ Tuki TMX-kartoille	- Hankala asentaa
+ Ohjelmointikielenä Java	

3.3.3 Itse tehty pelimoottori

Vaihtoehtona pelin moottoriksi oli myös itse tehty pelimoottori. Pelimoottorin teko vaatii todella hyvän tuntemuksen alustasta, jolle moottoria tekee. Moottorin teko nostaa projektiin kulutettua aikaa todella paljon ja voi ehkä jopa viedä enemmän aikaa kun itse pelin teko (Taulukko 3). Hyvänä puolena oman moottorin tekemisessä on se että moottorin saa toimimaan juuri niin kuin haluaa ja siihen voi lisätä juuri halutut ominaisuudet

Taulukko 3. Oma pelimoottorin hyvät ja huonot puolet.

+ Moottori juuri sellainen kun haluat	- Vaatii vahvan alustan tuntemuksen
	- Vaatii paljon aikaa
	- Vaatii paljon taitoa

3.3.4 Pelimoottorin Valinta

Taulukossa 4 tarkastellaan vaadittuja ominaisuuksia ja niiden löytymistä eri vaihtoehdoista. Oman moottorin kohdalla ominaisuudet/dokumentointi riippuvat täysin moottorin kehittäjästä. Unity 3D:n kohdalla merkki on suluissa sen takia, että unity itsessään on ilmainen, mutta Android lisenssi on maksullinen. Android-lisenssistä on kumminkin olemassa ilmainen kuukauden testiversio, mutta se ei olisi riittänyt tarpeisiini.

Taulukko 4. Pelimoottoreiden ominaisuudet.

	AndEngine	Unity 3D	Oma moottori
Kehitys Androidille	X	X	X
Kielenä C# tai Java	X	X	X
Ilmainen	X	(X)	X
Hyvä dokumentointi		X	X

Moottorin valintaa mietin aika pitkään, mutta päädyin lopulta AndEngineen. Ensimmäisenä vaihtoehdoista poistin oman moottorin tekemisen. Moottorin tekeminen olisi nostanut projektin vaatiman ajan kaksinkertaiseksi enkä usko että minulla olisi riittänyt Android -alustan tuntemus tai koodaustaito moottorin tekemiseen. Jäljelle jäivät siis Unity ja AndEngine. Unity on minulle entuudestaan tuttu moottori ja tiedän sen helppokäyttöiseksi ja hyväksi moottoriksi. Unityn dokumentointi on myös todella hyvä, joten sille kehittäminen on mukavaa. AndEnginen ominaisuudet taas sopivat paremmin minun tarpeisiini.

Valinta vaati kumminkin miettimistä, sillä AndEnginen asennus on Unitya huomattavasti hankalampi ja siinä ei dokumentaatiota oikeastaan ollenkaan. Unity olisi myös vaatinut erillisen lisenssin Androidia varten, mutta noin kuukausi moottorin valinnan jälkeen sain kyseisen lisenssin. Vaihto oli siinä vaiheessa liian myöhäistä. Päädyin siis valitsemaan peliini moottoriksi AndEnginen sen huonoista puolista huolimatta. Moottorin käytön luulisi kumminkin opettavan itseäni kommentoimaan omaa koodia sekä opettavan koodin takaisinmallinnusta eli moottorin koodien kaivelua sen toiminnan selvittämiseksi. Moottoreiden erot on koottu taulukkoon 4.

3.4 Käytetyt ohjelmat

Pelin tekoon tarvitaan toki muitakin ohjelmia tai työkaluja kuin itse pelimoottori. Kun kehitetään mobiilipeliä, kasvaa työkalujen tarve entisestään. Valitsemani pelimoottori oli AndEngine, jolla ei ole omaa editoria vaan se lisätään kirjastona Eclipsen ohjelmointiympäristöön. Eclipse on mielestäni paras kehitysympäristö Androidille sille saatavan ADT-pluginin takia. Tämä plugin sisällyttää koko Android SDK:n toiminnallisuuden Eclipseen, joten Android-layout rakentaminen, virtuaalilaitteiden hallinta sekä laitteille kääntäminen onnistuu helposti suoraan Eclipsen kautta.

Eclipse ei ole kumminkaan ainut käyttämäni kehitysympäristö. Käytin palvelinpuolen PHP –scriptien tekemiseen ja palvelimelle siirtämiseen NetBeansia. Eclipseen ei ole kunnollista PHP-tukea ja NetBeansista PHP-tiedostojen siirtämisen palvelimelle saa helposti automatisoitua.

Siirryttäessä kehitysympäristöjen ulkopuolelle jää vielä muutama ohjelma, joita käytän pelin kehityksessä. Andenginen tukemien .TMX-karttojen rakentamiseen käytän Tiled-nimistä ohjelmaa. Kyseisessä ohjelmassa määrittelen käyttämäsi kartan palaset kuvasta, ja sen jälkeen voit maalata niitä ruutukaavaan ihan miten haluaa. Voit asettaa ruuduille erilaisia arvoja ja tehdä useamman kerroksen ruutuja jos niin haluat. Lopussa vain tallennat kartan .TMX –tiedostona, joka siis oikeasti vain XML-tiedosto. Kartta tarvitsee toimiakseen kyseisen TMX-tiedoston, joka määrittelee käyttämäsi kartan palaset sekä kuvatiedoston joka sisältää kyseiset palaset. Useampi kartta voi käyttää samoja kartanpalasia, joten karttojen koot jäävät tämän takia hyvin pieniksi.

Versionhallintaan käytän Mercurialia ja käyttämäni ohjelma on TortoiseHg. Valitsin käyttöön erillisen ohjelman, koska Eclipsen Mercurial plugin oli hieman vaikeakäyttöinen ja Eclipsen päivityttyä uuteen versioon se alkoi käyttäytyä hyvin epävakaasti. Kuvankäsittelyyn käytän GIMPiä sen ilmaisuuden takia. Kuvankäsittelyyn ei ole suurta tarvetta ja se on lähinnä vain korvikegrafiikoiden tekemistä varten.

3.5 Toteutus

Sopivan moottorin ja työkalujen valinnan jälkeen on aika siirtyä itse pelin toteutukseen. Toteutin pelin alusta alkaen itse mutta Advance Warsia mukaillen. Pelin toteutus aloitettiin tekemällä aloitusvalikot, jonka ensimmäisessä versiossa oli mahdollista vain aloittaa uusi peli tai sulkea peli kokonaan. Pelin alkuvaiheessa keskityttiin vain ja ainoastaan mekaniikoiden tekemiseen, sillä tekoälyä on hankala rakentaa peliin, jos mekaniikat eivät ole kunnossa. Jos tekoälyä lähtee toteuttamaan ennen kuin mekaniikat ovat täysin valmiit, saattaa tekoälyn toimintaan joutua tekemään suuriakin muutoksia myöhemmässä vaiheessa sen kehitystä. Kaikki tässä luvussa mainittavat havainnot ovat blogistani "Pelinkehitystä Susirajalla".

Ensimmäinen kehitysversio on kirjattu päivälle 8.2.2012. Se sisälsi vain yhden yksikön, jota pystyi liikuttelemaan Tiledillä tehdyllä kartalla. Yksikkö osasi kiertää vuoret ja sen liikkumiselle ei ollut asetettu mitään rajoituksia.

Tästä noin puolentoista viikon päästä peliin oli lisätty jo yksiköiden liikkumisen rajoitus sekä yksikön valinnan jälkeen tulevat ruudukko, joka näyttää minne asti yksikkö pystyy liikkumaan. Kyseinen ruudukko oli toteutettu vain ajoneuvoille, mutta samaa toteutusta pystytään käyttämään myös jalkaväelle sekä ajoneuvoille pienin muutoksin.

Seuraava kehitysversio on päivätty 7.3.2012, jolloin peliin on saatu lisättyä tehtaat, valikot tuotettavan yksikön valitsemiseen sekä yksikön tuottaminen tehtaasta. Tehtaiden ulkopuolella yksiköiden liikkumisruutujen näyttämislogiikkaa hiottiin, sillä vuoret aiheuttivat ongelmia ruutujen näyttämisen kanssa. Tässä kehitysversiossa korjattiin myös se, ettei yksikköä voi enää valita sen liikkumisen aikana. Tämä korjaus koski myös kaiken muun valintaa eli yksikön liikkuessa peli ei ota vastaan mitään komentoja.

Kahden viikon päästä 20.3. peliin oli lisätty vuorojärjestelmä sekä lisätty pelin käyttöliittymään lisättiin pelaajan nimen näyttäminen sekä vuoronvaihto-nappi. Pelissä estettiin myös yksiköiden valitseminen pelaajan selatessa valikkoja. Yksiköt eivät myöskään voi enää pysähtyä päällekkäin kartalla. Saman päivityksen aikana pelille oli myös pystytetty sekä mercurial repository sekä wunderkit sivu projektin koordinointia varten.

Seuraavan kahden viikon aikana pelin kehittämisessä keskityttiin pelin kameran säätämiseen sekä lisättiin yksikköjen värin vaihtaminen sen mukaan millä puolella ne ovat. Värin vaihtaminen kannatti tehdä koodipuolella, koska sillä saadaan säästettyä huomattavasti tilaa. Peliin ei tarvitse tällöin laittaa kuin yksi harmaa yksikkösprite kahden eri värisen spriten sijasta. Kameran toimintaa parannettiin lisäämällä mahdollisuus liikuttaa kameraa sekä tarkentamisominaisuus. Lisääminen ei kumminkaan ollut helppoa, sillä tässä vaiheessa huomasin että minulla oli vanhempi versio moottorista ja että tarkennusmahdollisuuteen tarvittava toiminnallisuus ei ole moottorissa automaattisesti vaan se pitää lisätä lisäosana moottoriin. Pienellä säätämisellä sain moottorin päivettyä viimeisimpään versioon ja lisäsin samalla tarvittavan lisäosan. Tein kameraa säätäessäni myös muutoksia yksiköiden valintaan.

Seuraavat kaksi viikkoa jatkettiin vielä pelin kameran ja visuaalisen ilmeen muutosten parissa. Uudet maastot sekä uusi tankin kuvake lisättiin peliin. Pelin kameraa säädettiin niin, ettei sitä voi enää liikuttaa pelialueen ulkopuolelle ja pelin kamera keskitetään jokaisen vuoron vaihdon jälkeen. Kameran resoluutio sekä maksimi/minimi zoomi lai-

tettiin mukautumaan käytetyn laitteen mukaan. Tämä aiheutti ongelman, jossa käyttöliittymän elementit eivät olleet aina niille tarkoitetuilla paikoilla, mutta tämä ongelma korjattiin nopeasti heti sen ilmaantumisen jälkeen. Peliin lisättiin myös kuvankaappaustyökalu, joka myöhemmässä kehitysvaiheessa huomasin turhaksi, sillä testilaitteestani löytyi kuvankaappausmahdollisuus.

Seuraavassa kehitysversiossa palattiin takaisin pelin ominaisuuksien lisäämiseen usean viikon kestäneen kamerasäädön jälkeen. Kyseisellä viikolla peliin lisättiin kaksi uutta yksikköä: kuljetushelikopteri sekä kiväärimies. Näille yksiköille tehtiin omat liikkumisrutiinit, sillä ne eroavat huomattavasti ajoneuvoista. Yksiköiden lataamiseen liittyvää koodia optimoitiin ja tehtiin selvemmäksi. Koodia muutettiin myös tulevaisuuden kannalta, jotta se osaa ladata useamman yksikön kerralla tarvittaessa.

Tässä kohtaa kehitykseen tuli lähes kuukauden mittainen paussi muiden koulutöiden ja projektien takia. Seuraava kehitysversio tuli 16.5.2012 ja muutoksia tulikin useampi. Pelin koodit jaettiin järkevimmin useampaan eri pakettiin sekä pelin lataus muutettiin asynkroniseksi, jotta tiedostoja voidaan ladata taustalla. Peliin lisättiin myös tallennus/lataus -ominaisuus joka toimii binääritiedostoksi serialisoitavalla tallennustietoluokalla. Peliin lisättiin myös perustoiminnot yksikköjen hyökkäystä varten, mutta tehtyjen vahinkojen viilaaminen jätettiin myöhemmäksi.

Seuraavassa kehitysetapissa peliin saatiin lisättyä kaupungit sekä päämajat. Unohdin tässä vaiheessa täysin lentokentät ja ne lisättiin myöhemmin. Rakennusten valloitus lisättiin myös tässä päivityksessä. Peliin lisättiin myös toimintavalikko, joka näytetään aina yksikön liikkumisen jälkeen tai jos yksikön valitsee ja koskettaa yksikköä uudestaan. Tästä valikosta yksikön voi käskeä hyökkäämään, valloittamaan, lastautumaan kuljetusyksikköön tai odottamaan.

Seuraavassa, 9.6.2012 julkaistussa kehitysversiossa peliin saatiin valmiiksi toimiva moninpeli. Moninpeli toimii ilmaispalvelimella, joka osoittautui hyvin epävakaaksi ja hitaaksi. Palvelimen toimiessa moninpeli kumminkin toimi hyvin. Moninpeli perustuu yhteen edestakaisin liikkuvaan tallennustiedostoon, johon molemmat pelaajat tekevät vuorollaan muutoksia. Peli rakentaa javalla apukirjaston kautta HTML -kyselyjä jotka

lähetetään palvelimelle jossa ne otetaan vastaan ja käsitellään PHP:llä. Tarvittaessa PHP tekee muutoksia MySQL -tietokantaan sekä lähettää ja ottaa vastaan tallennustiedostoja.

Seuraavassa kehitysversiossa peliin lisättiin kaikki yksiköt. Yksikköjä tuli kaiken kaikkiaan 13. Tällä viikolla huomasin myös sen, että pelissä on lentäviä yksiköitä, muttei lentokenttää lisättynä kartan paloihin tai koodiin. Tämä ongelma korjattiin nopeasti yksiköiden lisäämisen mukana. Yksiköiden hyökkääminen hiottiin samassa päivityksessä myös loppuun. Vahingon laskeminen perustuu Advance Warsin vahingonlaskukaavaan. Siinä hyökkäävä yksikkö tekee vahinkonsa ensin, jonka jälkeen puolustava yksikkö ampuu takaisin vähennetyllä voimalla. Vahingonteossa otetaan huomioon maasto, jossa yksiköt ovat sekä yksiköiden tyypit.

Yksiköiden lisäksi peliin lisättiin valuutta jota käytetään yksiköiden valmistamiseen sekä korjaukseen. Raha näytetään pelaajan nimen alla pelin käyttöliittymässä. Päivityksessä parannettiin myös yksiköiden liikkumisalueen näyttämistä sekä lisättiin samaa logiikkaa käyttävä yksiköiden hyökkäysalueen näyttö. Päivityksen jälkeen jalkaväkiyksiköitä pystyy myös lastaamaan kuljetusajoneuvoihin. Myös pelin käyttöliittymä sai parannuksia tässä päivityksessä. Menuista tehtiin paremmin järjestellyt, pelin käyttöliittymän elementtejä liikuteltiin järkevimpiin paikkoihin ja yksiköille lisättiin terveyspisteiden näyttäminen, jos niiden terveyspisteet tippuvat alle maksimin.

Seuraavan viikon päivitys toi mukanaan lisäyksiä tehtaiden valikoihin, joissa näytetään nyt yksikön tyyppi, sen liikkumismatka ja ynnä muut tiedot. Tämä oli myös viimeisiä päivityksiä pelin mekaniikoihin ja käyttöliittymään. Seuraavissa päivityksissä alettiin jo keskittymään pelin tekoälyyn.

Tekoälyn toteutus aloitettiin 7.8.2012 tuotantokontrolleista. Pelin käyttämiä kutsuja piti muokata hieman, jotta tietokoneen ohjaama tekoälyvastustaja pystyi käyttämään niitä järkevästi. Tuotantologiikan kehittäminen jätettiin kumminkin myöhemmälle ajankohdalle. Pelistä korjattiin myös yksi kriittinen virhe, joka aiheutti pelin kaatumisen, kun käyttäjä yritti ladata moninpelitallennusta.

Seuraavalla viikolla peliin lisättiin sekä liikkumis- että valloituskontrollit tekoälylle. Näidenkin ominaisuuksien käyttämiä piti hieman muokata että ne soveltuivat tekoälyn

käytettäviksi. Tekoälylle asetettiin myös säännöt, jonka mukaan se valloittaa rakennuksia. Valloituslogiikassa otetaan huomioon rakennuksen tyyppi, sen etäisyys yksiköstä sekä se, onko rakennuksen ympärillä vihollisen yksiköitä.

Tässä vaiheessa aiheelliseksi osoittautui pelin päätekoälyn toteutus, joka yhdistää jo tehdyt tekoälyn osat sekä tulevaisuudessa toteuttavat tekoälyn osat. Päätekoäly hoitaa kommunikoinnin muun pelin kanssa, ajaen vuoron alussa ja lopussa tarvittavia rutiineja, halliten tehtävien suoritusjärjestystä sekä ohjaten peliltä tulevia kutsuja oikeille tekoälyn osille. Tähän liittyy myös prioriteettitekoäly joka laskee tekoälyn hallitsemille yksiköille niille sopivan tehtävän.

Tämän jälkeen pureuduttiin pelin tekoälyn taistelukontrolleihin sekä logiikkaan. Ennen näiden toteuttamista moninpelistä löytyi jälleen virhe, joka esti yksiköiden valitsemisen jos ne ovat rakennusten päällä. Virhe löytyi myös tekoälyn kanssa pelattavasta yksinpelistä. Molemmat virheet korjattiin ja pääsin takaisin tekoälyn toteuttamiseen. Totuttuun tapaan taistelun käyttämiä kutsuja piti muokata, että ne soveltuivat tekoälyn käyttöön. Tekoälyn taistelulogiikkaan sisältyi jo valloituslogiikassa käytetty ominaisuus jolla katsotaan onko hyökättävän yksikön ympärillä muita vihollisen yksiköitä jotka voisivat olla uhkaksi yksikölle, jos se hyökkäsi kyseisen yksikön kimppuun. Logiikkaan kuuluu myös tekoälyn taito ennustaa taistelun lopputulos. Tekoäly ajaa simulaation taistelun kulusta ja hyökkää vain, jos simulaation lopputulos on sille suotuisa.

Opinnäytetyön kirjoitushetkellä tekoäly ei ole vielä täysin valmis mutta se on tarpeeksi valmis opinnäytetyöksi.

4. Pohdinta

4.1 Sisällön ja tulosten tarkastelu

Opinnäytetyön sisältö muuttui sen teon aikana muutamaan otteeseen. Nämä muutokset olivat lähinnä peliin tulevien ominaisuuksien lisäämistä tai poistamista. Suurimmaksi osaksi uusia ominaisuuksia lisättiin peliin ja kovin montaa asiaa ei tarvinnut karsia pois. Pelin mekaniikat olivat lainasivat todella paljon esikuvaltaan, koska kehitysvaiheessa tajusin, että täysin uusien mekaniikoiden suunnittelu ja tasapainostus olisi yksin kutakuinkin mahdotonta. Haasteena olikin enemmän se, pystynkö toteuttamaan täysin toisenlaisella alustavalla olevan pelin kloonin mobiilialustalle. Mekaniikoiden toteutus onnistui ja mielestäni pelin kloonaus toiselle alustalle onnistui hyvin. Pelin moninpeli ominaisuus todettiin testivaiheessa toimivaksi, mutta ainoaksi heikoksi lenkiksi muodostui käyttämäni ilmainen palvelin, joka toimi todella hitaasti ja epävakaasti. Pelin tekoäly on perustasolla toimintakunnossa, mutta vaatii vielä paljolti hienosäätöä ja kaikkien toimintojen yhteensovitusta. Pelin toiminnot onnistuin myös tekemään niin että niitä käyttää sulavasti sekä tekoäly että ihmispelaaja. Kokonaisuudessa olen tyytyväinen tekemääni peliin ja opinnäytetyöhöni.

4.2 Toteutuksen ja menetelmän tarkastelu

Pelin toteutus kulki välillä todella hyvää vauhtia eteenpäin, mutta välillä projektin eteenpäinvienti tuntui tuskallisen hitaalta. Pahin kömmähdys toteutusvaiheessa oli ilmaisen palvelimen kanssa. Kyseisellä palveluntarjoajalla on sääntö, jonka mukaan tilisi ja sivusi ilmaisella palvelimella deaktivoidaan, jollet käy hallintapaneelissa kuukauden välein. En tietenkään muistanut käydä hallintapaneelissa ja tilini deaktivoitiin. Paniikki iski, kun huomasin, ettei minulla ollut palvelinpuolen tiedostoja tallessa missään omalla koneella, vaan ne olivat pelkästään tällä ilmaisella tarjoajalla. Sain kumminkin tilini aktivoitua uudestaan ja sain pelastettua tiedostot. Tulevaisuudessa en käyttäisi ilmaista palvelua missään tapauksessa, vaan yrittäisin löytää mahdollisimman halvan palvelun jo etukäteen. Mahdollista olisi myös koko moninpeliominaisuuden poisjättö, jos hyvää palvelua ei löytyisi. Huonosti toimiva ominaisuus aiheuttaa minun mielestäni vain tur-

hautumista sekä kehittäjälle että käyttäjälle. Painiskelin myös pitkään muutaman ohjelmavirheen kanssa, joiden korjaaminen loppujen lopuksi osoittautui hyvin helpoksi.

Graafisella puolella toteutusta oli hoitamassa toinen henkilö, joka kumminkin valitettavasti joutui jättäytymään projektista pois henkilökohtaisista syistä. Tällaiset poisjäännit ovat valitettavia, muttei niille kumminkaan voi mitään ja niitä varten on hieman hankala varautua. Tulevaisuudessa varsinkin isompien projektien kanssa tekisin graafikon kanssa sopimuksen että saan käyttää hänen teoksiaan pelissä, vaikka hän lähtisikin jossain vaiheessa projektista. Graafikon lähteminen projektista ei onneksi haitannut omaa toteutuspuoltani sillä pystyin käyttämään korvikegrafiikoita ja ne voidaan korvata varsinaisilla grafiikoilla myöhemmin.

Tekoälyn toteutus hoitui helposti siten, että tein yhden tekoälyn osan kerrallaan ja sen jälkeen sovitin ne yhteen. Suunnitteluvaiheen jälkeen rakenteeseen tarvitsi tehdä vain pieniä muutoksia, jotka liittyivät lähinnä eri tehtävien suoritusjärjestykseen. Toteutuspuolella minut yllätti se, että tekoäly pystyi suorittamaan tehtäviä liiankin nopeasti. Tekoälyn toimintaa piti hidastaa keinotekoisesti sillä jos tekoälyn antoi suorittaa tehtävät niin nopeasti kun se pystyi, alkoi peli käyttäytyä kummallisesti. En tiedä, johtuiko tämä testilaitteestani vai pelimoottorin kyvyttömyydestä pysyä perässä.

Sain toteutettua tekoälyn eri osat, mutta vaikeimmaksi osaksi toteutusta on osoittautui niiden yhteistoiminta. Suurin osa tekoälyn toteutuksesta oli yritystä saada osaset toimimaan yhdessä järkevästi sekä prioriteettitekoälyn säätäminen niin että yksiköille annettaisiin aina optimaaliset tehtävät. Tämän kaltaisessa pelissä mahdollisia toimintoja on jokaisella yksiköllä todella suuri määrä ja niiden riippuessa todella paljon ympäristöstä, tekoäly vaatisi todella paljon pelitestausta, analysointia ja hienosäätöä toimiakseen kunnolla. Myös moottorin kanssa oli ongelmia. Peli kaatuilee hyvin outojen virheilmoitusten saattamana todella oudoissa kohdissa ja tämän kaltaisten ongelmien ratkaiseminen vei todella paljon aikaa. Tällaiset ongelmat ovat valitettavia, mutta uskon että tällaisiin ongelmiin on mahdollista törmätä moottorista riippumatta. Moottoria en kumminkaan enää vaihtaisi, mutta aivan aloitushetkellä hieman tarkempi eri vaihtoehtojen tutkiminen olisi saattanut olla tarpeen. Olen kumminkin tyytyväinen moottorivalintaani, vaikka dokumentoinnin vähyys onkin harmillista.

4.3 Oppimisprosessi

Projektin teko oli osittain minulle tuttua aiempien peliprojektien takia, mutta tämä oli kuitenkin ensimmäinen sooloprojektini, joten projektin kuluessa opin paljon uutta. Suurin uusi asia oli Androidille kehittäminen, jota en ole tehnyt koskaan ennen. Tulin hyvin tutuksi kyseisen alustan hyvien ja huonojen puolien kanssa. Opin myös uuden pelimoottorin käytön ja opin tuntemaan sen hyvät ja huonot puolet sekä miten tietyt asiat pitää hoitaa kyseisen moottorin kanssa. Opin myös takaisinmallinnusta, koska kyseisellä moottorilla ei ole kunnollista dokumentaatiota, vaan asioita pitää oppia moottorin lähdekoodia tutkimalla.

Tekoälyn kehittäminen tämänkaltaiseen peliin on laaja ja suhteellisen suuritöinen ottaen huomioon nykyiset taitoni ja tietoni. Päätin olla perustamatta tekoälyn toimintaa esikuvana toimivaan peliin, vaan tehdä sen alusta alkaen itse. Gamasutrasta löydetty artikkeli antoi idean tekoälyn pohjaan, jota kumminkin sovelsin mieleisekseni. Tekoälyä tehdessäni saatoin tehdä yhden osan tekoälyä, aloittaa toisen tekoälyn osan tekemisen ja keksiä tätä toista osaa tehdessä paremman tavan tehdä jokin tietty asia ja palasin takaisin ensimmäiseen tekoälyn osaan muuttamaan sitä paremmin toimivaksi. Tällainen kehitystapa oli minulle jokseenkin tuttu työharjoittelusta. Asiat on toki tarkoitus tehdä mahdollisimman hyvin heti ensimmäisellä kerralla. Myöhemmin kehityksessä saattaa ilmetä asioita tai tapoja, joita ei oltu ajateltu aikaisemmin ja mielestäni näiden muutosten teko on hyödyllistä ohjelman kokonaisuuden kannalta.

4.4 Kehittämisideat

Droidwarsilla on mielestäni jatkokehityspotentiaalia, mutta tehokkaaseen jatkokehitykseen tarvittaisiin mielestäni isompi kehitystiimi. Pystyn kyllä toteuttamaan ideoita, mutta en ole hirveän hyvä ideoimaan asioita tyhjästä. Projektiin tarvittaisiin siis muutama ihminen lisää ideoimaan ja suunnittelemaan yksiköitä sekä kenttiä. Mukaan tarvittaisiin myös äänimies tekemään pelin tarvitsemat äänet. Pelin graafinen puoli työllistää ainakin yhden, ehkä jopa kaksi graafikkoa, varsinkin jos peliin haluaisi muutakin grafiikkaa kuin itse pelin grafiikat.

Peliin voisi lisätä esimerkiksi tekoälyä vastaan pelattavan kampanja-tilan, jossa pelaaja istutettaisiin erilaisiin tilanteisiin ja kenttiin tekoälyvastustajaa vastaan. Tämänlaisen kampanjan suunnittelu ja toteutus vaatisi useamman henkilön ideoimaan sekä kenttiä että mahdollista juonta. Peliin voisi myös lisätä uusia yksiköitä ja muuttaa tämän hetkisiä yksiköitä jottei pelisi olisi kumminkaan liian lähellä esikuvaansa. Tämä lisäisi peliin huomattavasti lisää syvyyttä , mutta uusien yksiköiden keksiminen ja niiden tasapainottaminen jo olemassa olevien yksiköiden kanssa on hankalaa ja aikaa vievää.

Lähteet

Black P. 2006. Dijkstra's algorithm. http://xlinux.nist.gov/dads//HTML/dijkstraalgo.html. 11.6.2012.

Champandard A. 2007. Evolving with Creatures' AI: 15 Tricks to Mutate into Your Own Game. http://aigamedev.com/open/review/creatures-ai/. 6.11.2012.

Champandard A. 2007. Top 10 Most Influential AI Games. http://aigamedev.com/open/highlights/top-ai-games/. 6.11.2012.

Feigenbaum E. 2003. Some Challenges and Grand Challenges for Computational Intelligence. http://www.stanford.edu/class/cs227/Readings/20030101-JACM-Feigenbaum%20Some%20Challenges.pdf. 6.11.2012.

Generation5. 2012. Philosophical Arguments For and Against AI. http://library.thinkquest.org/18242/ai_phil.shtml. 24.5.2012.

Goertzel B. 2007. Artificial General Intelligence. http://people.inf.elte.hu/csizsekp/ai/books/artificial-general-intelligence-cognitive-technologies.9783540237334.27156.pdf. 24.5.2012.

Hietala J. 2012. Pelinkehitystä Susirajalla. http://pelinkehitys.blogspot.fi. 7.11.2012.

Lester P. 2005. A* Pathfinding for Beginners. http://www.policyalmanac.org/games/aStarTutorial.htm. 25.9.2012.

Mateas M. 2003. Expressive AI: Games and Artificial Intelligence. http://lmc.gatech.edu/~mateas/publications/MateasDIGRA2003.pdf. 6.11.2012.

McCarthy J. 2007. What is artificial intelligence?. http://www-formal.stanford.edu/jmc/whatisai/whatisai.html. 24.5.2012.

McCorduck P. 2008. Machines Who Think. http://www.pamelamc.com/html/machines_who_think.html. 11.6.2012.

Oppy, Graham, Dowe, David. 2011. The Turing Test. http://plato.stanford.edu/archives/spr2011/entries/turing-test/ 30.10.2012.

Paterl A. 2012. Introduction to A*. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html. 1.4.2012.

Pavlovic U. 2008. Brief History of Video Game AI. http://www.actiontrip.com/features/briefhistoryofvideogameai.phtml. 15.3.2012.

Poole D., Mackworh A., Goebel R. 1998. Computational Intelligence, A logical Approach. http://www.cs.ubc.ca/~poole/ci/ch1.pdf. 11.6.2012.

Sniedovich M. 2012. Dijkstra's Algorithm revisited: the OR/MS Connexion. http://www.ifors.ms.unimelb.edu.au/tutorial/dijkstra_new/index.html. 25.9.2012.

Tozorou P. 2008. Fixing Pathfinding Once and For All. http://www.ai-blog.net/archives/000152.html. 23.3.2012.

Voss P. 2002. Essentials of General Intelligence: The direct path to AGI. http://www.adaptiveai.com/research/index.htm#different_approach. 24.5.2012.

Welch E. 2007. Designing AI Algorithms For Turn-Based Strategy Games. http://www.gamasutra.com/view/feature/1535/designing_ai_algorithms_for_.php. 31.10.2012.

Wexler J. 2002. Artifial Intelligence in Games: A look at the smarts behind Lionhead Studio’s “Black and White” and where it can and will go in the future. http://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf. 15.3.2012.

A* Reitinetsinnän pseudokoodi

```
Pisteiden arvot:
M = Matkan hinta aloituspisteestä tämänhetkiseen pisteeseen
H = Heureettinen arvio matkan hinnasta tämänhetkisestä pisteestä päätepisteeseen
T = M ja H laskettuna yhteen, kertoo kuinka paljon matka aloituspisteestä
 päätepisteeseen tulisi maksamaan tämänhetkisen pisteen kautta
Tulopiste = Piste josta kyseiseen pisteeseen on tultu

Avoinlista = Lista avoimista pisteistä, sisältää alussa vain aloituspisteen
Suljettulista = Lista käydyistä pisteistä, alussa tyhjä

luokka AStar{
 funktio etsiReitti{
  While(Avoimien pisteiden lista EI OLE tyhjä)
   Tämänhetkinenpiste = alhaisimman T-arvon omaava piste avoimessa listassa
   If(Tämänhetkinen piste on päätepiste){
    Reitti on löydetty, lopeta
   }
   Else{
    Siirrä Tämänhetkinen piste suljettuunlistaan ja käy läpi sen naapuripisteet
    For(jokainen Naapuripiste){
     If(Naapuripiste on suljettujen pisteiden listassa{
      Siirryy seuraavaan Naapuripisteeseen
     }
     AlustavaM = Tämänhetkinen pisteen M-arvo + Matkan hinta Tämänhetkisen
     pisteen ja Naapuripisteen välillä
     If(Naapuripiste ei ole avoimessa listassa OR AlustavaM on pienempi kuin
     naapuripisteen M-arvo){
      If(Naapuripiste ei ole avoimessa listassa){
       Lisää naapuripiste avoimeen listaan
      }
      Aseta tämänhetkinen piste naapuripisteen tulopisteeksi
      Aseta naapuripisteen M-arvoksi AlustavaM
      Aseta naapuripisteen T-arvo
     }
    }
   }
  }
  Reittiä ei löydetty, lopeta
 }
}
```

Tekoälyn toiminnan pseudokoodi

```
luokka DroidWarsAI{
 AIYksiköt = Lista tekoälypelaajan yksiköistä
 AITehtävät = Lista tekoälypelaajan yksiköiden tehtävistä
 NykyinenTehtävä = Tällä hetkellä läpikäytävä tehtävä

 funktio aloitaAIVuoro{
  if (AIYksiköt on tyhjä){
   Etsi tekoälypelaajan yksiköt ja lisää ne listaan
  }
  else{
   Tarkista onko tekoäly tehnyt uusia yksiköitä viime vuorolla, jos on,
   lisää ne listaan
  }
  Tarkista kartalla olevat rakennukset
  Tarkista vihollisen yksiköt
  Aseta yksiköille tehtävät
  Aseta NykyinenTehtävä 0-vaiheeseen
  Suorita funktio AloitaNykyinenTehtävä
 }
 funktio AloitaSeuraavaTehtävä{
  Nosta NykyinenTehtävä yhdellä vaiheella ylöspäin
  Suorita funktio AloitaNykyinenTehtävä
 }

 funktio AloitaNykyinenTehtävä{
  switch(NyktinenTehtävä){
   Vaihe 0:
   Lähetä viesti taistelutekoälylle joka hoitaa tarpeelliset toimenpiteet
   tukikohdan suojelua varten.
   Suorita funktio AloitaSeuraavaTehtävä

   Vaihe 1:
   Lähetä viesti valloitustekoälylle joka katsoo aikaisemmin tarkastetut
   rakennukset ja jos tekoälypelaajalla on vapaita jalkaväkiyksiköitä,
   asetetaan ne valtaavaan rakennuksia.
   Suorita funktio AloitaSeuraavaTehtävä

   Vaihe 2:
   Lähetä viesti taistelutekoälylle joka tarkistaa vihollisen yksiköt ja
   asettaa sopivat yksiköt niiden eliminointiin.
   Suorita funktio AloitaSeuraavaTehtävä

   Vaihe 3:
   Lähetä viesti taistelutekoälylle joka katsoo onko jokin tekoälypelaajan
   yksiköistä vastustajan yksiköiden kantamalla. Jos kyseinen yksikkö on
   vihollisten yksiköiden kantamalla eikä se pärjää näille yksiköille,
   liikuta yksikköä pois vihollisyksiköiden kantamalta tai etsi yksikölle
   mahdollisimman suojaisa paikka.
   Suorita funktio AloitaSeuraavaTehtävä

   Vaihe 4:
   Lähetä viesti liikkumistekoälylle joka etsii yksikölle sopivan kohteen
   ja liikuttaa yksikköä tätä kohdetta kohti.
   Suorita funktio AloitaSeuraavaTehtävä

   Vaihe 5:
   Lähetä viesti tuotantotekoälylle joka tuottaa tekoälypelaajalle uusia
   yksiköitä. Yksiköitä tehdessä otetaan huomioon tekoälypelaajan varat
   sekä vihollispelaajan yksiköt.
   Suorita funktio AloitaSeuraavaTehtävä

   Vaihe 6:
   Lopeta Vuoro
  }
 }
```

```
 funktio LiikkuminenLoppunut{
  switch(NyktinenTehtävä){
   Vaihe 0:
   Lähetä viesti taistelutekoälylle että liikkuminen on loppunut
   Vaihe 1:
   Lähetä viesti valloitustekoälylle että liikkuminen on loppunut
   Vaihe 2:
   Lähetä viesti taistelutekoälylle että liikkuminen on loppunut
   Vaihe 3:
   Lähetä viesti taistelutekoälylle että liikkuminen on loppunut
   Vaihe 4:
   Lähetä viesti liikkumistekoälylle että liikkuminen on loppunut
  }
 }

 funktio ToimintaLoppunut{
  switch(NyktinenTehtävä){
   Vaihe 0:
   Lähetä viesti taistatelutekoälylle että toiminta on loppunut

   Vaihe 1:
   Lähetä viesti valloitustekoälylle että toiminta on loppunut

   Vaihe 2:
   Lähetä viesti taistelutekoälylle että toiminta on loppunut

   Vaihe 3:
   Lähetä viesti taistelutekoälylle että toiminta on loppunu

  }
 }
}
```