

Geneettinen algoritmi optimaalisen hyökkäystavan määrittämiseksi

Joel Kaasinen

30. marraskuuta 2006

Valkeakosken lukio
Matematiikka/Tietotekniikka

Tiivistelmä

Tutkielman aiheena oli geneettisten algoritmien soveltaminen taktiikoitten optimointiin mallinnetussa taistelussa. Tutkimuksessa käsiteltiin aluksi mahdollista metodiikkaa yleisellä tasolla, jonka jälkeen toteutettiin eräälle tapaukselle, ylivoimaa vastaan aikapaineen alla hyökkäämiselle, taktiikoita kehittävä geneettinen algoritmi.

Algoritmia ajettiin erilaisilla lähtöarvoilla passiivista vastustajaa vastaan, jotta saatiin todettua algoritmin toimivuus ja stabiilius. Tämän jälkeen tutkittiin algoritmin monimutkaisempiin tilanteisiin tuottamia ratkaisuja. Algoritmi osoittautui toimivaksi ja tarpeeksi suurilla populaatioilla nopeasti suppenevaksi. Algoritmi onnistui kehittämään mielekkäitä taktiikoita ja niiden parametreja.

Jatkotutkimuksena metodiikkaa voidaan soveltaa monimutkaisempiin ja epäsymmetrisempiin tilanteisiin, joissa puolien tavoitteet tai tappioiden sietokyvyt eriävät. Algoritmia laajemmalti käytettäessä todennäköisesti ilmenee tarvetta algoritmin hienosäädölle ja optimoinnille.

Sisältö

1	Johdanto	1
2	Taistelumallien taustoja	1
3	Sandis-malli	2
3.1	Sandis-mallin perusteet	2
3.2	Sotilaat tilakoneina	2
3.3	Osumamallinnus	3
4	Geneettiset algoritmit	4
4.1	Yleistä	4
4.2	Geneettisen algoritmin kulku	5
4.3	Risteytys ja mutaatio	5
5	Tutkimusprosessi	6
5.1	Tutkittava ongelma	6
5.2	Ratkaisuavaruuden koodaus	6
5.3	Hyvyysfunktio	7
5.4	Toteutuskieli	7
5.5	Tietorakenteet	8
5.6	Toteutetun algoritmin tarkka kulku	9
5.6.1	Taistelumalli	9
5.6.2	Geneettinen algoritmi	9
6	Testiajot ja tulokset	10
6.1	Testiajo 1	10
6.2	Testiajo 2	11
6.3	Testiajo 3	11
6.4	Testiajo 4	12
6.5	Testiajo 5	12
7	Pohdinta	12
7.1	Johtopäätökset	12
7.2	Jatkotutkimus	13
	Viitteet	13
A	Lähdekoodi	15

B	Testiajot	18
B.1	Lukuohje	18
B.2	Ajo 1	19
B.3	Ajo 2	24
B.4	Ajo 3	25
B.5	Ajo 4	26

1 Johdanto

Ihmiskunta on sotinut läpi aikojen, ja kaikkina aikoina on sotaa sekä taistelua pyritty ymmärtämään. Ensimmäisen matemaattisen — deterministisen ja analyttisen — taistelumallin loi Lanchester ensimmäisen maailmansodan aikana [1]. Tämän jälkeen taistelumallintamisessa on edetty varsinkin stokastisen mallintamisen alalla [2].

Geneettiset algoritmit ovat epädeterministinen ongelmanratkaisumetodiikka, joka pyrkii matkimaan luonnonvalinnan käyttäytymistä. Niitä on sovellettu onnistuneesti monenlaisiin ongelmiin ja niitten parissa on tehty paljon tutkimustyötä sitten John Hollandin vuonna 1975 julkaistun *Adaptation in Natural and Artificial Systems* -teoksen, joka voidaan nähdä geneettisten algoritmien tutkimuksen liikkeellepanijana.

Tutkielma käsittelee geneettisten algoritmien soveltamista optimaalisten taktiikoitten ratkaisemiseen taistelusimulaatioissa. Käytetty malli vastaa pääosiltaan Puolustusvoimien Teknillisessä Tutkimuslaitoksessa kehitettyä Sandistaistelumallinnusohjelmistoa ja tutkielma hyödyntää mallia varten tehtyjä tutkimuksia.

Tutkimuksen tavoitteena on kehittää geneettiseen algoritmiin pohjautuva metodiikka optimaalisten taktiikoitten ratkaisemiseen vaihteleviin tilanteisiin. Mahdollisten metodiikkojen tarkastelemisen lisäksi geneettinen algoritmi toteutetaan ja sitä sovelletaan esimerkkiongelmaan: ylivoimaista puolustusta vastaan etenemiseen käytettävissä olevan ajan ollessa rajoitettu. Tutkimalla optimaalisten taktiikoitten hakumetodiikkaa aikapaineen alla tämä tutkielma tukee Puolustusvoimien Teknillisessä Tutkimuslaitoksessa tehtävää taistelumallinustutkimusta.

2 Taistelumallien taustoja

Ensimmäisiä matemaattisia taistelumalleja olivat Lanchesterin kehittämät differentiaaliyhtälöt [1], jotka tunnetaan nimellä Lanchesterin tähdätyn tulen malli:

$$\begin{aligned}\frac{dR}{dt} &= -k_B B(t) \\ \frac{dB}{dt} &= -k_R R(t)\end{aligned}$$

Jossa R on punaisen puolen ja B sinisen puolen numeerinen miesvahvuus. k_R ja k_B ovat vastaavasti osapuolien taistelutehokkuusluvut. Lanchesterin

mallin heikkoutena on kuitenkin sen jatkuva ja analyyttinen luonne, joka tarjoaa vain korkean tason kuvan taistelun kulkuun [1].

3 Sandis-malli

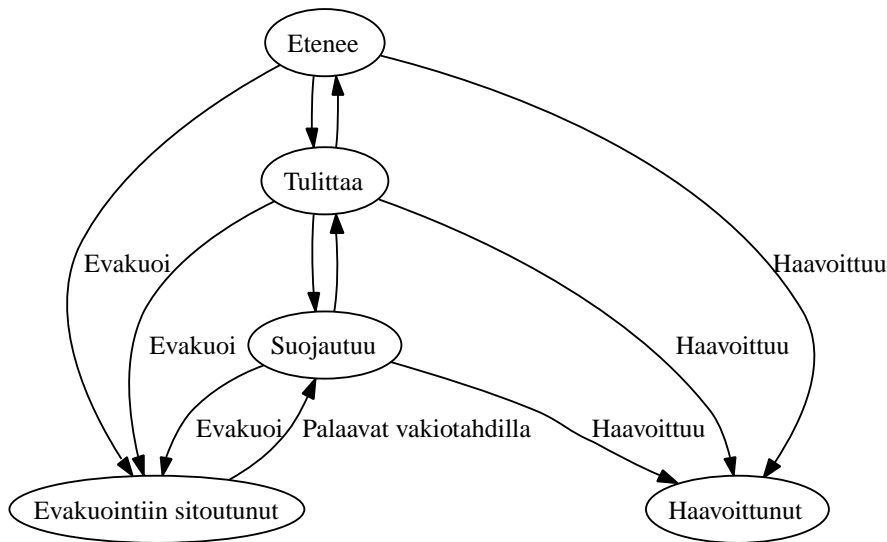
Seuraava esitys perustuu pääasiassa Lapin et al Sandis-ohjelmistoon[3] ja julkaisuihin [4] [5] sekä Kankaan diplomityöhön [2].

3.1 Sandis-mallin perusteet

Lanchesterin mallia tarkempia ennusteita tarvittaessa on kehitetty monia vaihtoehtoisia taistelumalleja varsinkin jalkaväkitaistelujen alueella. Näistä esimerkkeinä M. K. Laurenin kaupunkisodan fraktaalimalli [6] sekä Lapin et al Sandis-taistelumallinsohjelmiston käyttämä malli.

Sandis-malli on todennäköisyystilakonepohjainen malli, jossa sekä pataljoona että sen yksittäiset sotilaat mallinnetaan tilakoneilla. [3] Todennäköisyypohjaista taistelumallinnusta käsitellään julkaisuissa [2] ja [5].

3.2 Sotilaat tilakoneina



Kuva 1: Sotilaan tilat ja niiden väliset siirtymät Sandis-mallissa

Sandis-mallissa sotilaat esitetään yksinkertaisina tilakoneina. Tilat ja niiden väliset siirtymät esitetään kuvassa 1. Taistelukykyiset sotilaat jakautuvat

tiloihin “suojautee” eli hakeutuu suojaan tulelta muttei vastaa siihen, “tulittaa” eli aktiivisesti vastaa tuleen ja “etenee” eli pyrkii saavuttamaan tavoitteen vastaten tuleen. Näistä tiloista sotilaat siirtyvät haavoittuessaan tilaan “haavoittunut”. Lisäksi jokaisen haavoittuneen mukaan irroitetaan yksi sotilas evakuointiin, joka siirtyy tilaan “evakuointiin sitoutunut”. Evakuointiin sitoutuneet sotilaat palaavat tilaan “suojautee” vakiotodennäköisyydellä per aika-askel. [7]

Koska jokaisen tilasiirtymän kohdalla on siirtyvien sotilaitten odotusarvo helposti laskettavissa (osumatodennäköisyyksien laskemista käsitellään kohdassa 3.3), käytetään mallissa diskreettien sotilaitten sijasta muuttujia, jotka sisältävät sotilaitten määrän odotusarvon kussakin tilassa. [5]

3.3 Osumamallinnus

Sandis-mallissa ampumatapahtumat oletetaan riippumattomiksi, ja näin todennäköisyydeksi, että taistelija haavoittuu, kun häneen kohdistetaan n ammusta on (p yhden ammuksen osumistodennäköisyys):

$$1 - (1 - p)^n$$

Nyt kun n taistelijaa tulittaa kohti m :ää taistelijaa, on todennäköisyys yksittäisen taistelijan haavoittumiselle:

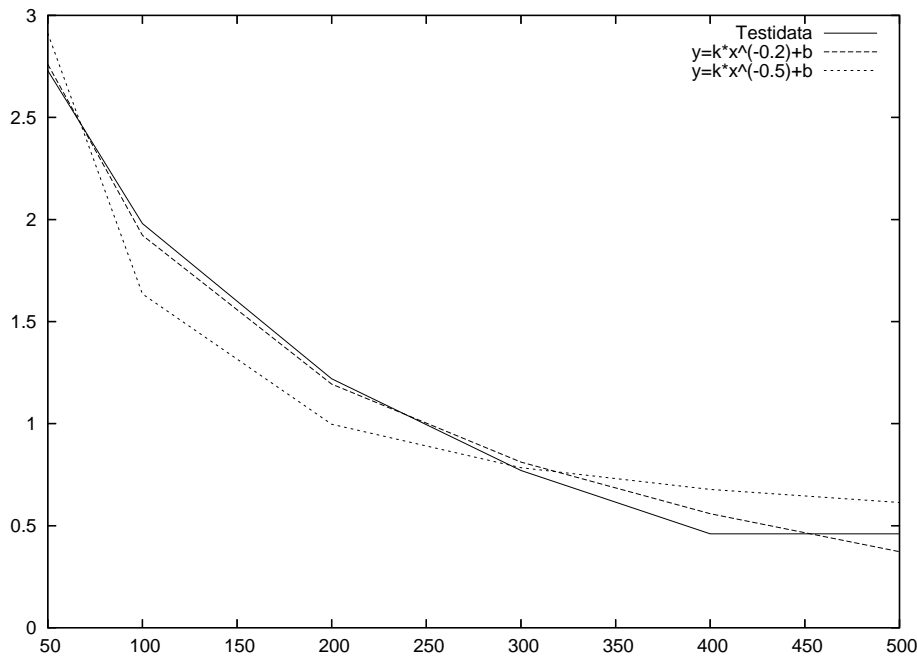
$$1 - (1 - p)^{\frac{n}{m}}$$

Lappi et al käyttivät Sandis-mallia kehittäessään vanhoja puolustusvoimien koeammuntojen tuloksia mallin parametrien (osumatodennäköisyyksien) (ks. taulukko 1) laskemiseen. Koska mallia tutkiskellaan abstraktisti, kiinnostaa meitä vain näitten parametrien etäisyysriippuvuus.

Etäisyys	0-50m	100m	200m	300m	400m	500m
Osumatarkkuus	2.73%	1.98%	1.22%	0.77%	0.46%	0.46%

Taulukko 1: Lapin et al. laskemat osumatodennäköisyydet kivääritulelle.[4]

Osumatarkkuus näyttäisi olevan kääntäen verrannollinen etäisyyden viidenteen juureen. Laskennallisista syistä olen kuitenkin käyttänyt etäisyyden neliöjuurta, joka antaa tarpeeksi hyvän approksimaation testidatalle (ks. kuva 2).



Kuva 2: Kaksi eri pienimmän neliösumman sovitusta ampumadataan

4 Geneettiset algoritmit

4.1 Yleistä

Geneettinen algoritmi on epädeterministinen kokeellinen ongelmanratkaisumetodiikka, joka pyrkii matkimalla luonnonvalintaa löytämään tehokkaasti hyvän (ei välttämättä parhaan) ratkaisun optimointiongelmaan [8]. Jotta geneettistä algoritmia voitaisiin käyttää optimointiongelmassa, on rakennettava

1. Ratkaisuavaruuden geneettinen esitys
2. Sopivuusfunktio

Ratkaisuavaruuden geneettisellä esityksellä tarkoitetaan koodausmetodiikkaa, jolla jokainen ratkaisu saadaan esitettyä jonona genejä, mielellään siten, että yhden geenin muutos ei muuta ratkaisua liikaa (näin mutaatiosta saadaan maksimaalinen hyöty). [8]

Sopivuusfunktio on jokin funktio, jolla ratkaisuihin saadaan liitettyä numeeriset hyvyysarvot.

4.2 Geneettisen algoritmin kulku

Geneettisen algoritmin eteneminen tapahtuu yleisessä muodossa seuraavasti (mukailtuna lähteistä [8] [9]):

1. Arvotaan n :n satunnaisen ratkaisun joukko
2. Ratkaisujen arvotus
3. Risteytys ja mutaatio
4. Ratkaisujen karsinta
5. Palataan kohtaan 2

Algoritmin toiminta perustuu siihen oletukseen, että hyvien ratkaisujen risteyttäminen tuottaa todennäköisemmin hyviä ratkaisuja kuin uusi satunnainen ratkaisu. Esimerkiksi testidatan sovituksia etsittäessä yhden termin tai kertoimen muuttaminen — termit ja kertoimet ovat yksittäisiä geenejä — antaa yleensä sovituksen, joka on lähellä alkuperäistä tai ainakin sen kaltainen. Samoin kahden hyvän sovituksen risteyttäminen — otetaan osa termeistä toisesta ja osa toisesta — pysyttelee lähellä testidataa jos alkuperäiset sovitukset olivat hyviä.

4.3 Risteytys ja mutaatio

Erilaisia metodeja seuraavaan ratkaisusukupolven konstruointiin on useita. Näistä [8] esittelee sukupolvimallin (generational algorithm), jossa koko ratkaisukanta korvataan uudella sekä elitismimallin, jossa edellisen sukupolven parhaat yksilöt jatkavat sellaisenaan seuraavaan sukupolveen.

Kuten aikaisemmin mainittiin, uudet ratkaisukandidaatit tuotetaan risteyttämällä vanhoja. Tämä toteutetaan käytännössä siten, että bitti- tai arvojenkoiksi koodatut ratkaisut pätkäistään ja uudelleenliitetään ristiin. Esimerkiksi geneistä

```
asdfj klg  
abbba bba
```

saadaan risteyttämällä välilyönnin osoittamasta kohdasta seuraavat uudet geenit:

```
asdfj bba  
abbba klg
```

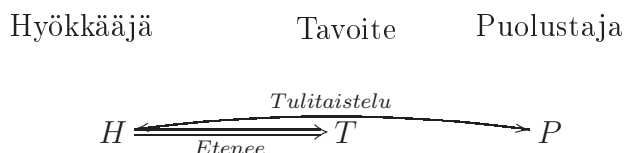
Tämä vastaa kromosomien risteytymistä luonnossa.

Pariutuvien yksilöiden valitsemiseksi yksi mahdollisuus on kaksintaistelumalli, jossa ratkaisut jotenkin pelautetaan toisiaan vastaan ja voittajat saavat pariutumisluvan. Toinen vaihtoehto olisi suhteuttaa pariutumistodennäköisyys sopivuusfunktion antamaan hyvyyslukuun. [8] Vielä yksi mahdollisuus on antaa elitismimallin suoraan siirtyvien yksilöitten pariutua muodostaakseen loppuosan seuraavasta sukupolvesta. Tällöin ratkaisujen monimuotoisuus tosin saattaa kärsiä.

5 Tutkimusprosessi

5.1 Tutkittava ongelma

Tutkimuksessa tarkasteltiin tilannetta, jossa ylivoimaa vastaan hyökkävällä joukkueella on saavutettavanaan tavoite (edetä x metriä) jonkin aikarajan sisällä:



Koska taisteluissa on nimenomaan kyse paremman osapuolen selviämisestä, on geneettisen algoritmin ja kaksintaistelumallin (ks 4.3) soveltaminen taktiikoitten ratkaisemiseen perusteltua. Samoista syistä on taktiikoitten arvotuksessa otettava huomioon omat tappiot mikäli halutaan mielekkäitä tuloksia — pitkän päälle viimeiseen mieheen taisteleva kansakuntahan kuolee sukupuuttoon.

Mallissa käytettävät parametrit eivät kuvaa mitään oikeaa taistelua vaan ne on valittu mielivaltaisesti ja niiden tarkoituksena on vain osoittaa metodiikan toimivuus.

5.2 Ratkaisuvaramuuden koodaus

Keskeisenä ongelmana geneettisen algoritmin soveltamisessa taistelumallinukseen on taktiikoiden koodaaminen geneiksi siten, että geneettisen algoritmin oletukset toteutuvat.

Tässä toteutuksessa käytetty malli on ärsyke-reaktio tyyppinen taktiikkaa ohjaava tekoäly. Osa geneistä koodaa, milloin taistelun olosuhteet aiheuttavat ärsyksen (esimerkiksi tappioiden noustessa yli 50%) ja loput, miten näihin ärsykkeisiin reagoidaan. Koska muuttujia on taistelussa suhteellisen

vähän, saadaan ärsykkeet ja reaktiot esitettyä yksinkertaisesti muodoissa “parametri n laskee/nousee ohi rajan k ” ja “muuta parametriä n määrällä k ”. Oppivia tekoälyjä taistelumallinnuksessa on jo tutkittu mm. Norjan puolustustutkimuslaitoksessa [10].

Risteytys tälle mallille voidaan hoitaa yksinkertaisesti valitsemalla osa ärsyke-reaktio-pareista toiselta vanhemmalta ja osa toiselta. Mutaatiossa pitäydytään yksinkertaisessa mutaatioalgoritmissa, joka tietyllä pienellä todennäköisyydellä korvaa ärsykeen tai reaktion satunnaisella uudella ärsykkeellä tai reaktiolla.

5.3 Hyvyysfunktio

Taistelusimulaatiota ajetaan valitun aika-askelmäärän verran (algoritmin kuvaus alempana). Tämän jälkeen taistelun lopputulos arvotetaan seuraavalla tavalla:

- Onko tavoite saavutettu?
 - Kyllä: Hyökkäyksen hyvyysluku on taistelukuntoisten miesten määrä
 - Ei: Puolustuksen hyvyysluku on etäisyys tavoitteesta
- Määrittelemätön hyvyysluku asetetaan toisen negaatioksi, ts. pätee, että *puolustuksen hyvyys* = *-hyökkäyksen hyvyys*.

Näin määritelty hyvyysfunktio asettaa sekä hyökkääjän että puolustajan taktiikat sopivaan järjestyksen, sillä hyökkäys on sitä onnistuneempi, mitä enemmän miehiä on jäljellä kun tavoite on saavutettu. Tämä vastaa intuitiivista käsitystä hyökkäyksen onnistumisesta. Toisaalta epäonnistunut hyökkäys on sitä parempi, mitä lähemmäs kohdetta on päästy. Tämäkin tuntuu intuitiivisesti varsin sopivalta hyvyysmitalta.

Tämä arvotus saattaisi suosia epäonnistuneille hyökkäyksille vääränlaisia taktiikoita, esimerkiksi “eteenpäin tappioista piittaamatta”. Kun testiajot tehdään tilanteissa, joissa on olemassa hyökkääjän voittostrategia, niin mahdolliset taktiikan vääristymät — kuten se, että edetään maaliin suuremmilla tappioilla kun on tarpeen — karsiutuvat pois optimaalisempien taktiikkoitten kehittyessä.

5.4 Toteutuskieli

Ensimmäiset prototyypit koodattiin Python-ohjelmointikielellä [11] mutta lopullinen toteutus tehtiin funktionaalisen paradigman Haskell-ohjelmointikielellä

[12], joka soveltuu ongelman ratkaisussa tarvittavien monien tietotyyppien ja algoritmien sekä algoritmien matemaattisen luonteen takia hyvin toteutuskieliksi.

5.5 Tietorakenteet

Seuraavassa hahmotellaan algoritmin käyttämä tietorakennehierarkia eli sitä, mitkä algoritmin komponentit vastaavat mitäkin mallinnettavia olioita. Ohjelman lähdekoodi on liitteenä (liite A). Mallinnettavia olioita mainittaessa niiden perässä on **tasalevyisellä** fontilla kyseiseen olioon liittyvän tietorakenteen tai funktion nimi lähdekoodin lukemisen helpottamiseksi.

Algoritmi mallintaa kahden *joukkueen* (koodissa `Team`) välistä taistelua. Joukkue koostuu *taktiikasta* (`Tactics`) sekä *joukkueen alkutilasta* (`BattleState`).

Joukkueen alkutila esitetään samalla tavoin kuin *taistelutila* (`BattleState`), joka sisältää tiedon joukkueen hetkellisestä *tilasta* (`TeamState`) sekä joukkueen etenemästä etäisyydestä. *Taistelun lopputilan* perusteella arvotetaan kummankin osapuolen taktiikat. Joukkueen alkutilana taistelutila ilmoittaa joukkueen aloituskokoonpanon sekä -etäisyyden.

Joukkueen *tila* (`TeamState`) on eri tiloissa olevien sotilaitten lukumäärät. Sotilaan mahdolliset tilat ovat (kuten Sandis-mallissakin, ks luku 3.2):

1. Etenee (`adv`)
2. Ampuu (`sh`)
3. Suojautuu (`cov`)
4. Evakuointiin sitoutunut (`evac`)
5. Haavoittunut (`wound`)

Taktiikka on lista *ärsyke* (`Event`) - *reaktio* (`Action`) pareja. Ärsyke koostuu tarkkailtavasta muuttujasta, ärsykerajasta sekä suunnasta. Ärsykkeen sanotaan *lauenneen* kun muuttuja ylittää ärsykerajan ärsykkeen osoittamaan suuntaan. Reaktio koostuu kohteesta, lähteestä ja lukumäärästä. Kun reaktio *laukeaa* (siihen liittyvän ärsykkeen lauetessa), siirretään lähdetilan sotilaita lukumäärä kohdetilaan, kuitenkin siten, ettei minkään tilan sotilaitten määrä laske negatiiviseksi. Reaktion lähde- ja kohdetilat on rajoitettu kolmeen ensimmäiseen tilaan (“etenee”, “ampuu”, “suojautuu”).

5.6 Toteutetun algoritmin tarkka kulku

5.6.1 Taistelumalli

Yhden taisteluaskeleen kuluksi valittiin toteutettaessa seuraava:

1. Lasketaan joukueitten toisillensa aiheuttamat tappiot ja siirretään tämän mukaisesti sotilaita tiloihin “haavoittunut” ja “evakuointiin sitoutunut”
2. Lasketaan kummankin joukkueen taktiikoitten mukaiset tilasiirtymät ja toteutetaan ne. Lisäksi lasketaan joukkueen eteneminen.

Sotilaat siirtyvät tiloista toisiin luvussa 3.2 kuvaillun mallin mukaisesti ja osumat käsitellään probabilistisesti kuten luvussa 3.3 esitetään. Tämän lisäksi osumatodennäköisyydet eri sotilasluokkiin on skaalattu seuraavalla tavalla:

$$p(\text{etenee}) = 2 p(\text{tulittaa}) = 4 p(\text{suojautuu})$$

Jossa $p(x)$ on tilassa x olevaan sotilaaseen osumisen todennäköisyys. Ampujina eri sotilasluokat käyttäytyvät samalla tavalla.

Eteneminen lasketaan siten, että jokainen tilassa “etenee” oleva sotilas siirtää joukkuetta eteenpäin yhden pituusyksikön. Etenemisessä joukkueet lähentyvät toisiaan ja näin siis osumatodennäköisyydet muuttuvat. Lähestymiselle on kuitenkin toteutettu alaraja, sillä kuten taulukosta 1 (s. 3) näkyy, on osumatodennäköisyys vakio lähietäisyyksillä.

Simulaatiota ajetaan *taistelun keston* verran askelia, jonka jälkeen taistelun senhetkinen tila eli lopputila arvotetaan ja tämän perusteella taistelun osapuolien taktiikoihin liitetään hyvyysluku, jonka perusteella suoritetaan myöhemmin valinta.

Yhdestä taistelun askeleesta vastaa koodissa `fight`, joka kutsuu funktioita `strike` vahinkojen laskemiseen, `think` taktiikan toteuttamiseen ja `advance` etenemisen laskemiseen. `battle` on kääre, joka tuottaa taistelun kulun listana taistelutilapareja.

5.6.2 Geneettinen algoritmi

Taktiikoitten jalostamista varten koodatun geneettisen algoritmin toteutukseen valittu metodiikka on:

- Potentiaalisten ratkaisujen joukkona on jatkuvasti n hyökkäystaktiikkaa.

- Ratkaisut alustetaan luomalla n satunnaista hyökkääjää (`randomTeams`). Jokaisella luodulla joukkueella on sama alkutila. Myös taistelun tavoite pysyy vakiona koko algoritmin kulun ajan.
- Hyvyysfunktio (`goodness`): toteutetaan kuten luvussa 5.3 esitetään.
- Seuraavan sukupolven valinta: Ajetaan taistelusimulaatio puolustajaa vastaan kaikille hyökkäjille. Lopputulokseen liitetään hyvyysarvo ja parhaat 5 siirretään suoraan seuraavaan sukupolveen. Loppuosa seuraavasta sukupolvesta saadaan pariuttamalla nämä parhaat hyökkäjät toistensa kanssa ja mutatoimalla jälkeläiset.
- Mutaatio: mutaatiofunktio (`mutate`), joka (pienellä) vakiotodennäköisyydellä muuttaa kutakin taktiikan komponenttia, ajetaan jokaisen askeleen lopuksi kaikille alkiuille. Mutaatiotodennäköisydeksi valittiin kiinteä 0.7%

`evolvea` on funktio, joka toteuttaa yhden askeleen geneettisestä algoritmista. `ea` on tämän kääre, joka ajaa halutun määrän evoluutioaskelia, ja palauttaa kehittyneet joukkueet.

Tämän algoritmin lisäksi kokeiltiin algoritmia, joka kehittää hyökkäys- ja puolustustaktiikoita samanaikaisesti. Tämä toteutettiin siten, että hyökkäjät ja puolustajat pariutettiin satunnaisesti ja taisteluitten lopputulosten perusteella ajettiin yllä kuvattu algoritmi. Mallin ongelmana oli kuitenkin se, että hyvät ratkaisut karsiutuivat pois kun huono hyökkäystaktiikka pariutettiin passiivisen puolustustaktiikan kanssa ja hyvä hyökkäystaktiikka hyvän puolustustaktiikan kanssa. Tällöin kävi usein niin, että huono hyökkääjä sai paremman tuloksen kuin hyvä ja siten pääsi etusijalla seuraavaan vaiheeseen toisin kuin luvussa 5.3 oletettiin.

6 Testiajot ja tulokset

Testiajot suoritettiin interaktiivisesta `ghci-haskell` tulkista. Ajojen tulosteet ovat liitteenä mutta niiden lukijan oletetaan perehtyvän ohjelman lähdekoodiin.

6.1 Testiajo 1

Ensimmäisen testiajon tarkoituksena oli varmistaa pelkästään hyökkääjiä kehittävän algoritmin toiminta. Ajon tuloste löytyy liitteestä B.2. Ajo suoritettiin 60 satunnaisjoukkueelle ja geneettistä algoritmia ajettiin kolme kertaa

40 kehitysaskelen ajan. Puolustajana oli staattinen 300 tulittajan joukkue ja hyökkäjien alkutilana 100 suojautuvaa sotilasta. Kohteena läheneminen 500:sta 100:aan metriin 10:ssä aika-askeleessa. Tuloksista näemme, että algoritmi suppenee hyvin nopeasti tässä tapauksessa tehokkaaseen massahyökkäystaktiikkaan: tulosteessa esiintyvät taktiikat kuten (tarkka syntaksi liitteessä B.1):

(5 <94,3->1 15), (4 <76,3->1 79)

joka siirtää heti alussa (molemmat ehdot toteutuvat alussa, kun haavoittuneita (5) ja evakuoijia (4) ei ole) 94 (79 + 15) miestä suojasta etenemään (3->1). Algoritmi suppenee myös joka ajokerralla likimain samaan parhaaseen tulokseen (4.7 – 4.8), joka viittaisi siihen, että samanlaiset taktiikat kehittyvät joka ajolla.

6.2 Testiajo 2

Toisen testiajon tarkoituksena oli tarkastella algoritmin skaalautuvuutta. Ajettiin kaksi samanlaista ajoa kuin testiajossa 1, mutta vain 10 joukkueen otoksella. Tuloksista nähdään että otoskoon pienentyessä taktiikka ei parane läheskään yhtä nopeasti, itseasiassa kummassakaan ajossa ei näy tuloksen johdonmukaista paranemista. Ainoastaan toisessa tapahtuu yksi hyppäys ylöspäin kolmannella aika-askeleella.

Ajon tuloste liitteessä B.3.

6.3 Testiajo 3

Kolmannessa testiajossa nostettiin aika-askelten määrää neljäänkymmeneen ja korvattiin passiivinen puolustaja sellaisella, joka siirtää joukkonsa vähitellen ampuma-asemiin, mutta nostaa tahtia evakuointiin sitoutuneitten määrän (eli haavoittumistahdin) kasvaessa. Geneettistä algoritmia ajettiin 100 askelta.

Liitteessä B.4 on kolme viimeistä ajoa (tulosteet lyhenneltyjä). Kussakin tapauksessa kehittyi taktiikka, joka ensin siirsi joukkoja taisteluasemiin, ja sitten tavoitteen saavutettuaan ryhmitti joukot puolustukseen. Seuraavassa eräs algoritmin tuottama, korkeimpaan hyvyyslukuun yltänyt taktiikka:

[(4 <49,3->2 88), (3 <56,2->1 19), (3 >45,3->2 63)]

Ensimmäinen ehto, 4 <49, toteutuu heti alussa eli suurin osa joukoista (88) siirretään tulitusasemiin. Myös kolmas ehto toteutuu heti ensimmäisellä askeleella ja loputkin sotilaat siirretään tuliasemiin. Toinen reaktio-ärsyke pari

siirtää tämän jälkeen (3 <56 pitää paikkansa luokan 3 tyhjennyttyä) joka askeleella 19 miestä etenemään. Taktiikka tuntuu järkevältä, ovathan etenijät vähiten suojattuja (ks. 5.6.1). Geneettinen algoritmi on siis löytänyt tulituen käytön etenemisen suojaamiseksi.

6.4 Testiajo 4

Neljännessä testiajossa päätettiin hakea algoritmin pitkällä ajolla taktiikoita hieman erilaiseen taistelutilanteeseen. Tilanne vastaa edellisen testiajon tilannetta, mutta kohteena on tällä kertaa vain 200 metrin eteneminen ja aikaa on 50 aika-askelta. Geneettistä algoritmia ajettiin 200 askelta.

Mielenkiintoista on huomata, että parhaan tuloksen (hyvyysarvo 65) saavutti tyyppiä

(5 >15,3->1 33), (5 >22,1->3 90)

oleva taktiikka, joka siis siirtää miehiä etenemään muutaman ensimmäisen askeleen ajan, mutta haavoittuneitten noustua yli tietyn rajan siirtää miehet suojaan. Muutaman etenemisaskeleen aikana joukkue etenee tavoitteeseen asti ja sitten suojautuu taistelun loppukestoksi vihollisen tulelta vähentääkseen tappioita. Liitteessä B.5 ovat kolmen testiajon tuottamat taktiikat sekä niitten hyvyysarvot.

6.5 Testiajo 5

Ajossa viisi sovelletaan algoritmia aivan erilaiseen tilanteeseen, ylivoiman hyökkäykseen. 400 miestä hyökkää 1000 metrin päästä tavoitenaan päästä 100 metrin päähän 100 miehen puolustusvoimasta. Puolustajien taktiikaksi valittiin ajossa kolme käytetty. Aikaa hyökkäykselle on toisessa ajossa 20 ja toisessa 50 askelta. Geneettistä algoritmia ajetaan 100 askelta.

Kumpikin ajo johti taktiikkaan, joka siirtää nopeasti kaikki joukot etenemään. Taktiikka ylsi parempiin tuloksiin aikaa ollessa vähän, se ei siis saanut puolustusta tuhottua täysin.

7 Pohdinta

7.1 Johtopäätökset

Algoritmi kehitti taktiikoita nopeasti eteenpäin tarpeeksi suurella populaatiolla. Toteutus onnistui välttämään mahdolliset elitismimallin sudenkuopat

(ks. 4.3). Yksinkertaisuudesta huolimatta, algoritmi osoittautui tehokkaaksi yksinkertaisissa tilanteissa.

Monimutkaisemmissa tilanteissa (esim. ajo 3) algoritmi onnistui kehittämään taktiikoita, jotka hyödynsivät useampaa ärsyke-reaktio paria sekä vastasivat intuitiivista käsitystä tositilanteissa käytetyistä taktiikoista. Lisäksi eri tilanteisiin kehitetyt taktiikat ovat yksilöllisiä joten algoritmi todellakin tuottaa tilannekohtaisia taktiikoita.

Geneettinen algoritmi yhdistettynä stokastiseen taistelumalliin näyttäisi lupaavalta apuvälineeltä taktiseen päätöksentekoon. Jo näin yksinkertainen toteutus sai metodiikalla tuotettua aidosti mielekkäitä tuloksia, eikä vain pelkkiä raakaan voimaan perustuvia taktiikoita.

7.2 Jatkotutkimus

Jatkotutkimuskohteina Puolustusvoimien Teknillisessä Tutkimuslaitoksessa ovat esimerkiksi erilaisten taistelutilanteitten parametroidit. Yhtenä mahdollisuutena esimerkiksi rauhanturvaoperaatiot, joissa osapuolien tavoitteet, voimasuhteet ja hyväksyttävät tappiot voivat vaihdella rajustikin. Joka tapauksessa on algoritmin uusiin tilanteisiin soveltamisessa avainasemassa uuden hyvyysfunktion toteuttaminen tilanteen vaatimuksia vastaavaksi.

Geneettisen algoritmin toteutus on hyvin karu — se ei esimerkiksi käytä useita lähteessä [8] mainittavista optimointikeinoista. Lisäksi pariutusmalli on hyvin yksinkertainen ja algoritmin nykyinen toteutus on jokseenkin hidas. Algoritmin jatkokehitys on tarpeellista, mikäli tutkielmassa esiteltyä metodiikkaa halutaan soveltaa laajempiin tilanteisiin. Yhtenä algoritmin laajennusmahdollisuutena on tutkielmassa toteuttamatta jäänyt puolustuksen ja hyökkäyksen yhtäaikainen kehittäminen.

Myös taistelumallista oli toteutettu vain yksinkertainen versio. Jatkotutkimuksessa voidaan taistelumallissa toteuttaa mm. eri sotilasluokkien erilaiset ampumaominaisuudet, eri asetyypit ja monimuotoisempi eteneminen (mallinnetussa maastossa eikä vain lineaarisesti). Näiden muutosten ei kuitenkaan pitäisi muuttaa geneettisen algoritmin pätevyyttä.

Viitteet

- [1] MacKay, N. J. Lanchester combat models. (Accepted) URL: <http://arxiv.org/abs/math.HO/0606300> (Luettu 8.10.2006)
- [2] Kangas, Lauri. Taistelun stokastinen mallinnus. Diplomityö, Systemianalyysin laboratorio, Teknillinen korkeakoulu 2005.

- [3] Lappi, Esa, Kangas, Lauri, Murtola, Teemu, Pajukanta, Santtu, Peussa, Yrjö & Pottonen, Olli. Sandis-ohjelmisto (Osa) 1.10.2006. PVT-TEIOS 2006. TLLIV Viranomaiskäyttö.
- [4] Lappi Esa & Pottonen Olli. Combat parameter estimation in Sandis OA-software. Lanchester and beyond, a workshop on operational analysis methodology. Defence Forces Technical Research Center Publications 11. Riihimäki 2006.
- [5] Lappi, Esa. SANDIS-ohjelmisto. Insinööriupseeri 3/2006. Insinööriupseeriliiton 80v juhlalehti. Insinööriupseeriliitto 2006 (Hyväksytty julkaistavaksi)
- [6] Lauren, M. K. Describing Rates of Interaction Between Multiple Autonomous Entities: An Example Using Combat Modelling. Defence Technology Agency, New Zealand Defence Force 2001
- [7] Lappi, Esa. A Markov chain based method to evaluate combat value of a platoon after battle casualties. Lanchester and beyond, a workshop on operational analysis methodology. Defence Forces Technical Research Center Publications 11. Riihimäki 2006.
- [8] Simula, Pekka. Johdatus evoluutiolaskentaan ja geneettisiin algoritmeihin. URL: <http://www.lce.hut.fi/teaching/S-114.240/k97/ga/>. (Luettu 28.11.2006)
- [9] Keet, Marijke. Genetic Algorithms - An overview. <http://www.meteck.org/gaover.html>. (Luettu 28.11.2006)
- [10] Kråkenes Tony & Halck Ole Martin. Learning to Play Operation Lucid from Human Expert Games. FFI/RAPPORT-2002/04041
- [11] The Python Programming Language URL: <http://www.python.org>
- [12] The Haskell Programming Language URL: <http://www.haskell.org>

A Lähdekoodi

```
import List
import System.Random
import Array
import Control.Monad
import Control.Arrow
import Debug.Trace

-- *** BASE *** --

data Event = Event { param::Int, thresh::Double, dir::Bool } -- dir = True if detect over thresh

instance Show Event where
  show (Event p t d) = show p ++ " " ++ (if d then ">" else "<") ++ show (round t)

instance Random Event where
  random g0 = (Event p t d,g)
  where (p,g1) = randomR (1,5) g0
        (t,g2) = randomR (0,100) g1
        (d,g) = random g2

data Action = Action { from::Int, to::Int, amount::Double }

instance Show Action where
  show (Action f t a) = show f ++ "->" ++ show t ++ " " ++ show (round a)

instance Random Action where
  random g0 = (Action f t a,g)
  where (f,g1) = randomR (1,3) g0
        (t,g2) = randomR (1,3) g1
        (a,g) = randomR (0,100) g2

type Tactic = (Event,Action)
type Tactics = [Tactic]

randomTactics :: Int -> IO Tactics
randomTactics 0 = return []
randomTactics n = liftM3 ( ( (:). ). (,) ) randomIO randomIO (randomTactics (n-1))

data Team = Team { wins::Integer, tactics::Tactics, init::BattleState }
  deriving Show

-- *** TEAMSTATE *** --

data TeamState = TeamState { adv::Double, sh::Double, cov::Double, evac::Double, wound::Double}
  deriving (Read,Eq)

instance Show TeamState where
  show (TeamState a s c e w) = "a:" ++ show a ++ " s:" ++ show s ++ " c:" ++ show c ++ " e:" ++ show e ++ " w:" ++ show w

type StateDiff = TeamState

data BattleState = BattleState { ts::TeamState, dist::Double }
  deriving (Show,Read)

instance Num TeamState where
  negate (TeamState a s c e w) = TeamState (-a) (-s) (-c) (-e) (-w)
  (TeamState a1 s1 c1 e1 w1) + (TeamState a2 s2 c2 e2 w2) =
  TeamState (a2+a1) (s2+s1) (c2+c1) (e2+e1) (w2+w1)
  (TeamState a1 s1 c1 e1 w1) * (TeamState a2 s2 c2 e2 w2) =
  TeamState (a2*a1) (s2*s1) (c2*c1) (e2*e1) (w2*w1)

instance Ord TeamState where
  compare (TeamState a1 s1 c1 e1 w1) (TeamState a2 s2 c2 e2 w2) =
  compare (a1+s1+c1) (a2+s2+c2)

fetch :: Int -> TeamState -> Double
fetch 1 = adv
fetch 2 = sh
fetch 3 = cov
fetch 4 = evac
fetch 5 = wound

-- n. of shooters
shoot st = adv st + sh st

diff :: Int -> Double -> TeamState
diff 1 n = TeamState n 0 0 0 0
diff 2 n = TeamState 0 n 0 0 0
diff 3 n = TeamState 0 0 n 0 0
diff 4 n = TeamState 0 0 0 n 0
```

```

diff 5 n = TeamState 0 0 0 0 n

nullstate = TeamState 0 0 0 0 0

-- *** AI *** --

-- return true when event is triggered by state
match :: TeamState -> Event -> Bool
match st ev | dir ev = a
            | otherwise = not a
            where a = fetch (param ev) st > thresh ev

-- return list of applicable actions
choose :: Tactics -> TeamState -> [Action]
choose tact st = map snd $ filter (match st . fst) tact

-- think and return new state, clipping actions as needed
think :: Tactics -> BattleState -> BattleState
think tact bs = bs{ts=foldl clipadd st acts}
  where st = ts bs
        acts = choose tact st
        clipadd st act = st + ((diff t x) + (diff f (-x)))
          where t = to act
                f = from act
                x = min (fetch f st) (amount act)

-- *** BATTLE *** --

-- advance team according to amount of soldiers in state "advancing"
-- clip distance at 50
advance :: BattleState -> BattleState
advance bs = bs { dist = x }
  where d = (dist bs) - (adv $ ts bs)
        x = if d<50 then 50 else d

-- probabilities of hitting respective soldier classes
-- at dist 100 this should be 2% for sh
hitProb dist = TeamState (0.04*t) (0.02*t) (0.01*t) 0 0
  where t = 10/sqrt dist -- 1 when dist is 100
evacratio = 0.2

-- calculate new state of defender after attacker has hit --
strike :: BattleState -> BattleState -> BattleState
strike att def = def{ts=TeamState (adv ds -2*advloss) (sh ds -2*shloss) (cov ds -2*covloss+evacreturn) (evac ds -evacreturn+loss) (wound ds +loss)}
  where f x | isNaN x = 0.0
        | otherwise = abs x
        ds = ts def
        as = ts att
        hit p = (p ds)*(1-(1-p prob)**(shoot as/p ds))
          where prob = hitProb (dist def)
        advloss = f $ hit adv -- advancer losses
        shloss = f $ hit sh -- shooter losses
        covloss = f $ hit cov -- covering losses
        loss = advloss + shloss + covloss
        evacreturn = evacratio*evac ds -- men returning from evac

-- calculate soldiers' status changes, casualties etc
-- i.e. do one step of battle simulation
-- attacker's distance is copied to defender's because defender is static
fight :: (Team,Team) -> (BattleState, BattleState) -> (BattleState, BattleState)
fight (att,def) (bs1,bs2) = (bs1',bs2') { dist = dist bs1' }
  where new bs2 bs1 t1 = advance $ think (tactics t1) $ strike bs2 bs1
        bs1' = new bs2 bs1 att
        bs2' = new bs1 bs2 def

-- return progress of battle as list
battle t1@(Team _ tact1 st1) t2@(Team _ tact2 st2) = battle_ (t1,t2) (st1,st2)
battle_ (t1,t2) (st1,st2) = (st1,st2):battle_ (t1,t2) (fight (t1,t2) (st1,st2))

-- *** GENETIC ALGORITHM *** --

-- Mutate team's tactics
mutate :: Team -> IO Team
mutate team = do t <- mapM mutate_ (tactics team)
  return (team{tactics=t})

mutate_ :: Tactic -> IO Tactic
mutate_ (e,a) = do s <- randomIO
  e' <- randomIO
  a' <- randomIO
  if s < 0.007 then return ((e', a)) else

```

```

        if s < 0.014 then return ((e, a')) else
    return (e,a)

-- Crossbreed teams
cross :: Team -> Team -> IO Team
cross t1 t2 = do t' <- cross_ (tactics t1) (tactics t2)
    return t1{tactics=t', wins=0}

cross_ :: Tactics -> Tactics -> IO Tactics
cross_ t1 t2 = do i <- randomRIO (0,min (length t1) (length t2) -1)
    return (take i t1++drop i t2)

-- Goal of attacker: achieve distance target in time simulation steps
data Goal = Goal { target::Double, time::Int }
    deriving (Show,Read)

-- "goodness" function for genetic algorithm: returns numeric value
-- for goodness of final state of team.
goodness :: Goal -> BattleState -> Double
goodness g bs | dist bs > target g = target g - dist bs
    | otherwise = adv s + sh s + cov s
    where s = ts bs

randomTeams :: BattleState -> Integer -> IO [Team]
randomTeams bs 0 = return []
randomTeams bs n = liftM2 (\x->(Team 0 x bs:)) (randomTactics 3) (randomTeams bs (n-1))

randomTArray ts n = randomTeams ts n >>= return . listArray (1,n)

-- shuffles list
shuffle :: [a] -> IO [a]
shuffle xs = return . map fst . sortBy (\a b -> compare (snd a) (snd b)) =<< return . zip (xs) =<< sequence (map (const (randomIO :: IO Integer)) xs)

-- evolve defenders and attackers simultaneously. DOESN'T WORK!!
evolveda :: Goal -> [Team] -> [Team] -> IO ([Team],[Team])
evolveda goal att def = do
def' <- shuffle def
let teams = zip att def'
let incw (x,(y,z)) | x>0 = (x,(y{ wins=1+wins y },z))
    | x<0 = (x,(y,z{ wins=1+wins z }))
    | otherwise = (x,(y,z))
-- Battle randomly paired teams, evaluate result, and increment wins of winners
let results = map (incw.(goodness goal.fst.(!!time goal).(uncurry battle) &&& id)) teams
putTraceMsg $ show $ (maximum (map fst results), minimum (map fst results))
let attsort = sortBy (\x y -> compare (fst y) (fst x)) results
let defsort = sortBy (\x y -> compare (fst x) (fst y)) results
let attbest = map (fst.snd) $ take 5 $ attsort
let attrest = map (fst.snd) $ drop 5 $ attsort
let defbest = map (snd.snd) $ take 5 $ defsort
let defrest = map (snd.snd) $ drop 5 $ defsort
let randomCross :: [Team] -> Team -> IO Team
    randomCross [] t = do tact <- randomTactics (length $ tactics t)
    return t{ tactics = tact, wins=0 }
    randomCross xs _ = do i <- randomRIO (0,(length xs)-1)
    j <- randomRIO (0,(length xs)-1)
    --putTraceMsg (show i ++ " ++ show j)
    cross (xs!!i) (xs!!j)
attnew <- mapM (randomCross attbest) attrest >>= mapM mutate
defnew <- mapM (randomCross defbest) defrest >>= mapM mutate
return (attbest++attnew,defbest++defnew)

-- wrapper for evolveda
eda :: Goal -> Integer -> ([Team],[Team]) -> IO ([Team],[Team])
eda _ 0 (att,def) = return (att,def)
eda g n (att,def) = evolveda g att def >>= eda g (n-1)

-- evolve attacker against static defender
evolvea :: Goal -> Team -> [Team] -> IO [Team]
evolvea goal def atts = do
let results = map (goodness goal.fst.(!!time goal).(flip battle) def) &&& id) atts
putTraceMsg $ show $ (maximum (map fst results), minimum (map fst results))
let attsort = sortBy (\x y -> compare (fst y) (fst x)) results
let attbest = map (snd) $ take 5 $ attsort
let attrest = map (snd) $ drop 5 $ attsort
let randomCross :: [Team] -> Team -> IO Team
    randomCross [] t = do tact <- randomTactics (length $ tactics t)
    return t{ tactics = tact, wins=0 }
    randomCross xs _ = do i <- randomRIO (0,(length xs)-1)
    j <- randomRIO (0,(length xs)-1)
    --putTraceMsg (show i ++ " ++ show j)
    cross (xs!!i) (xs!!j)
liftM2 (++) (return attbest) (mapM (randomCross attbest) attrest >>= mapM mutate)

```

```

-- wrapper for evolvea
ea :: Goal -> Integer -> Team -> [Team] -> IO [Team]
ea _ 0 def atts = return atts
ea g n def atts = evolvea g def atts >= ea g (n-1) def

-- *** FOR TESTING *** --

tact1=[(Event 1 79.0 False, Action 1 3 10.0), (Event 2 50.0 False, Action 3 2 1.0)]
tactnull=[]
bs1=BattleState (TeamState 300 0 0 0 0) 500.0
bs2=BattleState (TeamState 0 100 0 0 0) 500.0
t1=Team 0 tact1 bs1
t2=Team 0 tactnull bs2

as=BattleState (TeamState 0 0 100 0 0) 500.0
ds=BattleState (TeamState 0 0 400 0 0) 500.0

```

B Testiajot

B.1 Lukuohje

Main*> on ghci-tulkin komentorivi, kun ohjelmatiedosto on ladattu. Testiajot käynnistäneet komennot ovat mukana ajotulosteissa tulosten reprodusoinnin helpottamiseksi.

Algoritmi tulostaa ensin jokaisella ajo-askeleella hyökkääjän maksimi- ja minimituloksen ja sen jälkeen palauttaa lopulliset joukkueet. Algoritmia kutsutaan ghci-komentoriviltä siten, että lopuksi myös tulosjoukkueet tulostetaan.

Ärsykkeen syntaksi on: “<param> <suunta><arvo>” jossa “<param>” on parametrin numero (luvut 1-5 vastaavat tiloja “etenee”, “tulittaa”, “suojautuu”, “evakuointiin sitoutunut” ja “haavoittunut”). “<suunta>” on joko merkki “<” tai “>” ja indikoi sitä, laukeaako ärsyke parametrin arvon alittaessa vai ylittäessä arvon “<arvo>” . Esimerkiksi seuraava ärsyke laukeaa tulittajien määrän laskiessa alle 50:

```
2 <50
```

Reaktion syntaksi on: “<lähde>-><kohde> <määrä>” jossa “<lähde>” ja “<kohde>” ovat lähde- ja kohdeparametrien numerot ja “<määrä>” siirrettävien sotilaitten määrä. Esimerkiksi seuraava reaktio siirtää 30 miestä suojautumasta etenemään:

```
3->1 30
```

Taktiikka on lista (ärsyke,reaktio) -pareja eli se näytetään muodossa:

```
[(ärsyke1,reaktio1), (ärsyke2,reaktio2), ...]
```

Muut tietorakenteet tulostetaan samassa (lähes selväsanaisessa) formaatissa kuin niitten määrittelyt ovat lähdekoodissa (ks. liite A)


```

(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-453.0623834334639)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
(-6.473471199361327,-6.473471199361327)
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(1 <9,2->2 12),(2 <4,2->3 33),(2 <99,3->1 45)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}

```

B.4 Ajo 3

```

*Main> let d = (Team 0 [(Event 2 51 False,Action 3 2 50),(Event 4 50 True,Action 3 2 50)]) ds
*Main> a <- randomTeams as 60
*Main> ea (Goal 100.0 10) 40 d a >>= mapM_ print
(59.96765217511706,-400.0)
...
(61.08541312623597,61.08541312623597)
...
(61.398813631437605,61.08541312623597)
...
(61.398813631437605,-116.04025674284452)
Team {wins = 0, tactics = [(4 <49,3->2 88),(3 <56,2->1 19),(3 >45,3->2 63)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(4 <49,3->2 88),(3 <56,2->1 19),(3 >45,3->2 63)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
...
*Main> ea (Goal 100.0 40) 40 d a >>= mapM_ print
(1.205330987462959,-7.39106400433496e51)
...
(33.68256193231963,-400.0)
...
(42.20796178885298,42.20796178885298)
...
(43.16884657370979,42.20796178885298)
...
(49.92423494896064,38.35247260108976)
...
(49.92423494896064,-400.0)
Team {wins = 0, tactics = [(3 <87,3->1 70),(3 <24,1->3 90),(1 <12,1->1 40)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(3 <87,3->1 70),(3 <24,1->3 90),(1 <12,1->1 40)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
...
*Main> ea (Goal 100.0 40) 40 d a >>= mapM_ print
(1.205330987462959,-7.39106400433496e51)
(18.903170000967133,-1.1845430528526849e234)
...
(26.985356364447252,1.205330987462959)
...
(26.995177408595175,26.985356364447252)
...
(27.010239654678724,1.205330987462959)
...
(30.328206146607034,-400.0)
Team {wins = 0, tactics = [(3 <90,3->1 75),(2 <20,2->1 48),(5 >45,1->3 31)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(3 <90,3->1 75),(2 <20,2->1 48),(5 >45,1->3 31)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
*Main> ea (Goal 100.0 40) 100 d a >>= mapM_ print
(1.205330987462959,-7.39106400433496e51)
...
(33.68256193231963,33.68256193231963)
...
(53.215119135373975,1.205330987462959)
...
(53.215119135373975,-400.0)
Team {wins = 0, tactics = [(3 <93,3->1 70),(3 <66,1->3 90),(5 <66,2->1 40)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
Team {wins = 0, tactics = [(3 <93,3->1 70),(3 <66,1->3 90),(5 <66,2->1 40)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
...

```

B.5 Ajo 4

```
*Main> let d = (Team 0 [(Event 2 51 False,Action 3 2 50),(Event 4 50 True,Action 3 2 50)] ds)
*Main> a <- randomTeams as 60
*Main> ea (Goal 300.0 50) 200 d a >>= mapM_ print
...
(65.94878291761658,-200.0)
Team {wins = 0, tactics = [(5 >15,3->1 33),(5 >22,1->3 90),(5 <66,2->1 40)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
...
*Main> a <- randomTeams as 60
*Main> ea (Goal 300.0 50) 200 d a >>= mapM_ print
...
(38.47917184837891,-200.0)
Team {wins = 0, tactics = [(5 <91,2->1 100),(3 >47,3->2 44),(3 <69,1->3 26)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
...
*Main> a <- randomTeams as 60
*Main> ea (Goal 300.0 50) 200 d a >>= mapM_ print
...
(59.77805247217892,59.77805247217892)
Team {wins = 0, tactics = [(3 <18,1->1 90),(5 >17,3->1 86),(3 <29,1->3 96)], init = BattleState {ts = a:0.0 s:0.0 c:100.0 e:0.0 w:0.0, dist = 500.0}}
```