

Mika Lemström

Go-ohjelmien vertailu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

25.11.2015

Tekijä	Mika Lemström
Otsikko	Go-ohjelmien vertailu
Sivumäärä	26 sivua + 3 liitettä
Aika	25.11.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Fil. maist. Teemu Saukonoja
<p>Insinööriyön tarkoituksena oli tutkia ja vertailla eri tietokoneohjelmia, jotka pelaavat Goota. Tutkimuksessa selvitettiin, millä tavoilla tietokoneohjelmia voitaisiin parantaa ja kehittää, jotta ne tarjoaisivat kunnon vastuksen Goota amatikseen pelaaville. Go on yli 2500 vuotta vanha aasialainen lautapeli, jossa vasta viime vuosikymmeninä tietokoneohjelmat ovat saavuttaneet tason, jolla voittaa edistyneitä amatööripelaaajia. Tavoite oli selvittää valittujen tietokoneohjelmien mahdollisia heikkouksia ja ehdottaa niihin parannuksia.</p> <p>Insinööriyöprojektissa etsittiin kolmea yleistä avoimen lähdekoodin Go-tietokoneohjelmaa internetistä. Valitut tietokoneohjelmat asennettiin kahdelle eri tietokoneelle ja tutkittiin, miten ne saadaan pelaamaan toisiaan vastaan. Kaikki ohjelmat tukivat samaa protokollaa, niitä pystyttiin peluuttamaan keskenään. Ohjelmissa oli mukana ohjeet, joista saatiin selvitettyä mitä komentoja pelit vaativat voidakseen pelata monta peliä peräkkäin. Näin saatiin tulokset nopeammin selville. Insinööriyössä tutkittiin ohjelmia myös kooditasolla, joten tutkimuksessa tarkasteltiin vain sellaisia ohjelmia, joissa on pääsy lähdekoodiin. Tutkimuksen ulkopuolelle rajattiin suljetun lähdekoodin ohjelmat. Goota pelaavia ohjelmia on lukuisia, ja tässä tutkimuksessa vertailtiin niistä kolmea. Nämä Go-ohjelmat ovat nimeltään Gnugo, Fuego ja Pachi.</p> <p>Go-ohjelmia peluutettiin toisiaan ja itseään vastaan. Eniten pelejä voitti Pachi, seuraavaksi eniten pelejä voitti Fuego ja vähiten Gnugo. Ohjelmat eivät aina osanneet laskea pelien lopullista pistemäärää oikein, eivätkä ne löytäneet pelin oikeaa voittajaa. Tämä ilmeni, kun tulokset tarkastettiin manuaalisesti jälkikäteen.</p> <p>Pelitalanteiden tulkitseminen tuotti ohjelmille vaikeuksia, mistä seurasi, että ohjelmat luovuttivat pelejä ennenaikaisesti. Näin ollen ohjelmien pelitalanteiden hahmottamista tulisi parantaa.</p> <p>Lupaava Go-ohjelmiin liittyvä jatkotutkimuskohde voisivat olla syväoppivat konvolutionaaliset neuroverkot.</p>	
Avainsanat	keinoäly, koneoppiminen, avoin lähdekoodi, go-ohjelmat, go, 囲碁, baduk, 바둑, weiqi, 围棋

Author	Mika Lemström
Title	Comparison of Go Programmes
Number of Pages	26 pages + 3 appendices
Date	25 November 2015
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor	Teemu Saukonoja, M.Sc.
<p>The purpose of this thesis was to study and compare different computer programmes that play a game called Go, which is an Asian board game that is over 2,500 years old. Only during the last few decades have computer programmes reached the same playing skills as advanced amateur players.</p> <p>First, the thesis deals with the history of Go. Next it covers Go servers and Go client programmes with which Go is played on the Internet either against other people or other Go programmes. The thesis also deals with the Monte Carlo method and its applications which are most often used in the Go programmes. The programmes are also studied on a code level; only programmes giving access to a source code are looked into. Closed source programmes are excluded. There are numerous Go playing programmes, and in this study, three of them are compared. These are Gnugo, Fuego and Pachi.</p> <p>The Go programmes were played against each other and against themselves. The best programme was Pachi, the second best was Fuego and the worst was Gnugo. However, the programmes could not always count the final score correctly nor find the right winners which could be seen when the results were checked manually afterwards.</p> <p>The programmes had difficulties interpreting the situation on the game board. This caused the programmes to give up the game prematurely. Thus, how the programmes perceive different situations should be improved.</p> <p>A promising field for further studies relating to Go programmes could be Deep Convolutional Neural Networks. The Go programmes could play more human-like moves in the game.</p>	
Keywords	artificial intelligence, machine learning, open source, computer go, go programs, go, 囲碁, baduk, 바둑, weiqi, 围棋

Sisällys

Sanasto

1	Johdanto	1
2	Go – yleistä	2
2.1	Goon historia	2
2.2	Go internetissä	4
2.3	Go-palvelimien historia	4
3	Matemaattisen mallin soveltaminen	5
3.1	Monte-Carlon synty	5
3.2	Monte-Carlo-menetelmä	6
3.3	UCT-algoritmi	9
3.4	RAVE-algoritmi	10
4	Tutkittavat ohjelmat	11
4.1	Gnugo-ohjelma	11
4.2	Fuego-ohjelma	13
4.3	Pachi-ohjelma	14
5	Ohjelmien vertailua	16
5.1	Gnugo ja Pachi	17
5.2	Gnugo ja Fuego	18
5.3	Pachi ja Fuego	19
5.4	Gnugo ja Gnugo	20
5.5	Fuego ja Fuego	20
5.6	Pachi ja Pachi	21
6	Kehityskohteet	22
7	Yhteenveto	24
	Lähteet	25
	Liitteet	
	Liite 1 Mitä Go on?	
	Liite 2 Goon viralliset säännöt	
	Liite 3 Pelien tarkemmat tiedot	

Sanasto

AMAF		Kaikki-laitot-ensin (all-moves-as-first) -heuristiikka
Atari	アタリ	Ryhmä, jolla on vain yksi vapaus
BF		Paras-ensin (best-first) -menetelmä
Chūban	中盤	Keskipeli
Client		Asiakasohjelma, ottaa yhteyden palvelimeen
Dame	駄目	Alueitten rajalla oleva tyhjä risteys, joka ei tuo pisteitä
Dan	段	Mestari
DGS		Dragon Go Server, Go-palvelin
Go	碁	Peli, jossa tarkoituksena on vallata aluetta pelilaudalta
Gote	後手	Laitto, johon on pakotettu senten jälkeen
Hane	跳ね	Omaan kiveen nähden viistosti pelattu laitto, joka koskee vastustajan kiveen ja estää sen etenemisen yhteen suuntaan
Hoshi	星	Tähtipiste; merkitsee tasoituskivien paikkaa
Komi	コミ	Valkoiselle annettava pistemäärä korvauksena mustan aloituksesta
IGS		Pandanet Internet Go Server, Go-palvelin
Jigo	持碁	Tasapeli (pelitulos)
Joban	序盤	Alkupeli
Joseki	定石	Kirjalaitto(sarja), pelataan eniten alkupelissä
KGS		Kiseido Go Server, Go-palvelin
Kō	劫	Erikoistilanne, jossa pelaajan on kerran pelattava muualle
Konvoluutio		Kahden signaalin yhdistelmästä syntyvä uusi signaali

Sanasto jatkuu

Kyuu	級	Aloittelija
MCTS		Monte Carlo tree search, heuristinen algoritmi
Nakade	中手	Laitto, jonka pelaamalla voi estää ryhmän elämän
Neuroverkko		Itseoppiva, ihmisaivojen toimintaa ja matemaattista logiikkaa yhdistelevä laskentamalli
Ō	大	Iso, suuri
OGS		Online Go Server, Go-palvelin
RAVE		Rapid Action Value Estimation -algoritmi
Seki	セキ	Ryhmät, joilla yhteinen elämä
Sente	先手	Laitto, johon vastustajan on vastattava
Server		Palvelin, vastaa asiakasohjelmien pyyntöihin
Shūban	終盤	Loppupeli
Tengen	天元	Pelilaudan keskipiste
Tenuki	手抜き	Laitto muualle
UCT		Upper Confidence bounds applied to Trees -algoritmi
Wbaduk		Korealainen Go-asiakasohjelma

Käytän lopputyössäni seuraavia taivutus muotoja goosta:

Go [go:]

Taivutus

yks. nom. go

yks. gen. goon

yks. part. goota

yks. ill. goohon

Sana Go tulee japanin kielestä, ja siksi sanan muissa paitsi perusmuodossa on kaksi o-kirjainta.

1 Johdanto

Go on vanhan aasialainen lautapeli, jota pelataan paljon Kiinassa, Japanissa ja Koreassa, mutta pelin pelaaminen on yleistynyt myös länsimaissa. On kehitetty lukuisia sellaisia ohjelmia, jotka pelaavat Goota tietokoneella, mutta Goota ammatikseen pelaavat ihmiset voittavat vielä Goota pelaavat tietokoneohjelmat [1]. Ihmisten tämänhetkinen paremmuus johtuu todennäköisesti siitä, että kokenut ihmispelaaja hahmottaa vaistonvaraisesti ja paremmin erilaisia pelitilanteita. Tällaisia pelitilanteita ovat esimerkiksi pelialueen hahmottaminen sekä yksittäisten kivien ja kivetjujen strateginen arvo. Hyvin toimivalta ohjelmalta vaaditaan edistyksellistä hahmontunnistamista ja pelitietokannan tehokasta käyttöä.

Insinööriyön tarkoituksena on tutkia ja vertailla eri Go-ohjelmia. Valitsen kolme go-ohjelmaa ja peluutan niitä toisiaan ja itseään vastaan. Sen jälkeen analysoin pelatut pelit ja tarkistan manuaalisesti pelien tulokset ja vertaan niitä ohjelmien laskemiin tuloksiin. Tavoitteena on selvittää eri tietokoneohjelmien mahdollisia heikkouksia ja ehdottaa niihin parannuksia.

Goon säännöissä on pieniä eroavaisuuksia eri sääntövariaatioiden kesken [2]. Suurimmat erot liittyvät pistelaskuun ja saataviin hyvityspisteisiin (japaniksi hyvityspiste on $\sqsupset \xi$ (komi), joka on tasa-alulla valkean saama 6,5 pisteen etu). Rajaan tutkimukseni siten, että tarkastelen joitain sellaisia ohjelmia, jotka osaavat pelata goota japanilaisilla säännöillä. Sovellan tutkimuksessani japanilaisia vuoden 1989 sääntöjä, jotka ovat käännettynä liitteessä 2.

Insinööriyössäni rajaan tutkimuksen ulkopuolelle suljetun lähdekoodin ohjelmat, koska tarkoitukseni on tutkia ohjelmia myös kooditasolla. Tutkimuksessani käsittelen siis avoimen lähdekoodin ohjelmia [3]. Goota pelaavia ohjelmia on lukuisia, ja tässä tutkimuksessa vertaillen niistä kolmea, Go-ohjelmat ovat nimeltään Gnugo, Fuego sekä Pachi. Nämä kolme ohjelmaa käyttävät kaikki Monte-Carlo menetelmää pelatessaan goota. Ensimmäinen Go-ohjelma, joka käytti Monte-Carlo menetelmää oli nimeltään Gobble. Monte-Carlo menetelmästä kerron enemmän luvussa 3.

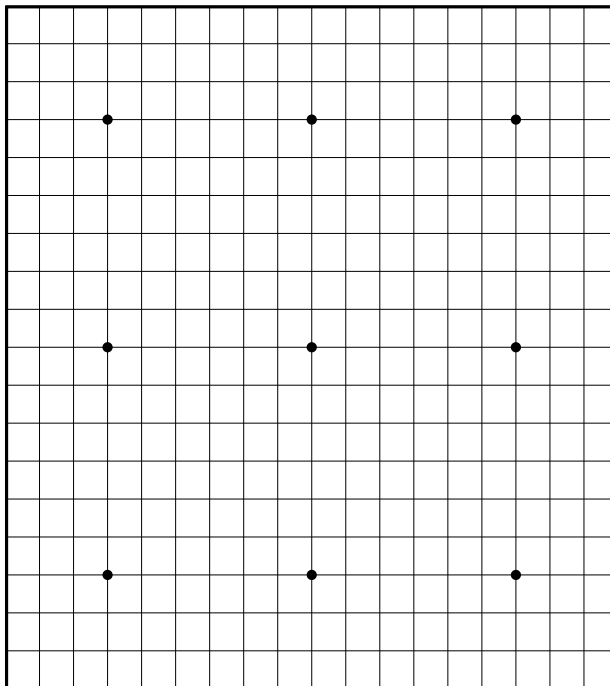
Työssäni käsittelen ensin Goon sääntöjä ja historiaa. Seuraavaksi käsittelen Go-asiakasohjelmien ja -palvelimien historiaa, joilla Goota voi pelata internetin välityksellä joko ihmistä tai tietokonetta vastaan. Sitten käyn läpi Monte-Carl-menetelmiä, joita yleisimmin käytetään Go-ohjelmissa määrittämään, mikä laitto seuraavaksi pelataan. Goossa ei siirretä nappuloita kuten shakissa, vaan ne

laitetaan paikalleen. Tämän jälkeen käsittelen, mitä eroja valittujen Go-ohjelmien välillä on. Lopuksi esittelen parannusehdotuksia ja teen yhteenvedon.

2 Go – yleistä

2.1 Goon historia

Go on yli 2500 vuotta vanha aasialainen peli [4], jota kutsutaan kiinassa nimellä 围棋 (weiqi), koreassa; 바둑 (baduk) ja japanissa; 囲碁 (igo) tai 碁 (go). Peliä pelataan pelilaudalla, joka koostuu vaakaja pystyviivoista. Pelaajat vuorottelevat laittamalla kiviä pelialueella olevien viivojen risteyskohtiin yksi kivi kerrallaan. Kuvassa 1 on täysikokoinen Go-pelilauta.

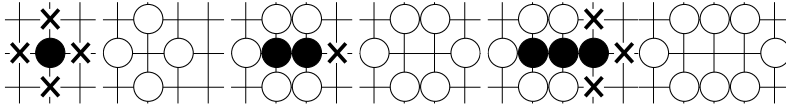


Hoshit (星) eli tähtipisteet merkitsevät mahdollisten tasoituskivien paikan.

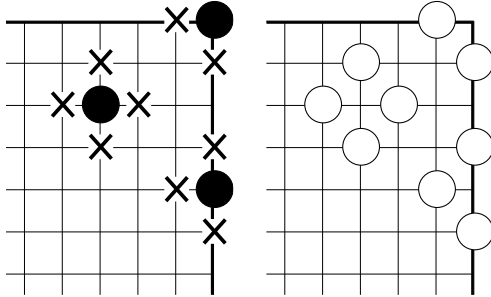
Kuva 1: Täysikokoinen 19 × 19 go-lauta.

Täysikokoinen pelilauta koostuu 19 vaakaja pystyviivan rajaamasta alueesta, ja sitä pelataan kahdenvärisillä kivillä, mustilla ja valkoisilla. Muita yleisiä pelilaudan kokoja ovat 9 × 9 ja 13 × 13, joita käytetään goon opettamisessa tai nopeammissa peleissä.

Pelin tarkoituksena on saartaa mahdollisimman paljon aluetta omilla kivillään, ja enemmän aluetta saartanut voittaa. Pelissä voi myös vangita vastustajan kiviä viemällä niiltä vapaudet (yhdellä kivellä voi olla neljä vapautta). Kun vapaudet viedään, kivet poistetaan laudalta. (Ks. kuvat 2 ja 3.)



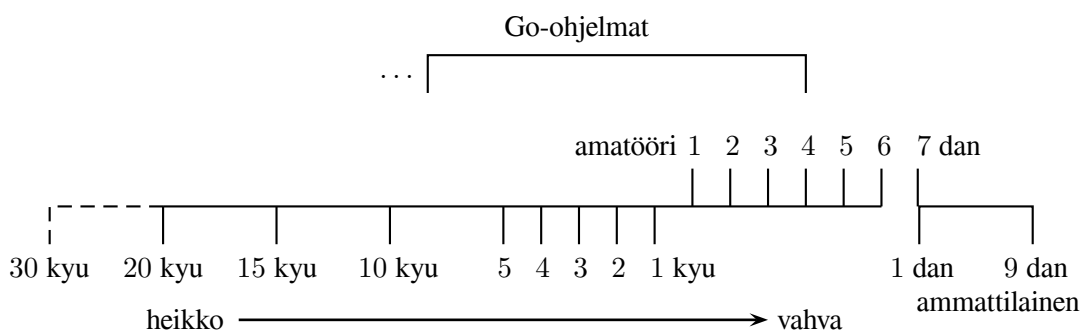
Kuva 2: Kiviä syödään. × mustan vapaus.



Kuva 3: Pelilaudan reunalla kivillä on vähemmän vapauksia.

Goossa on käytössä luokitusjärjestelmä, jossa kyu (jap. 級) tarkoittaa aloittelijaa ja dan (jap. 段) mestaria. Luokitusjärjestelmä on peräisin Kiinasta, mutta sitä formalisoitiin 1600-luvulla Japanissa, minkä jälkeen sitä on Goon lisäksi alettu käyttää muissakin yhteyksissä, muun muassa japanilaisessa kukkienasettelussa (jap. 生け花 ikebana) ja suurimmassa osassa itsepuolustuslajeja [5].

Go-ohjelmien taso nousee vuosi vuodelta, kun keksitään yhä parempia ja parempia tapoja opettaa Go-ohjelmille Goota [6, 7]. Tällä haavaa vahvimmat Go-ohjelmat ovat 4 danin vahvuisia. Kuvassa 4 esitetään Goon luokitusjärjestelmä.



Kuva 4: Go-luokitukset.

2.2 Go internetissä

Yhtenäinen kirjausformaatti helpottaa go-pelien seuraamista ja pelien läpikäyntiä internetissä. SGF eli Smart Game Format on yleisin goossa käytetty kirjausformaatti. Sen ensimmäisen version on kehittänyt Anders Kierulf, ja se julkaistiin vuonna 1987. Uusimman, versio neljän, on julkaissut Arno Hollosi vuonna 1997 [8]. Goon pelaaminen siirtyi manuaaliselta laudalta internetiin nopeasti. Ensin sitä pelattiin sähköpostin välityksellä; esimerkiksi GoWrite-lukuohjelma tukee sähköposti Goota ja useita kirjausformaatteja, muun muassa ugf- ja sgf-formaatteja. Yksi ensimmäisiä länsimaisia Go-palvelimia oli nimeltään IGS (Internet Go Server). Se käynnistettiin Meksikon yliopistossa helmikuussa 1992. Nykyään sen omistavat yhdessä korealainen ISP ja japanilainen NKB-yritys, ja siitä käytetään nimeä Pandanet IGS. Palvelimet yhdistävät botit ja pelaajat pelaamaan vastakkain, tai botit tai ihmiset keskenään.

Goota pelataan internetissä eripituisilla ajoilla sekä nopeina että pitkinä peleinä. Pitkät pelit saattavat kestää yli vuoden ja lyhyet minuutteja. Useat luvun 2.3 listassa olevat pelipalvelimet mahdollistavat vain nopeat, alle päivän kestävät pelit, OGS mahdollistaa myös pidemmät pelit, DGS vain pidemmät.

2.3 Go-palvelimien historia

Go-palvelimia on kahdenlaisia. Vuoropohjaisilla pelataan ilman aikarajoitusta tai pitkillä aikarajoituksilla esimerkiksi viikko per siirto. Reaaliaikaisilla pelataan lyhyillä, alle viikkon aikarajoituksilla. Yleensä aikarajaoletus on 30 minuuttia pääaikaa. Lisäksi jokaisesta siirrosta saa 10 sekuntia lisää aikaa (Fischer-ajanotto).

Palvelimiin saa yhteyden asiakasohjelmilla (client). Yleisimpiä go-palvelimia ovat

- OGS** Online Go Server (reaaliaikainen ja vuoropohjainen, asiakasohjelma web-pohjainen)
- IGS** Pandanet Internet Go Server (reaaliaikainen, asiakasohjelma web-pohjainen)
- KGS** Kiseido Go Server (reaaliaikainen, asiakasohjelma Java-pohjainen)
- NNGS** No Name Go Server, (ei enää toiminnassa, aikaisemmin reaaliaikainen)
Cyberoro (reaaliaikainen, (Wbaduk) asiakasohjelma Windows-pohjainen)
Tygem (reaaliaikainen, asiakasohjelma, vain Windows-pohjainen)
- DGS** Dragon Go Server, (vuoropohjainen, web-pohjainen).

Joka hetki näihin go-palvelimiin on yhteydessä keskimäärin seuraava määrä pelaajia: OGS (700, vain reaaliaikaiset pelit, määrä tarkistettu 9.2.2015), IGS (1 131, määrä tarkistettu 28.12.2013),

KGS (587, määrä tarkistettu 22.10.2015), Tygem (2 470, suosituin Tygem-palvelin, määrä tarkistettu 20.2.2015), Wbaduk (25 000, arvio) [9].

OGS osaa englantia, kiinaa, suomea, ranskaa, saksaa, italiaa, japania, puolaa venäjää ja espanjaa. KGS osasi alun perin vain englantia. Nykyään se osaa myös saksaa, espanjaa, suomea, ranskaa, italiaa, japania, koreaa, puolaa, portugalia, romanian, slovakiaa, turkkia ja ruotsia. Tygem osaa englantia, koreaa, kiinaa ja japania ja Wbaduk englantia, koreaa, kiinaa ja japania (alun perin vain koreaa).

3 Matemaattisen mallin soveltaminen

3.1 Monte-Carlon synty

Monte-Carlo on tilastollisen mallintamismenetelmä, jota on vuosien mittaan käytetty menestyksellisesti ratkomaan tieteellisiä ongelmia. Siinä hyödynnetään todennäköisyyslaskentaa ja tilastotiedettä. Alun perin se kehitettiin ennustamaan neutronien liikettä fissiopro sessissa [10].

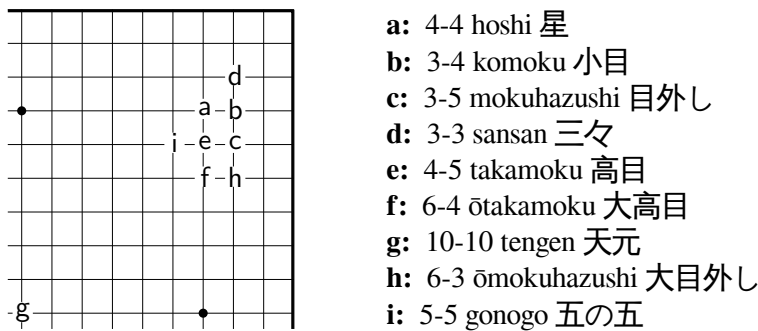
Menetelmän keksijä Stanisław Ulam sai idean laskiessaan, millä todennäköisyydellä voi voittaa Canfield-nimisessä pasianssissa (peli tunnetaan myös nimellä Demon [11]). Ulam esitteli idean vuonna 1946 John Von Neumannille. Von Neumann toimi tuolloin konsulttina Yhdysvaltain armeijan projekteissa. Vuonna 1947 Von Neumann kirjoitti Monte-Carlo-menetelmästä Robert Richtmyerille, joka toimi silloin armeijan palveluksessa Los Alamosissa (nykyinen Los Alamos National Laboratory) [12]. Kirjeen mukana oli luonnos ohjelmasta (tentative computer sheets), jota voisi käyttää ENIAC:ssa (Electronic Numerical Integrator And Computer).

Tämä oli ensimmäinen versio Monte-Carlo-menetelmästä. Vuonna 1947 Los Alamosissa menetelmää käytettiin ratkomaan ongelmia lämpöydin- ja fissiolaitteisiin liittyen, ja vuonna 1948 Ulam raportoi Atomienergiakomissiolle (Atomic Energy Commission), että menetelmä soveltui sekä kosmisen säteilyn (cosmic ray showers), että Hamilton-Jacobin osittaisdifferentiaaliyhtälön (Hamilton Jacobi partial differential equation) tutkimiseen. Se otettiin nopeasti käyttöön myös tutkittaessa muunlaisia ongelmia. [10.]

3.2 Monte-Carlo-menetelmä

Monte-Carlo-menetelmää käytettäessä tehdään sarja satunnaisia arvauksia, joista jokainen eliminoi joukon mahdollisia ratkaisuja. Tulos on sitä tarkempi, mitä enemmän arvauksia tehdään.

Goossa on mahdollista tehdä ensimmäinen laitto 361 eri paikkaan, mutta tyypillisesti pelaajat laittavat ensimmäisen laitton mustilla pelaavan näkökulmasta pelilaudan oikeaan yläneljännekseen. Kuvassa 5 esitellään yhdeksän yleisintä aloitusta.



Kuva 5: Yleisimmät aloituslaitot ja niiden japaninkieliset nimet [13].

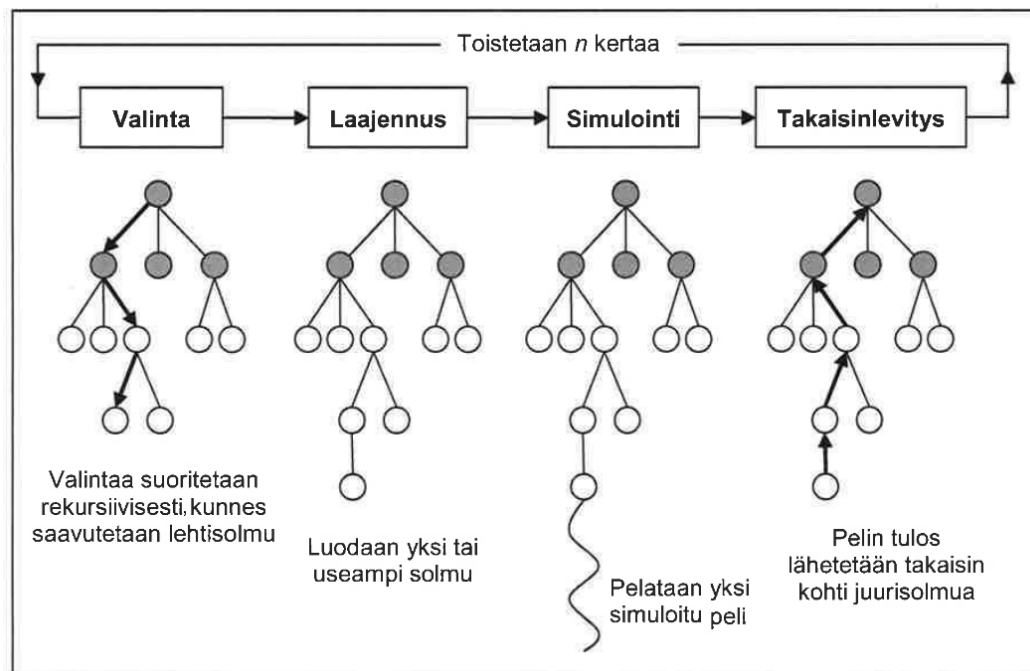
Goossa pelipuu eli kaikkien mahdollisten pelien määrä lasketaan ottamalla kertoma luvusta 361, joka on $361! = 1,4 \cdot 10^{768}$. Tämä ei tosin kuvaa todellista valintojen määrää, sillä mukana ovat myös laittomat pelitilanteet, esimerkiksi kiven pelaaminen risteykseen, jonka ympärillä ei ole kivelle, vapauksia. Taulukko 1 kuvaa erikokoisia pelipuita. Taulukossa näkyvät risteykset n ja permutoimalla lasketut pelipuiden koot $n!$. Taulukossa 1 näkyvät myös keskivertopelien laittomäärät l ja niiden avulla lasketut uudet pelipuiden koot n^l , jotka antavat paremman kuvan todellisista pelipuiden kooista [14].

Taulukko 1: Pelipuun koko n^l , jossa n = risteyksien lukumäärä ja l = laittojen määrä [14].

Laudan koko	Risteyksiä n	$n!$ (täydellinen pelipuu)	pelin pituus l (keskimäärin)	n^l (pelipuu käytännössä)
3×3	9	$3,6 \cdot 10^5$	5	$5,9 \cdot 10^4$
5×5	25	$1,6 \cdot 10^{25}$	15	$9,3 \cdot 10^{20}$
9×9	81	$5,8 \cdot 10^{120}$	45	$7,6 \cdot 10^{85}$
13×13	169	$1,6 \cdot 10^{304}$	90	$3,2 \cdot 10^{200}$
17×17	289	$2,1 \cdot 10^{587}$	130	$8,3 \cdot 10^{319}$
19×19	361	$1,4 \cdot 10^{786}$	200	$3,2 \cdot 10^{511}$
21×21	441	$2,5 \cdot 10^{976}$	250	$1,3 \cdot 10^{661}$

Go-ohjelmissa käytetään Monte-Carlon parannettua versiota. Yksi näistä menetelmistä on Monte-Carlo-puuhaaku (Monte-Carlo Tree Search), tästä eteenpäin MCTS, joka on niin sanottu paras-ensinmenetelmä (best-first method). Aluksi MCTS-haut ovat täysin satunnaisia, mutta myöhemmin se kykenee ennustamaan siirtoja, jotka ovat todennäköisesti parempia [14].

MCTS:llä tarkoitetaan sitä, että jokainen puuhaun tila eli solmu kuvaa yhtä pelilaudan tilannetta. Tila sisältää vähintään seuraavat kaksi tietoa: tilan senhetkisen arvon, joka on yleensä solmun läpi simuloitujen pelitulosten keskiarvo, ja solmuun tehtyjen vierailukertojen määrän. MCTS alkaa yleensä siten, että puussa on ainoastaan juurisolmu, jota lähdetään laajentamaan. Varsinainen hyvän laitton etsiminen koostuu neljästä vaiheesta (kuva 6), joita toistetaan niin kauan, kuin on aikaa.



Kuva 6: Puuhaun (MCTS) yleiskuvaus [14].

Kuvassa 6 vaiheet ovat seuraavat:

1. **Valinta:** Puun juuresta matkataan lehtisolmuun käyttäen valittua strategiaa.
2. **Laajennus:** Siitä lehtisolmusta, johon on päästy, liitetään yksi tai useampia lapsisolmuja puuhun valitun laajennusstrategian mukaisesti.
3. **Simulointi:** Ohjelma valitsee yhden simulointistrategian ja pelaa sillä niin kauan, kunnes kyseinen peli loppuu. Simuloidun pelin tulos on $+1$, jos musta voitti, ja -1 , jos valkoinen voitti. Lopputulos voidaan laskea joko kiinalaisen tai japanilaisen pistelaskujärjestelmän mukaan.
4. **Takaisinlevitys:** Ohjelma pelaa pelejä lukuisia kertoja, vertaa laittoja ja valitsee lopuksi sen laitton, jossa on eniten vierailukertoja.

Esimerkkikoodi MCTS:lle esitetään koodiesimerkissä 1.

```
void MCTS(Node root node)

while (has time)
{
    current node ← root node
    while (current node ∈ ST )
    {
        last node ← current node
        current node ← Select(current node)           // Selection
    }
    last node ← Expand(last node)                     // Expansion
    R ← P lay simulated game(last node)              // Simulation
    while (current node ∈ ST )
    {
        current node.Backpropagate(R)                // Backpropagation
        current node.visit count ← current node.visit count + 1
        current node ← current node.parent
    }
}
return best move = argmax N ∈ N c (root node) (N.visit count)
```

Koodiesimerkki 1: Pseudo-koodi MCTS:hen [15].


3.3 UCT-algoritmi

On olemassa myös UCT-algoritmi (Upper Confidence bounds applied to Trees), joka on tarkennettu versio MCTS:stä ja jota käytetään goota pelaavissa ohjelmissa. UCT käsittelee solmuja tehokkaammin kuin MCTS siten, että se kasvattaa pelipuuta epäsymmetrisesti laajentaen niitä solmuja, jotka vaikuttavat parhailta. Jos käytössä on tarpeeksi aikaa, UCT takaa optimaalisen laitton löytymisen [16].

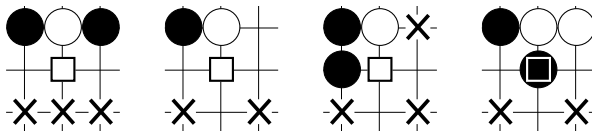
Alla esitetään kaava, jolla UCT saa suurimman arvonsa.

$$Q^\oplus(s, a) = Q(s, a) + c \sqrt{\frac{\ln N(s)}{N(s, a)}}$$

- $N(s)$ simulaatioiden s kokonaismäärä
- $Q(s, a)$ keskiarvo kaikista simulaatiosta s , jossa oli mukana toiminto a
- $N(s, a)$ simulaatioiden määrä s , jossa toiminto a valittiin
- c vakio, jonka teoreettinen arvo lähellä $\sqrt{2}$, käytännössä valitaan empiirisesti [17]

Ohjelmia voidaan tehostaa käyttämällä valmiita siirtokirjastoja ja 3×3 -alueella olevia kuvioita. Ohjelma käy läpi eri vaihtoehtoja etsien parhaan mahdollisen laitton, jonka pelata. Hane (japaniksi 跳ね) on omaan kiveen nähden viistosti pelattu laitto, joka estää vastustajaa yhdestä suunnasta .

Kuvassa 7 esitetään eri vaihtoehdot hane-laitolle. Jonkin kuvissa olevan kuvion täytyy täsmätä täysin, jotta laitto olisi hane. (○● valmiina olevat kivet; rastin × kohdalla olevia kiviä ei huomioida; neliö □ kuvaa pelattavaa kohtaa. Viimeisessä kuviossa täytyy olla mustan vuoro, jotta kuvio täsmäisi.)



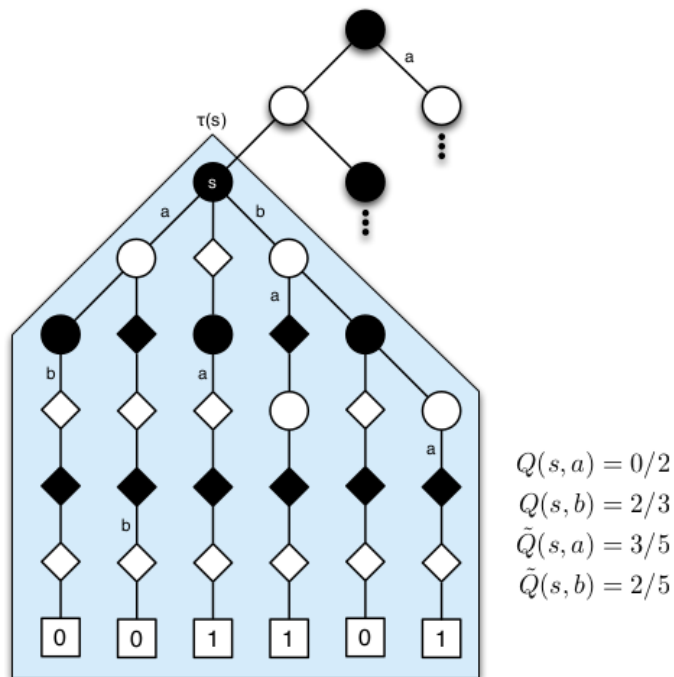
Kuva 7: Hane-kuvioita [19].

3.4 RAVE-algoritmi

RAVE eli Rapid Action Value Estimation -algoritmi on algoritmi joka yhdistää Monte-Carlo-puuhaun ja kaikki-laitot-ensin-heuristiikan (all-moves-as-first heuristics). Kaikki-laitot-ensin-heuristiikka käsittelee kaikkia laittoja siten, kuin jokainen laitto olisi ensimmäinen. Kaikki-laitot-ensin-heuristiikka myös pystyy jakamaan tietoa eri simulaatioiden tilanteiden kesken tallentamalla parhaat saadut tulokset muistiin. Tämä auttaa pienentämään pelipuun kokoa. Heuristiikan huonona puolena on kuitenkin se, että algoritmista saadut tulokset eivät todennäköisesti ole luotettavia. Siten RAVE-algoritmi soveltuu parhaiten laittojen valitsemiseen tilanteessa, jossa niiden määrä on rajoitettu muutamaaan parhaaseen [17].

Kuvassa 8 näkyy, kuinka RAVE-algoritmi löytää parhaat pelattavat laitot.

RAVE-algoritmin peruseriaate on, että se yleistää alipuista löydetyt laitot ja ottaa myös oletusarvoisesti huomioon laittojen paremmuuden.



Kuva 8: Esimerkki RAVE-algoritmin käytöstä. RAVE-algoritmi arvioi mustan laittoa a ja b tilanteessa s . Tilasta s on tehty kuusi simulaatiota, ja niiden tulokset näkyvät neliöissä kuvan alaosassa. Laitto a johti kahteen häviöön, joten Monte-Carlo-menetelmä suosii laittoa b . Pelattaessa laitto a milloin tahansa myöhemmin, se johti voittoon kolmessa tilanteesta neljästä, joten RAVE-algoritmi suosii laittoa a . Simulaation aloittava laitto a ei kuulu alipuuhun $\tau(s)$. [17.]

4 Tutkittavat ohjelmat

4.1 Gnugo-ohjelma

Gnugo on Goota pelaava vapaa ohjelma (free software). Se on käännetty muun muassa seuraaville alustoille: GNU/Linux, Unix, Windows ja Mac OS sekä Mac OS X. Ohjelmalla ei ole omaa graafista käyttöliittymää, mutta sitä on mahdollista käyttää muilla ohjelmilla [20]. Gnugon ensimmäinen vakaa versio (gnugo 1.1) julkaistiin marraskuussa 1989 ja viimeisin vakaa versio (gnugo 3.8) julkaistiin helmikuussa 2009. Uusin kehitysversio (gnugo 3.9.1) ohjelmasta joulukuussa 2010. Gnugon-luokitus on kasvanut tasaisesti alun 30 kyuusta 5–7 kyuhyn.

Ohjelma käyttää Monte-Carlo-menetelmää päättäessään laitoista. Koodiesimerkissä 2 Gnugo alustaa lautatilanteen tarkempaa tarkastelua varten.

```

/* Initialize a Monte Carlo board struct from the global board. */
static void
mc_init_board_from_global_board(struct mc_board *mc)
{
    int stones[BOARDMAX];
    int num_stones;
    int pos;
    int k;
    int r;

    memcpy(mc->board, board, sizeof(mc->board));
    mc->board_ko_pos = board_ko_pos;
    mc->hash = board_hash;
    memset(mc->queue, 0, sizeof(mc->queue));
    mc->queue[0] = 1;

    memset(mc->next_stone, 0, sizeof(mc->next_stone));
    for (pos = BOARDMIN; pos < BOARDMAX; pos++) {
        int geometry = ((mc->board[SE(pos)] << 14)
            | (mc->board[NE(pos)] << 12)
            | (mc->board[NW(pos)] << 10)
            | (mc->board[SW(pos)] << 8)
            | (mc->board[EAST(pos)] << 6)
            | (mc->board[NORTH(pos)] << 4)
            | (mc->board[WEST(pos)] << 2)
            | mc->board[SOUTH(pos)]);
        mc->local_context[pos] = geometry;
        if (board[pos] == EMPTY) {
            int s;

```

```

int captured_black_stones = 0;
int captured_white_stones = 0;
if (is_self_atari(pos, WHITE))
    mc->local_context[pos] |= 1 << 20;
if (is_self_atari(pos, BLACK))
    mc->local_context[pos] |= 1 << 21;
if (is_suicide(pos, WHITE))
    mc->local_context[pos] |= 1 << 22;
if (is_suicide(pos, BLACK))
    mc->local_context[pos] |= 1 << 23;
for (s = 0; s < 4; s++) {
    if (board[pos + delta[s]] == BLACK
        && countlib(pos + delta[s]) == 1)
        captured_black_stones += countstones(pos + delta[s]);
    else if (board[pos + delta[s]] == WHITE
        && countlib(pos + delta[s]) == 1)
        captured_white_stones += countstones(pos + delta[s]);
}
if (captured_black_stones > 3)
    captured_black_stones = 3;
if (captured_white_stones > 3)
    captured_white_stones = 3;
mc->local_context[pos] |= captured_black_stones << 16;
mc->local_context[pos] |= captured_white_stones << 18;
}
if (IS_STONE(board[pos]) && mc->next_stone[pos] == 0) {
    num_stones = findstones(pos, BOARDMAX, stones);
    mc->first_liberty_edge[pos] = 0;
    for (r = 0; r < num_stones; r++) {
        mc->next_stone[stones[r]] = stones[(r + 1) % num_stones];
        mc->reference_stone[stones[r]] = pos;
        for (k = 0; k < 4; k++) {
            if (board[stones[r] + delta[k]] == EMPTY)
                mc_add_liberty_edge(mc, stones[r], stones[r] + delta[k],
                    (k + 2) % 4);
        }
    }
}
}}}

```

Koodiesimerkki 2: Funktio Gnugo-ohjelmasta.

Koodiesimerkissä funktio saa syötteenä laudan kokonaistilanteen, jota se alkaa käsitellä. Muuttujien alustuksen jälkeen funktio kopioi laudan tilanteen. Seuraavaksi funktio alkaa tarkastella lautatilannetta. Löydettyään sopivan paikan funktio pääättelee parhaan laitton ja myös sen, muuttaako vastustajan seuraava laitto kokonaistilannetta. Jos vastustajan seuraava laitto muuttaa tilanteen huonommaksi, funktio laskee Monte-Carlo-menetelmän avulla uuden paikan.

4.2 Fuego-ohjelma

Fuegon perusta on kahdessa aiemmassa ohjelmassa [21], Anders Kierulfin Smart Game Board -ohjelmassa [22;23] ja Martin Müllerin Explorer-ohjelmassa [24;25]. Smart game board on (workbench) pelejä pelaava ohjelma, jonka kehitys alkoi 1980-luvun puolivälissä. Explorer on goota pelaava ohjelma, joka on rakennettu Smart game board -ohjelman pohjalta. Sen historia ulottuu vuoteen 1988, jolloin ohjelmaa kutsuttiin nimellä Go Explorer. Fuegon viimeisin versio 1.1 julkaistiin marraskuussa 2013. Fuego pelaa 2 kyuun – 1 danin tasolla.

Koodiesimerkissä 3 Fuego-ohjelman ensimmäinen funktio arpoo satunnaisen paikan laudalta hyödyntäen toista funktiota.

```
SgPoint GoBook::LookupMove(const GoBoard& bd) const
{
    vector<SgPoint> moves = LookupAllMoves(bd);
    size_t nuMoves = moves.size();
    if (nuMoves == 0)
        return SG_NULLMOVE;
    SgPoint p = moves[rand() % nuMoves];
    return p;
}
vector<SgPoint> GoBook::LookupAllMoves(const GoBoard& bd) const
{
    vector<SgPoint> result;
    const GoBook::MapEntry* mapEntry = LookupEntry(bd);
    if (mapEntry == 0)
        return result;
    size_t id = mapEntry->m_id;
    SG_ASSERT(id < m_entries.size());
    const vector<SgPoint>& moves = m_entries[id].m_moves;
    const int rotation = mapEntry->m_rotation;
    const int size = mapEntry->m_size;
    for (vector<SgPoint>::const_iterator it = moves.begin();
         it != moves.end(); ++it)
    {
        SgPoint p = SgPointUtil::Rotate(rotation, *it, size);
        if (!bd.IsLegal(p))
        {
            // Should not happen with 64-bit hashes, but not impossible
            SgWarning() << "illegal_book_move_(hash_code_collision?)\n";
            result.clear();
            break;
        }
        result.push_back(p);
    }
    return result;
}
```

Koodiesimerkki 3: Kaksi funktiota Fuego-ohjelmasta.

Koodiesimerkin 3 toinen funktio tarkistaa, löytyykö laitto tai sen pelikuva pelikirjastosta, ja tarkistaa sen jälkeen, onko laitto laillinen.

4.3 Pachi-ohjelma

Pachi on avoimen koodin tekoäly (GPLv2), jonka taso on 1 danin luokkaa. Se pystyy hajautettuun laskentaan ja tukemaan pelin analysointia. Tekoäly on koodattu noin 17 000 rivillä C:tä, ja se on modulaarinen. Muotojen tunnistus perustuu 3×3 kokoisen alueen tarkasteluun. Botin ajankäyttö perustuu aikavälien käyttöön (käyttää eniten aikaa keskipeliin). Uusin versio ohjelmasta on julkaistu huhtikuussa 2015, ja se on versionumeroltaan 11.0. Pachin eri versiot on nimetty eri kuuluisien Goon pelaajien mukaan. Uusimman version nimi on Retsugen. Honinbo Retsugen (烈元) (1750–1808) oli Honinbo-huoneen kymmenes päämies [18].

Koodiesimerkissä 4 esitetään Pachin Monte-Carlo-funktion alustus. Funktio saa laudan tilanteen ja muuta olennaista informaatiota, kuten kiven värin ja aikatieta. Funktio tarkastaa alussa, paljonko aikaa on käytettävissä. Seuraavaksi funktio tarkistaa, pitääkö peli luovuttaa vai pitääkö sitä jatkaa. Tässä vaiheessa funktio pelaa pelin loppuun monta kertaa ja valitsee parhaan siirron aiemmin pelattujen pelien perusteella. Funktio tarkistaa valitun laitton mahdollisten erikoistilanteiden varalta, ja jos niitä ei ole, se pelaa laitton, joka on laskutoimituksen perusteella paras.

```
static coord_t *
montecarlo_genmove(struct engine *e, struct board *b,
struct time_info *ti, enum stone color, bool pass_all_alive)
{
    struct montecarlo *mc = e->data;

    if (ti->dim == TD_WALLTIME) {
        fprintf(stderr,
            "Warning: _TD_WALLTIME_time_mode_not_supported, _resetting_to_defaults.\n");
        ti->period = TT_NULL;
    }
    if (ti->period == TT_NULL) {
        ti->period = TT_MOVE;
        ti->dim = TD_GAMES;
        ti->len.games = MC_GAMES;
    }
    struct time_stop stop;
    time_stop_conditions(ti, b, 20, 40, 3.0, &stop);

    /* resign when the hope for win vanishes */
    coord_t top_coord = resign;
    floating_t top_ratio = mc->resign_ratio;

    /* We use [0] for pass. Normally, this is an inaccessible corner
    * of board margin. */
    struct move_stat moves[board_size2(b)];
```

```

memset(moves, 0, sizeof(moves));

int losses = 0;
int i, superko = 0, good_games = 0;
for (i = 0; i < stop.desired.plyouts; i++) {
    assert(!b->superko_violation);

    struct board b2;
    board_copy(&b2, b);

    coord_t coord;
    board_play_random(&b2, color, &coord, NULL, NULL);
    if (!is_pass(coord) && !group_at(&b2, coord)) {
        /* Multi-stone suicide. We play chinese rules,
         * so we can't consider this. (Note that we
         * unfortunately still consider this in plyouts.) */
        if (DEBUGL(4)) {
            fprintf(stderr,
                "SUICIDE_DETECTED_at_%d,%d:\n", coord_x(coord, b), coord_y(coord, b));
            board_print(b, stderr);
        }
        continue;
    }

    struct plyout_setup ps = { .gamelen = mc->gamelen };
    int result = play_random_game(&ps, &b2, color, NULL, NULL, mc->plyout);
    board_done_noalloc(&b2);

    if (result == 0) {
        /* Superko. We just ignore this plyout.
         * And play again. */
        if (unlikely(superko > 2 * stop.desired.plyouts)) {
            /* Uhh. Triple ko, or something? */
            if (MCDEBUGL(0))
                fprintf(stderr, "SUPERKO_LOOP...I_will_pass...Did_we_hit_triple_ko?\n");
            goto pass_wins;
        }
        /* This plyout didn't count; we should not
         * disadvantage moves that lead to a superko.
         * And it is supposed to be rare. */
        i--, superko++;
        continue;
    }

    if (MCDEBUGL(3))
        fprintf(stderr, "\tresult_for_other_player:_%d\n", result);

    int pos = is_pass(coord) ? 0 : coord;

    good_games++;
    moves[pos].games++;

    losses += result > 0;
    moves[pos].wins += 1 - (result > 0);

    if (unlikely(!losses && i == mc->loss_threshold)) {
        /* We played out many games and didn't lose once yet.
         * This game is over. */

```

```

        break;
    }
}

if (!good_games) {
    /* No moves to try??? */
    if (MCDEBUGL(0)) {
        fprintf(stderr, "OUT_OF_MOVES!_I_will_pass._But_how_did_this_happen?\n");
        board_print(b, stderr);
    } ...
}

```

Koodiesimerkki 4: Osittainen funktio Pachi-ohjelmasta.

Jos funktio häviää monia simuloituja pelejä eikä löydä sopivaa seuraava laittoa, se jättää yhden laitton väliin eli passaa.

5 Ohjelmien vertailua

Peluutin tutkimustani varten edellä esiteltyjä kolmea go-ohjelmaa keskenään ja niitä itseään vastaan kymmenen kertaa, kahdella eri tietokoneella. Molemmat tietokoneet käyttävät 64-bittistä Slackware Linux -käyttäjärjestelmää. Tietokoneet olivat kannettava tietokone *Ouroboros* ja pöytätietokone *Hiddenburg*. Kannettavassa tietokoneessa on 4 GB muistia ja Intelin i5 520M, 2,4 GHz -prosessori ja nVidia GT218M -näytönohjain. Pöytätietokoneessa on 8 GB muistia ja Intelin i5 670, 3,5 GHz -prosessori ja GeForce GTX 560 Ti -näytönohjain. Yksityiskohtaisemmat tiedot ovat taulukossa 2.

Taulukko 2: Käytettyjen tietokoneiden yksityiskohtaisemmat tiedot.

	Pöytätietokone	Kannettava tietokone
Nimi	Hiddenburg	Ouroboros
Prosessori	Intel i5 670	Intel i5 520M
Muistin määrä	7957 MiB	3803 MiB
Linux-ytimen versio	4.1.6	4.1.6
Näytönohjain	GeForce GTX 560 Ti	GT218M [NVS 3100M]
Ohjaimen muisti	2047 MiB	512 MiB
Ohjaimen ajuri	nVidia 352.41	nv 2.1.20

Taulukossa 3 esitetään pelien tulokset eli voitot ja häviöt sekä mustilla että valkoisilla kivillä. Kuten ohjelmaa peluutettiin kymmenen kertaa toisiaan vastaan värejä vaihdellen ja kymmenen kertaa itseään vastaan.

Tulokset tarkastettiin pelien jälkeen mahdollisten virheiden varalta. Sekä piste- että luovutusvoitoissa esiintyikin virheitä. Niistä kerrotaan tarkemmin luvussa 6 Kehityskohteet.

Taulukko 3: Kaikki pelitulokset. 1 on voitto, 0 on häviö; *w* on valkea, *b* on musta.

värit		b	w	b	w	b	w	b	w	b	w	
		w	b	w	b	w	b	w	b	w	b	
pelit	kerrat	1	2	3	4	5	6	7	8	9	10	
1	gnugo	0	0	0	0	0	1	0	0	0	1	2
	pachi	1	1	1	1	1	0	1	1	1	0	8
2	gnugo	0	1	0	1	0	1	0	0	0	1	4
	fuego	1	0	1	0	1	0	1	1	1	0	6
3	pachi	0	1	1	0	1	1	1	0	1	0	6
	fuego	1	0	0	1	0	0	0	1	0	1	4
4	gnugo 1	0	1	0	1	1	0	1	0	1	0	5
	gnugo 2	1	0	1	0	0	1	0	1	0	1	5
5	fuego 1	1	1	1	0	1	0	1	1	1	1	8
	fuego 2	0	0	0	1	0	1	0	0	0	0	2
6	pachi 1	1	0	0	1	0	0	0	0	1	1	4
	pachi 2	0	1	1	0	1	1	1	1	0	0	6

5.1 Gnugo ja Pachi

Gnugota ja Pachia peluutettiin kumpaakin omilla asetuksillaan. Gnugo pelasi asetuksilla `--mode=gtp --japanese-rules` ja Pachi asetuksilla `-r japanese`. Gnugoon ja Pachin välisestä kymmenestä pelistä Pachi voitti kahdeksan ja Gnugo kaksi. Puolet peleistä pelattiin niin, että Gnugo pelasi mustilla ja Pachi valkoisilla ja puolet päinvastoin. Pachi voitti kaikki viisi mustilla pelaamaansa peliä ja kolme peliä valkoisilla. Gnugo voitti molemmat pelinsä valkoisilla. Toisen pelin Gnugo voitti luovutuksella ja toisen puolella pisteellä. Pachi voitti yhden pelin luovutuksella ja muut 1,5–33,5 pisteen välisellä erolla. Peleissä pelattiin keskimäärin 223,3 laittaa. Pisin peli oli viides, jossa oli 306 laittaa ja lyhin peli oli seitsemäs, jossa oli 136 laittaa. Molemmat pelit voitti Pachi. Pelit kestivät keskimäärin 23 min ja 10,5 s.

Taulukko 4: Gnugo – Pachi tulokset. 1 on voitto, 0 on häviö; *w* on valkea, *b* on musta.

värit	b	w	b	w	b	w	b	w	b	w
	w	b	w	b	w	b	w	b	w	b
pelikerrat	1	2	3	4	5	6	7	8	9	10
gnugo	0	0	0	0	0	1	0	0	0	1
pachi	1	1	1	1	1	0	1	1	1	0

5.2 Gnugo ja Fuego

Gnugota peluutettiin Fuegoa ja samoilla asetuksilla kuin Pachiakin vastaan. Fuegoa käytettiin komennolla `fuego --config fuego.conf`. Asetustiedoston sisältö oli koodiesimerkin 5 mukainen.

```
go_param_rules capture_dead 0
go_param_rules two_passes_end_game 1
go_param_rules ko_rule pos_superko
go_param_rules allow_suicide 0
go_param_rules japanese_scoring 1
uct_max_memory 6000000
sg_param time_mode real
```

Koodiesimerkki 5: Fuegon asetustiedosto `fuego.conf`.

Fuegon asetuksissa määritellään, millä säännöillä ohjelma pelaa, paljonko muistia ohjelma enintään varaa ja mitkä ovat ohjelman aika-asetukset. Sen lisäksi Fuego käyttää omaa aloituskirjastoaan.

Myös nämä ohjelmat pelasivat kymmenen peliä, joista puolet valkoisilla ja puolet mustilla kivillä. Fuego voitti pelaamistaan peleistä kahdeksan, kaikki valkoisilla kivillä pelaamansa ja yhden mustilla kivillä pelaamansa. Gnugo voitti neljä valkoisilla pelaamaansa peliä, kaksi luovutuksella ja kaksi pisteillä. Peleissä pelattiin keskimäärin 250,4 laittoa. Pisin peli oli neljäs, ja siinä oli 361 laittoa, ja sen voitti Gnugo. Lyhin peli oli yhdeksäs, ja siinä oli 146 laittoa, ja sen voitti Fuego. Kymmenestä pelistä kaiken kaikkiaan neljä päättyi luovutusvoittoon ja loput kuusi 1,5–66,5 pisteen erolla. Pelit kestivät keskimäärin 17 min ja 57 s.

Taulukko 5: Gnugo – Fuego -tulokset. 1 on voitto, 0 on häviö; *w* on valkea, *b* on musta.

värit	b	w	b	w	b	w	b	w	b	w
	w	b	w	b	w	b	w	b	w	b
pelikerrat	1	2	3	4	5	6	7	8	9	10
gnugo	0	1	0	1	0	1	0	0	0	1
fuego	1	0	1	0	1	0	1	1	1	0

5.3 Pachi ja Fuego

Pachi ja Fuego pelasivat kymmenen peliä samoilla asetuksilla kuin aikaisemmissa peleissä, Pachi -r_japanese -asetuksilla ja Fuego asetustiedoston asetuksilla. Lisäksi Fuego käytti omaa aloituskirjastoaan.

Peleistä Pachi voitti kuusi ja Fuego neljä. Fuego voitti kaikki neljä peliään mustilla ja Pachi kaikki valkoisilla pelaamansa viisi peliä ja yhden mustilla pelaamansa. Kaikki kymmenen peliä olivat luovutusvoittoa. Keskimäärin peleissä oli 231,2 laittoa. Suurin laittojen määrä oli kuudennessa pelissä, jossa oli 291 laittoa ja sen voitti Pachi. Pienin laittojen määrä oli toisessa pelissä eli 191, ja myös sen voitti Pachi. Pelit kestivät keskimäärin 35 min ja 49,6 s.

Taulukko 6: Pachi – Fuego -tulokset. 1 on voitto, 0 on häviö; *w* on valkea, *b* on musta.

värit	b	w	b	w	b	w	b	w	b	w
	w	b	w	b	w	b	w	b	w	b
pelikerrat	1	2	3	4	5	6	7	8	9	10
pachi	0	1	1	0	1	1	1	0	1	0
fuego	1	0	0	1	0	0	0	1	0	1

5.4 Gnugo ja Gnugo

Gnugo 1- ja Gnugo 2 -ohjelmat, joilla oli samat asetukset, `--mode=gtp --japanese-rules`, voittivat molemmat viisi peliä. Gnugo 1 voitti kaksi peliä valkoisilla ja kolme mustilla. Gnugo 2 voitti kolme peliä valkoisilla ja kaksi mustilla. Molemmat voittivat mustilla pelatessaan yhden pelin luovusvoitolla. Gnugo-Gnugo pelien laittojen keskiarvo oli 195,3. Gnugo 1 voitti pelin isoimmalla piste-erolla 38,5, ja pienin pistevoitto kirjattiin Gnugo 2:lle 3,5. pistettä. Eniten laittoja oli seitsemännessä pelissä eli 241 ja vähiten yhdeksännessä pelissä eli 187. Pelit kestivät keskimäärin 2 min ja 55 s.

Taulukko 7: Gnugo – Gnugo tulokset. 1 on voitto, 0 on häviö; *w* on valkea, *b* on musta.

värit	b	w	b	w	b	w	b	w	b	w
	w	b	w	b	w	b	w	b	w	b
pelikerrat	1	2	3	4	5	6	7	8	9	10
gnugo 1	0	1	0	1	1	0	1	0	1	0
gnugo 2	1	0	1	0	0	1	0	1	0	1

5.5 Fuego ja Fuego

Fuego 1 ja Fuego 2 pelasivat vastakkain eri asetuksilla kymmenen peliä. Fuego 1 pelasi koodiesimerkissä 6 olevilla asetuksilla.

```
uct_param_search number_threads 4
uct_param_search lock_free 1
uct_max_memory 60000000
uct_param_player reuse_subtree 1
uct_param_player ponder 1
```

Koodiesimerkki 6: Fuegon asetustiedosto fuegox.conf.

Fuego 2 pelasi vuorostaan koodiesimerkistä 5 löytyvillä asetuksilla (sivu 18).

Fuego 1 voitti kahdeksan peliä, joista viisi valkoisilla ja kolme mustilla. Fuego 2 voitti kaksi peliä, molemmat mustilla. Kaikki pelit olivat luovutusvoittoa. Keskimäärin pelattiin 249,4 laittaa per peli. Pisin peli oli kymmenes, jossa oli 293 laittaa ja jonka voitti Fuego 1. Lyhin peli oli kuudes, jossa oli 210 laittaa ja sen voitti Fuego 2. Pelit kestivät keskimäärin 22 min ja 40 s.

Taulukko 8: Fuego – Fuego -tulokset. 1 on voitto, 0 on häviö; *w* on valkea, *b* on musta.

värit	b	w	b	w	b	w	b	w	b	w
	w	b	w	b	w	b	w	b	w	b
pelikerrat	1	2	3	4	5	6	7	8	9	10
fuego 1	1	1	1	0	1	0	1	1	1	1
fuego 2	0	0	0	1	0	1	0	0	0	0

5.6 Pachi ja Pachi

Pachi 1 ja Pachi 2 pelasivat vastakkain eri asetuksilla kymmenen peliä. Pachi 1 pelasi käyttäen Fuegon aloituskirjastoa, komennolla `pachi -r japanese -f /usr/share/fuego/book.dat`. Pachi 2 pelasi oletusasetuksilla komennolla `pachi -r japanese`. Peleistä Pachi 1 voitti viisi ja Pachi 2 samoin viisi peliä. Pachi 1 voitti kolme peliä mustilla ja kaksi valkoisilla ja Pachi 2 päin vastoin. Laittojen keskiarvo oli 228,6. Pisin peli oli yhdeksäs, ja siinä 301 laittoa ja sen voitti Pachi 1. Lyhin peli oli viides, jossa oli 178 laittoa ja sen voitti Pachi 2. Pelit kestivät keskimäärin 36 min ja 53 s.

Taulukko 9: Pachi – Pachi -tulokset. 1 on voitto, 0 on häviö; *w* on valkea, *b* on musta.

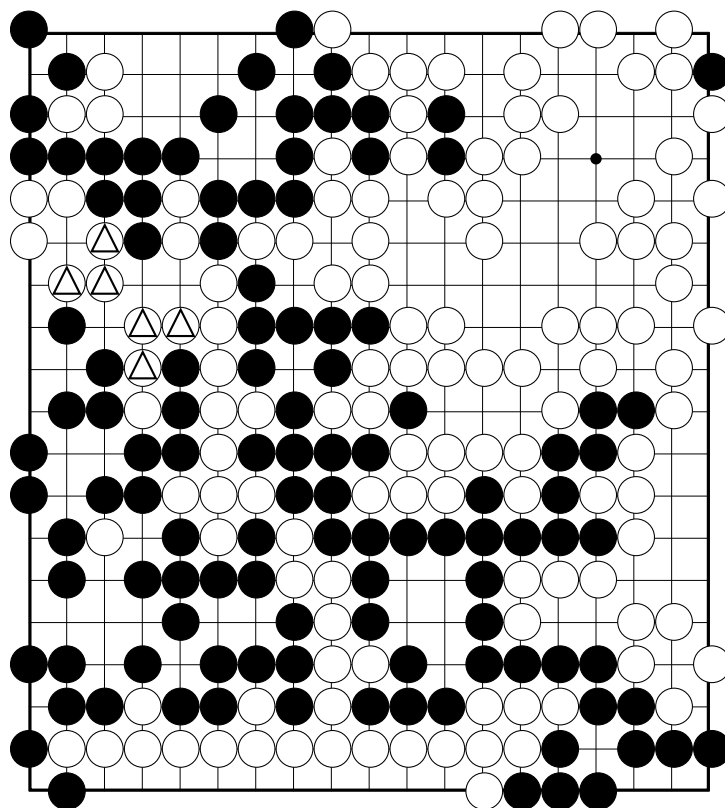
värit	b	w	b	w	b	w	b	w	b	w
	w	b	w	b	w	b	w	b	w	b
pelikerrat	1	2	3	4	5	6	7	8	9	10
pachi 1	1	0	0	1	0	0	0	0	1	1
pachi 2	0	1	1	0	1	1	1	1	0	0

6 Kehityskohteet

Tutkittujen Go-ohjelmien ongelmana on, että ne eivät osaa laskea pelissä rajattuja alueita oikein. Tämän vuoksi ihmisen pitää tarkistaa pelin lopputulos. Ohjelman laskeman ja ihmisen laskeman pelituloksen ero voi olla kymmeniä pisteitä.

Pelastusta 60 pelistä 33 päättyi pisteidenlaskuun ja loput 27 luovutukseen. Pisteidenlaskuun päättyneistä peleistä 10:ssä vaihtui voittaja, koska ohjelmat laskivat tuloksen väärin. Osassa peleistä pisteero oli niukka, osassa suurempi. Myös osassa luovutusvoittoja johdossa ollut ohjelma luovutti pelin. Näin ollen ohjelmien pistelaskua tulisi kehittää tarkemmaksi. Tarkimmin pisteet laski Gnugo-ohjelma.

Kuvassa 9 on esimerkki pelistä, jossa tulos oli luovutusvoitto valkealle, vaikka jos peliä olisi jatkettu, voitto olisi mennyt mustalle. Näin ollen näyttää todennäköiseltä, että Fuego-ohjelma laski kolmioilla merkityn ryhmän statuksen väärin.



Musta: Fuego
Valkoinen: Pachi
Tulos: W+R

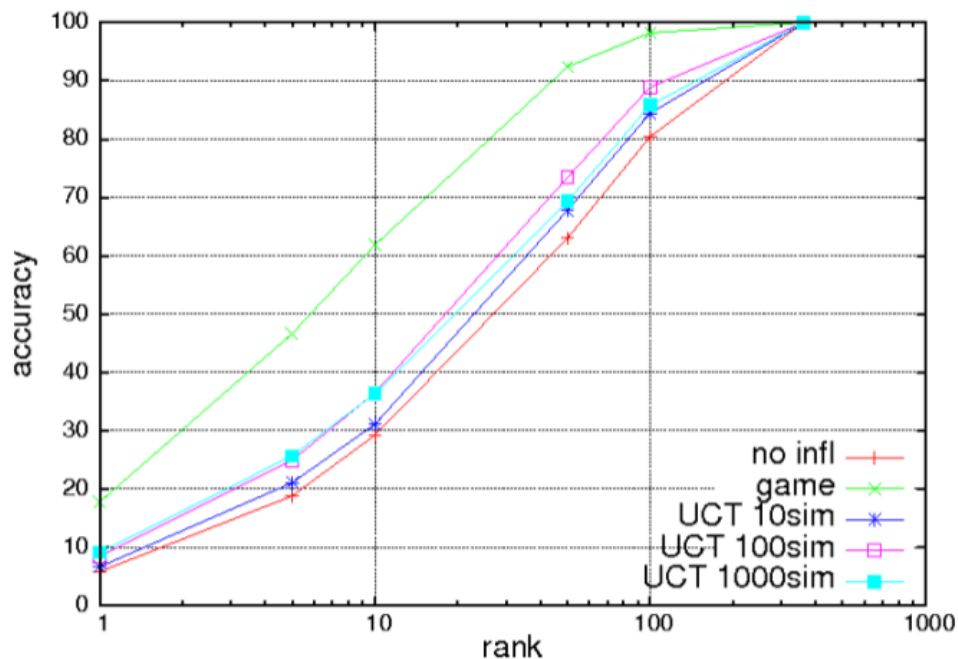
Vangit: Musta 20, Valkea 34
Komi: 6,5 pistettä

Kuva 9: Esimerkkipeli Fuego vastaan Pachi, kuudes peli.

Kiinnostavia tuloksia on saatu yhdistämällä konvoluutionaaliset syväoppivat neuroverkot ja UCT-algoritmi [26]. Neuroverkot ovat itseoppivia ihmisaivojen toimintaa ja matemaattista logiikkaa yhdistelevä laskentamalli. Konvoluutio tarkoittaa kahden signaalin yhdistelmästä syntyvää uutta signaalia. Sellaisia ovat esimerkiksi näkö- ja kuulohavainnon yhdistäminen sekä Goohon liittyen pelilaudan senhetkisen pelitilanteen yhdistäminen aiempiin, jo pelattuihin peleihin.

Kuvassa 10 näkyy konvoluutionaalisen syväoppivan neuroverkon tarkkuus ilman harjoitusta verrattuna siihen, mikä tarkkuus on 10, 100 ja 1000 harjoituskerran jälkeen. Lisäksi neuroverkon suoritusta verrataan parhaaseen mahdolliseen eli ideaaliin peliin.

Kuvassa 10 vihreä käyrä edustaa ideaalitulannetta, tummansininen neuroverkon peliä kymmenen harjoituskerran jälkeen, vaaleanpunainen 100 harjoituskerran jälkeen ja vaaleansininen 1000 harjoituskerran jälkeen. Punainen väri kuvaa sitä, miten neuroverkko pelaa ilman, että se on harjoitellut pelaamista etukäteen. Pystyakseli kuvaa laittotarkkuutta ja vaaka-akseli sitä, kuinka lähelle ideaalilaittoa on päästy. Kun neuroverkko pelasi ilman harjoitusta, se pelasi joka kolmannen laitton oikein. 1000 harjoituskerran jälkeen laittojen tarkkuus parani 29 %:sta 37 %:iin.



Kuva 10: Neuroverkon tarkkuus löytää kymmenen parasta laittoa 10, 100 ja 1000 simulaation jälkeen [26].

7 Yhteenveto

Insinööriyössä käsiteltiin aluksi Go-pelin historiaa ja Monte-Carlo-menetelmää. Sitten kerrottiin yleisesti Monte-Carlo-menetelmän parannelluista versiosta eli Monte-Carlo-puuhausta (MCTS:stä), yläraja-arvon funktiosta (UCT:stä) ja RAVE-algoritmista. Insinööriyöprojektissa etsittiin ensin kolme yleistä avoimen lähdekoodin Go-tietokoneohjelmaa internetistä. Tarkoituksenani oli etsiä myös neljäs yleinen ohjelma, mutta siihen ei löytynyt lähdekoodia. Tämän jälkeen asennettiin valitut tietokoneohjelmat kahdelle eri tietokoneelle ja tutkittiin, miten ne saadaan pelaamaan toisiaan vastaan. Koska kaikki ohjelmat tukivat samaa protokollaa, niitä pystyttiin peluuttamaan keskenään. Ohjelmissa oli mukana ohjeet, joista saatiin selvitettyä, mitä komentoja ohjelmat vaativat voidakseen pelata monta peliä peräkkäin. Näin saatiin tulokset nopeammin selville. Kolme tutkittavaa tietokoneohjelmaa olivat Gnugo, Fuego ja Pachi. Kaikki ohjelmat tukivat samaa protokollaa, ja siten niitä pystyttiin peluuttamaan keskenään. Ohjelmien pelaamat pelit analysoitiin, ja niiden tulokset tarkastettiin manuaalisesti. Lopuksi esiteltiin ohjelmien mahdollisia kehityskohteita.

Tulevaisuudessa tulisi Go-ohjelmien ohjelmoinnissa kiinnittää enemmän huomiota siihen, etteivät ohjelmat tekisi väärää tulkintoja pelitilanteista, mikä johtaa pahimmillaan siihen, että Go-ohjelma antaa luovutusvoiton, vaikka peliä jatkamalla Go-ohjelma voittaisi pelin. Myös pelin loppupisteiden laskua tulisi parantaa.

Eniten vaikeuksia tutkimuksen aikana tuotti ohjelmien saaminen pelaamaan keskenään, varsinkin joidenkin ohjelmien lähdekoodien löytäminen oli hankalaa. Siksi jätin pois alun perin mukana aikomani neljännen ohjelman. Myös lähdeaineiston määrä yllätti, ja oli vaikeaa löytää relevanttia tietoa aineistosta.

Uusia kiinnostavia jatkotutkimuskohteita Go-ohjelmien parantamisessa voisivat olla neuroverkot.

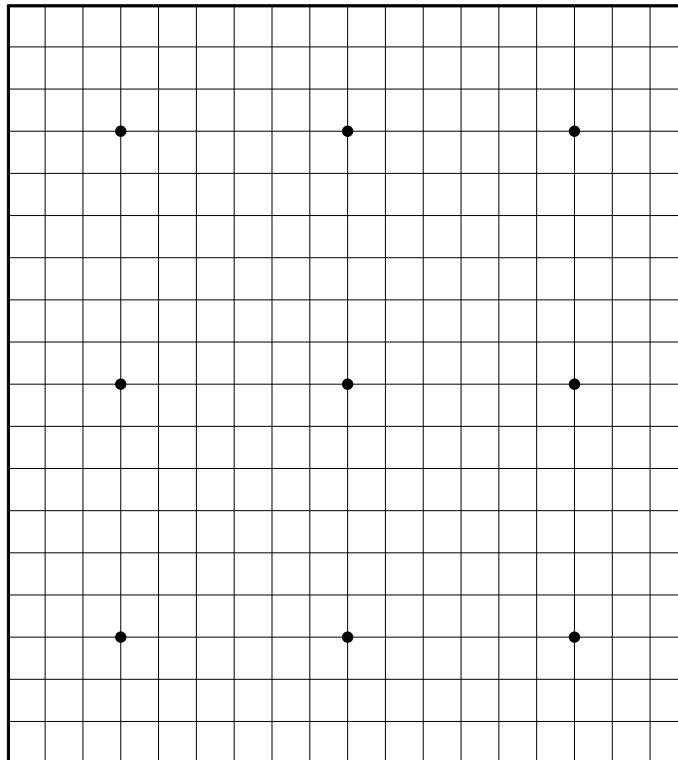
Lähteet

- 1 Levinovitz A. 2014; The Mystery of Go, the Ancient Game That Computers Still Can't Win. Verkkodokumentti. Wired.
<<http://www.wired.com/2014/05/the-world-of-computer-go/>>. Luettu 25.11.2015.
- 2 Go Rules comparison. 2014. Verkkodokumentti. online-go.com.
<<http://online-go.com/docs/go-rules-comparison-matrix>>. Luettu 25.11.2015.
- 3 Avoin lähdekoodi. Verkkodokumentti. coss.fi.
<<http://coss.fi/avoimuus/avoin-lahdekoodi>>. Luettu 25.11.2015.
- 4 Fairbairn J. 1995. Go in Ancient China. gobase.org.
<<http://gobase.org/reading/history/china>>. Luettu 25.11.2015.
- 5 Shotwell P. 2008. Speculations on its Origins and Symbolism in Ancient China. gobase.org. <http://www.usgo.org/files/bh_library/originsofgo.pdf>. Luettu 25.11.2015.
- 6 Millen JK. 1981. Programming the Game of Go. BYTE the small system journal. 04;6: s. 102–120. <https://archive.org/stream/byte-magazine-1981-04/1981_04_BYTE_06-04_Future_Computers#page/n101>. Luettu 25.11.2015.
- 7 Fotland D. 2013. World Computer Go Championships. Verkkodokumentti. smart-games.com. <<http://www.smart-games.com/worldcompgo.html>>. Luettu 25.11.2015.
- 8 Hollosi A. 2006. The official specification of the SGF standard. <<http://www.red-bean.com/sgf/>>. Luettu 25.11.2015.
- 9 Sensei's Library. <<http://senseis.xmp.net/?GoServers>>. Luettu 25.11.2015.
- 10 Eckhard R. 1987. Stan Ulam, John Von Neumann and the Monte Carlo Method. Los Alamos Science Special Issue. s. 131–141.
- 11 Rules for Canfield Solitaire. <<http://politaire.com/help/canfield>>. Luettu 25.11.2015.
- 12 American Institute of Physics. Los Alamos National Laboratory; 1981.
<<https://www.aip.org/history/acap/institutions/inst.jsp?lanl>>. Luettu 25.11.2015.
- 13 Prokop J. 2013. Waltheri database of professional game records.
<<http://ps.waltheri.net/>>. Luettu 25.11.2015.
- 14 Holmström M. 2010. Monte Carlo Go. Pro gradu -tutkielma. Helsingin yliopisto.
- 15 Chaslot GMJB, Winands MHM, Jaap Van Den Herik H, Uiterwijk JWJM, Bouzy B. 2008. Progressive Strategies For Monte-Carlo Tree Search. New Mathematics and Natural Computation. 04(03): s. 343–357.
<<http://www.worldscientific.com/doi/abs/10.1142/S1793005708001094>>. Luettu 25.11.2015.

- 16 Kocsis L, Szepesvári C. 2006. Bandit based Monte-Carlo Planning. ECML-06. Number 4212 in LNCS. Springer; s. 282–293.
- 17 Gelly S, Silver D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*. 175(11): s. 1856–1875.
- 18 Honinbo Retsugen (烈元). Sensei's Library; Verkkodokumentti. <<http://senseis.xmp.net/?Retsugen>>. Luettu 25.11.2015.
- 19 Gelly S, Wang Y, Munos R, Teytaud O. 2006. Modification of UCT with Patterns in Monte-Carlo Go. <<http://hal.inria.fr/docs/00/12/15/16/PDF/RR-6062.pdf>>. Luettu 25.11.2015.
- 20 Gnugo Manual: sec 25 Other Clients. Verkkodokumentti gnu.org. <http://www.gnu.org/software/gnugo/gnugo_3.html#SEC25>. Luettu 25.11.2015.
- 21 Enzenberger M, Müller M, Arneson B, Segal R. 2010. FUEGO – An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games*. 2: s. 259–270. <<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5599855>>. Luettu 25.11.2015.
- 22 Kierulf A. 1990. Smart Game Board: A Workbench for Game-playing Programs, with Go and Othello as Case Studies. Dissertationen. ETH Zürich. Institut of technology Zürich. <<http://books.google.co.kr/books?id=2aurPQAACAAJ>>. Luettu 25.11.2015.
- 23 Kierulf A, Gasser R, Geiser PM, Müller M, Nievergelt J, Wirth C. 1991. Every interactive system evolves into hyper-space: The case of the Smart Game Board. *Hypertext/Hypermedia*. 276: s. 174–180.
- 24 Müller M. 1995. Computer Go as a sum of local games: An application of combinatorial game theory. Ph.D. dissertation. Dept. Comput. Sci.,ETH Zürich. Zürich, Switzerland.
- 25 Müller M. 2002. Computer Go. *Artificial Intelligence*. 01;134: s. 145–179. <[http://dx.doi.org/10.1016/S0004-3702\(01\)00121-7](http://dx.doi.org/10.1016/S0004-3702(01)00121-7)>. Luettu 25.11.2015.
- 26 Paisarnsrisomsuk S, Wiratchotisan P. 2015. UCT-Enhanced Deep Convolutional Neural Network for Move Recommendation in Go. A Major Qualifying Project Report. Worcester Polytechnic Institute.
- 27 陈祖源. The History of Go Rules. 2011. <https://www.usgo.org/files/bh_library/historyofgorules.pdf>. Luettu 25.11.2015.
- 28 日本棋院 日本囲碁規約 (全文) Go Rules 1989. Nihon Ki-in 日本棋院. <<http://www.nihonkiin.or.jp/joho/kiyaku/zenbun.htm>>. Japaniksi. Luettu 25.11.2015. Käännös englanniksi Davies J. <<http://www.cs.cmu.edu/~wjh/go/rules/Japanese.html>>. Luettu 25.11.2015.

1 Mitä Go on?

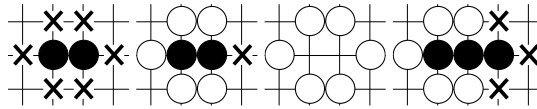
Go on vanha aasialainen kaksinpeli, joka tunnetaan lännessä japanin vaikutuksesta nimellä go (jap. 碁). Peli tunnetaan myös muilla nimillä: japaniksi 囲碁 (igo), koreaksi 바둑 (baduk) ja kiinaksi 圍棋 (weiqi). Pelin historia ulottuu yli 2500 vuotta taaksepäin. Vanhin maininta Goosta on Kungfutsen (551–479 eaa.) ja hänen seuraajansa Mengzin (372–289 eaa.) [4] kirjoituksista. Vanhimmat maininnat Goon säännöistä löytyvät Dunhuang Qijing 敦煌写本【碁经】 (käsinkirjoitetut Dunhuang go klassikot) kääröstä, Pohjoinen Zhou -dynastia (557–581 jaa.) [27].



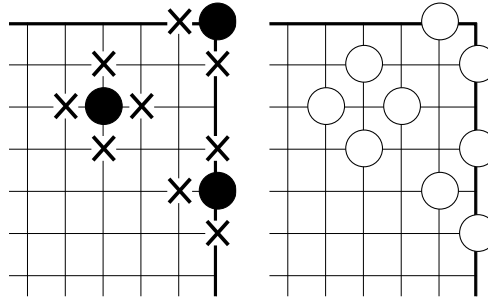
Hoshit (星) eli tähtipisteet merkitsevät mahdollisten tasoituskivien paikan.

Kuva 1: Täysikokoinen Go-lauta.

Goota pelataan 19 pysty- ja vaakaviivan rajaamalla alueella (ks. kuva 1), muut yleiset pelilaudan koot ovat 13×13 sekä 9×9 . Pelin tarkoituksena on saartaa mahdollisimman paljon aluetta omilla kivilään, ja enemmän aluetta saartanut voittaa. Pelissä voi myös vangita vastustajan kiviä viemällä niiltä vapaudet (yhellä kivellä voi olla neljä vapautta). Kun vapaudet viedään kivet poistetaan laudalta. (Ks. esimerkkikuvat 2 ja 3.)

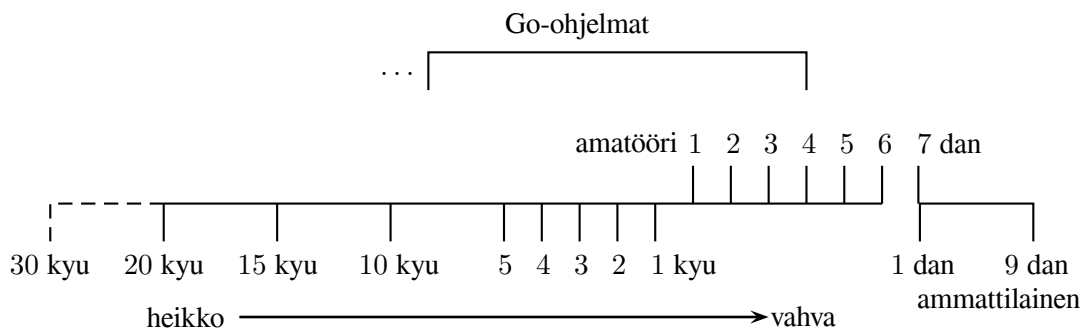


Kuva 2: Kiviä syödään. × mustan vapaus.



Kuva 3: Pelilaudan reunalla kivillä on vähemmän vapauksia.

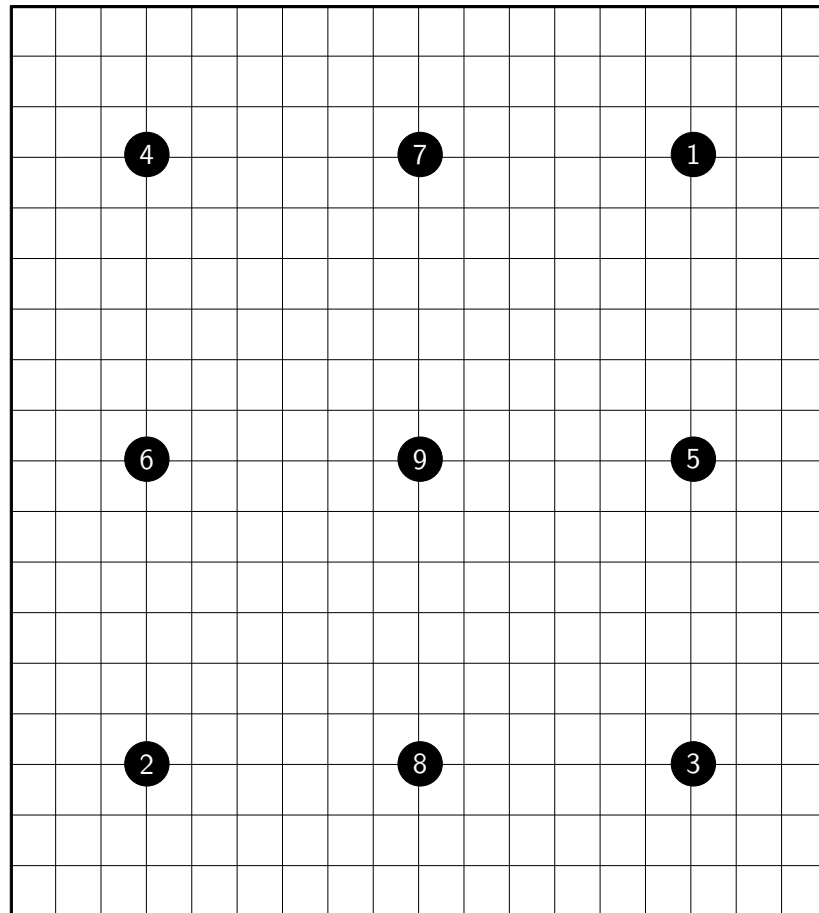
Koska Goossa musta aloittaa, saa valkea korvaukseksi komin (jap. ㊦ ≡ komi), jonka arvo on yleensä 6,5 pistettä. Käänteistä komia käytettäessä musta sekä saa pisteet että aloittaa pelin.



Kuva 4: Go-luokitukset.

Goossa on käytössä (myös itsepuolustuslajeihin siirtynyt) tasojärjestelmä, jossa kyu (級) tarkoittaa aloittelijaa ja dan (段) mestaria. Kyuu-taso alkaa 30:stä ja nousee 1:een, minkä jälkeen alkaa dan-taso, jossa amatöörit käyttävät 1–7:ä dania ja ammattilaiset 1–9:ä dania (amatööri 7 dan vastaa ammattilaisten shodania eli ammattilais 1 -dania).

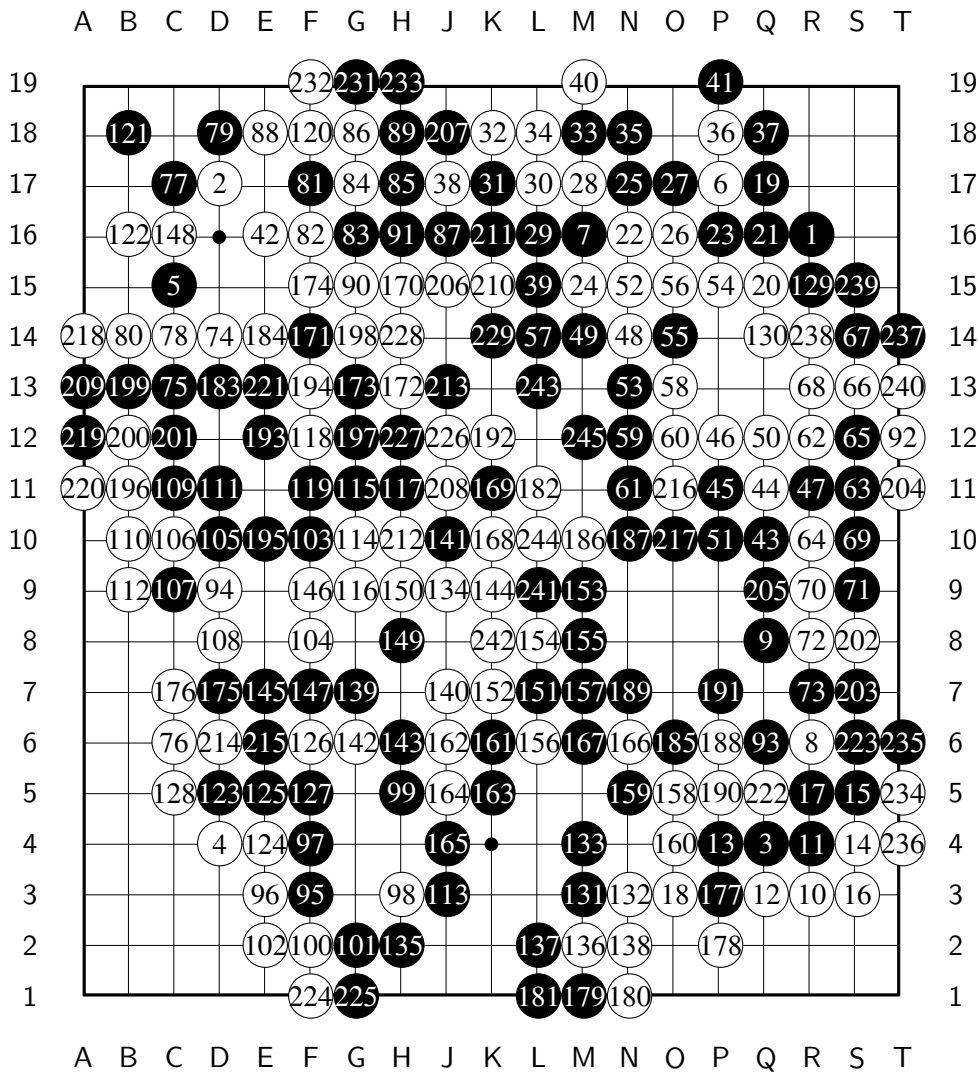
Goossa on mahdollista käyttää myös tasoituskiviä, jos pelaajilla on tasoeroa. Yleensä tasoituskiviä on enintään yhdeksän. Kuvassa 5 näkyy tasoituskivien paikat 9:llä kivellä pelattaessa. Pelattaessa 5 tai 7 kivellä viimeinen tasoituskivi pistetään laudan keskelle.



On hyvän tavan mukaista asettaa tasoituskivet juuri tässä järjestyksessä.

Kuva 5: Tasoituskivien paikat 9:llä kivellä mustan näkökulmasta.

Tasoituspeleissä ei yleensä ole käytössä komia eli komin arvo on 0 pistettä, mutta jos halutaan välttää tasapelejä, annetaan tasoituspelissä $\frac{1}{2}$ pisteen komi valkealle.



19
18 **Musta:** Yamashita Keigo, Kisei
山下 敬吾棋聖
17 **Valkoinen:** Iyama Yuuta, 9 dan
井山 裕太九段
16 **Tulos:** Iyamalle ½ pisteellä
15
14 **Alueet:** Musta 64, Valkoinen 66
13 **Vangit:**
12 Mustalla 7 + 11 kuollutta valkeaa kiveä
Valkealla 3 + 7 kuollutta mustaa kiveä
11 **Komi:** 6,5 pistettä
10
9 Valkea 230 @ 171
8
7
6
5
4
3
2
1

Kuva 6: Esimerkkipeli, 38:nen Kisei-turnauksen 1. päätösottelu, Yamashita vastaan Iyama.

2 Goon viralliset säännöt

Näitä sääntöjä [28, japanilaiset säännöt '89] tulee noudattaa yhteisymmärryksessä sekä hyvässä ja rakentavassa hengessä.

Sääntö 1. Go: Go on kahden hengen peli, jossa mitellään taitoja pelilaudalla ja jossa tarkoituksena on saartaa laudalta vastustajaa enemmän aluetta.

Sääntö 2. Peli: Pelataan vuorotellen, yksi pelaa mustilla kivillä ja toinen valkeilla kivillä.

Sääntö 3. Pelin kulku: Goota pelataan 19 pysty- ja vaakaviivan rajaamalla pelilaudalla. Pelaaja laittaa kiven tyhjään risteyskohtaan säännön 4 mukaan, minkä jälkeen on toisen pelaajan vuoro.

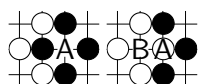
I Määritelmät ("risteys", "viereinen", "ketju", "vapaus"):

- Piste, jossa vaakaviiva kohtaa pystyviivan kutsutaan risteykseksi.
- Risteykset ovat vierekkäiset, jos ne ovat vierekkäin vaaka- tai pystysuunnassa eikä mitään ole niiden välissä.
- Vierekkäin olevia samanvärisiä kiviä kutsutaan ketjuksi.
- Kiven vieressä olevaa tyhjää risteystä kutsutaan vapaudeksi.

Sääntö 4. Pelaaminen: Jotta laitto olisi laillinen, omilla kivillä pitää olla laitton jälkeen vähintään yksi (1) vapaus. Vapaudettomia kiviä ei saa olla pelilaudalla. Poikkeuksena vangitseminen, katso sääntö 5.

Sääntö 5. Vangitseminen: Jos kiven asettamisen jälkeen vastustajan kivellä tai kiviketjulla ei ole vapauksia, pelaaja poistaa sen/ne laudalta. Poistettuja kiviä kutsutaan vangeiksi.

Sääntö 6. Ko: Tilannetta, jossa pelaajat voivat vuorotellen vangita ja uudelleen vangita vastustajan kiven, kutsutaan nimellä ko (jap. 劫). Pelaaja, jonka kivi vangitaan koossa, ei voi seuraavalla vuorolla uudelleen vangita vastustajan kiveä kyseisessä koossa.



Koska mustan kivi vangitaan pelaamalla A, musta ei voi koon vuoksi pelata B:hen, vaan hänen on ensin pelattava muualle.

Sääntö 7. Elämä ja kuolema:

I Kivien sanotaan olevan elossa, jos vastustaja ei voi vangita niitä tai jos niiden vangitseminen mahdollistaa sellaisten uusien kivien pelaamisen, joita vastustaja ei voi vangita. Kivet, jotka eivät ole elossa, ovat kuolleita.

II Kun peli pysähtyy säännön 9 mukaan, uudelleen saman koon vangitseminen on kielletty. Pelaaja, jonka kivi vangittiin koossa, voi kuitenkin, pelata koohon uudestaan, kunhan on kerran passannut kyseiseen kooheen.

Sääntö 8. Alueet: Aluetta ovat sellaiset laudan kohdat, joita ympäröivät keskenään samanväriset, elävät kivet. Tyhjät risteykset, jotka eivät ole kummankaan pelaajan alueella, ovat dameja (jap. 駄目 dame). Jos kiviketju koskettaa damea, sen sanotaan olevan sekissä (jap. セキ seki), eikä kiviketju risteyskohtaa ympäröidessäänkään tuota silloin pisteitä. Yksi alueella oleva tyhjä risteys vastaa yhtä pistettä.

Sääntö 9. Pelin päättäminen:

- I Kun pelaaja ja vastapelaaja passaavat peräkkäin, peli pysäytetään.
- II Pelin ollessa pysähtynyt pelaajat yhdessä tuumin sopivat elävistä ja kuolleista kivistä ja alueista. Tätä kutsutaan pelin päättymiseksi.
- III Jos pelaaja pyytää pysäytetyn pelin jatkamista epäselvässä tilanteessa, vastustajan on suostuttava tähän ja vastustajalla on oikeus pelata ensin
(*huomautus* myös passaus on laitto).

Sääntö 10. Pelin tulos:

- I Kun on sovittu pelin päättymisestä, molemmat pelaajat poistavat kuolleet (vastustajan) kivet alueiltaan ja lisäävät nämä vankeihinsa.
- II Vangeilla täytetään vastustajan aluetta ja lopuksi alueet lasketaan. Pelaaja, jolla on enemmän aluetta, voittaa. Jos alueita on saman verran, tuloksena on tasapeli, jota kutsutaan nimellä jigo (jap. 持碁).
- III Jos toinen pelaaja esittää vastalauseen pelin tuloksesta, pelaajien täytyy uudelleen vahvistaa pelin tulos, esimerkiksi rekonstruoida peli uudelleen.
- IV Kun molemmat pelaajat ovat yksimielisesti vahvistaneet tuloksen, tulosta ei voi enää jälkikäteen muuttaa.

Sääntö 11. Luovutus: Pelin aikana, pelaaja voi tunnustaa hävinneensä. Tätä kutsutaan luovutukseksi. Vastustajan sanotaan tällöin voittaneen luovutuksella.

Sääntö 12. Ei tulosta: Kun sama kokolaudan tilanne toistuu pelin aikana ja jos molemmat pelaajat suostuvat, peli päättyy ilman tulosta.

Sääntö 13. Molemmat häviävät:

- I Kun peli on pysäytetty säännön 9 mukaisesti ja pelaajat löytävät laitton joka muuttaisi pelin tulosta ja siksi eivät voi sopia pelin päättyneen, tuomitaan peli häviöksi kummallekin.
- II Jos pelilaudan kiviä on pelin aikana siirtynyt pois paikaltaan, peliä jatketaan siirtämällä kivet oikealle paikalleen. Jos tästä ei päästä sopuun, tuomitaan peli häviöksi kummallekin.

Sääntö 14. Rangaistukset: Jos yllä olevia sääntöjä rikotaan kesken pelin, tuomitaan peli toisen tai molempien häviöksi.

3 Pelien tarkemmat tiedot

Tiedosto 1: gnugo-pachi.dat

```
# Black: GNU Go
# BlackCommand: gnugo --mode=gtp --japanese-rules
# BlackLabel: GNU Go
# BlackVersion: 3.8
# Date: November 1, 2015 12:19:57 PM EET
# Host: hiddenburg (Intel(R) Core(TM) i5 CPU           670  @ 3.47GHz)
# Komi: 6.5
# Referee: Mika
# Size: 19
# White: Pachi UCT
# WhiteCommand: pachi -r japanese
# WhiteLabel: Pachi UCT
# WhiteVersion: 11.00 (Retsugen): Have a nice game!
# Xml: 0
```

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W
0	W+1.5	W+0.5	W+1.5	0	—	216	99.5	1234.8	100	0
1	W+11.5	W+17.5	B+11.5	1	—	267	119.8	1534.6	120.2	0
2	W+3.5	W+2.5	W+3.5	0	—	212	102.1	1108.7	104.2	0
3	W+5.5	W+7.5	B+5.5	1	—	235	123.3	1199.4	124.6	0
4	W+1.5	W+0.5	W+1.5	0	—	306	159.3	1814.4	159.3	0
5	B+0.5	W+1.5	W+0.5	1	—	213	109.8	1286.4	115	0
6	W+R	W+R	W+R	0	—	136	90.1	814.3	89.9	0
7	W+33.5	W+35.5	B+39.5	1	—	261	164.2	1359.5	164.7	0
8	W+13.5	W+24.5	W+13.5	0	—	214	161.6	1189.5	163.7	0
9	W+R	W+R	W+R	1	—	173	131.6	1062.2	131.3	0

Tiedosto 2: gnugo-fuego.dat

```
# Black: GNU Go
# BlackCommand: gnugo --mode=gtp --japanese-rules
# BlackLabel: GNU Go
# BlackVersion: 3.8
# Date: November 1, 2015 10:27:05 PM EET
# Host: hiddenburg (Intel(R) Core(TM) i5 CPU           670 @ 3.47GHz)
# Komi: 6.5
# Referee: Mika
# Size: 19
# White: Fuego
# WhiteCommand: fuego --config fuego.conf
# WhiteLabel: Fuego
# WhiteVersion: 1.1
# Xml: 0
```

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W
0	W+8.5	W+8.5	W+8.5	0	-	233	118.3	737.3	119.8	2641.4
1	W+R	W+R	W+R	1	-	153	138.9	531.5	138.7	1906.6
2	W+1.5	W+1.5	W+1.5	0	-	284	161.8	1031.4	163.2	3689.1
3	B+21.5	B+21.5	W+21.5	1	-	361	144.3	1454	144.4	5125.1
4	W+8.5	W+8.5	W+8.5	0	-	245	150.3	923.7	152	3236
5	W+R	W+R	W+R	1	-	147	144.8	537.8	144.6	1867.5
6	W+5.5	W+5.5	W+5.5	0	-	246	146.6	903	147.8	3192
7	B+R	B+R	B+R	1	-	354	147.2	1389.8	147	4935.3
8	W+R	W+R	W+R	0	-	146	126.7	529.4	126.5	1872.5
9	B+58.5	B+14.5	W+66.5	1	-	335	135	1317.4	135.2	4488.2

Tiedosto 3: pachi-fuego.dat

```
# Black: Pachi UCT
# BlackCommand: pachi -r japanese
# BlackLabel: Pachi UCT
# BlackVersion: 11.00 (Retsugen): Have a nice game!
# Date: November 1, 2015 4:28:38 PM EET
# Host: hiddenburg (Intel(R) Core(TM) i5 CPU           670  @ 3.47GHz)
# Komi: 6.5
# Referee: Mika
# Size: 19
# White: Fuego
# WhiteCommand: fuego --config fuego.conf
# WhiteLabel: Fuego
# WhiteVersion: 1.1
# Xml: 0
```

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W
0	W+R	W+R	W+R	0	-	224	1368.6	730.1	0	1950.1
1	W+R	W+R	W+R	1	-	191	1173.3	633	0	1728
2	B+R	B+R	B+R	0	-	245	1461	852.4	0	2324.2
3	B+R	B+R	B+R	1	-	216	1287	798.6	0	2183.3
4	B+R	B+R	B+R	0	-	225	1353.4	790.8	0	2161.4
5	W+R	W+R	W+R	1	-	291	1775.6	910.4	0	2504.5
6	B+R	B+R	B+R	0	-	205	1066.8	786.1	0	2143.4
7	B+R	B+R	B+R	1	-	242	1382.6	881.9	0	2418.3
8	B+R	B+R	B+R	0	-	225	1192.1	800.7	0	2187.1
9	B+R	B+R	B+R	1	-	248	1336.1	915.9	0	2499.9

Tiedosto 4: gnugo-gnugo.dat

```
# Black: GNU Go
# BlackCommand: gnugo --mode=gtp --japanese-rules
# BlackLabel: GNU Go:3.8
# BlackVersion: 3.8
# Date: November 1, 2015 5:27:26 PM EET
# Host: ouroboros (Intel(R) Core(TM) i5 CPU          M 520 @ 2.40GHz)
# Komi: 6.5
# Referee: Mika
# Size: 19
# White: GNU Go
# WhiteCommand: gnugo --mode=gtp --japanese-rules
# WhiteLabel: GNU Go:3.8
# WhiteVersion: 3.8
# Xml: 0
```

```
#
```

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W
0	W+14.5	W+14.5	W+14.5	0	-	193	52.1	72.1	55.1	75.1
1	B+4.5	B+4.5	W+4.5	1	-	225	66.4	84	68.4	86
2	W+13.5	W+13.5	W+13.5	0	-	224	105.2	102.4	107.7	104.9
3	B+3.5	B+3.5	W+3.5	1	-	192	55.4	59.6	58.1	62.3
4	B+38.5	B+38.5	B+38.5	0	-	246	76.6	94.4	78.6	96.3
5	W+11.5	W+11.5	B+11.5	1	-	215	111.9	118.5	115.9	122.9
6	B+33.5	B+33.5	B+33.5	0	-	241	74.6	90	76.3	91.8
7	B+R	B+R	B+R	1	-	124	58.9	58.5	58.8	58.4
8	B+R	B+R	B+R	0	-	187	88	71.2	87.8	71.1
9	W+14.5	W+14.5	B+14.5	1	-	206	72.7	90.2	75.1	92.4

Tiedosto 5: fuego-fuego.dat

```
# Black: Fuego
# BlackCommand: fuego --config fuegox.conf
# BlackLabel: Fuego:1.1
# BlackVersion: 1.1
# Date: November 2, 2015 2:53:34 PM EET
# Host: ouroboros (Intel(R) Core(TM) i5 CPU           M 520  @ 2.40GHz)
# Komi: 6.5
# Referee: Mika
# Size: 19
# White: Fuego
# WhiteCommand: fuego --config fuego.conf
# WhiteLabel: Fuego:1.1
# WhiteVersion: 1.1
# Xml: 0
```

```
#
#GAME  RES_B  RES_W  RES_R  ALT  DUP  LEN  TIME_B  TIME_W  CPU_B  CPU_W
0      B+R   B+R   B+R   0    -    231  601.4  782.6  3455.2 1157.7
1      W+R   W+R   W+R   1    -    275  798.3  796.3  4185    1205.9
2      B+R   B+R   B+R   0    -    219  559    748.4  3142.4 1075.7
3      B+R   B+R   B+R   1    -    282  772.6  926.7  4326.1 1395.2
4      B+R   B+R   B+R   0    -    267  715.4  893.1  4077.9 1350.9
5      B+R   B+R   B+R   1    -    210  511.8  680.7  2564.9 895.3
6      B+R   B+R   B+R   0    -    233  623.1  801    3471.3 1174.9
7      W+R   W+R   W+R   1    -    237  689.9  706.5  3369    1003
8      B+R   B+R   B+R   0    -    247  633.5  816.5  3576.4 1207.9
9      W+R   W+R   W+R   1    -    293  928.7  885.6  5012.2 1391.3
```

Tiedosto 6: pachi-pachi.dat

```
# Black: Pachi UCT
# BlackCommand: pachi -r japanese -f /usr/share/fuego/book.dat
# BlackLabel: Pachi UCT[1]
# BlackVersion: 11.00 (Retsugen): Have a nice game!
# Date: November 2, 2015 1:07:19 AM EET
# Host: ouroboros (Intel(R) Core(TM) i5 CPU           M 520  @ 2.40GHz)
# Komi: 6.5
# Referee: Mika
# Size: 19
# White: Pachi UCT
# WhiteCommand: pachi -r japanese
# WhiteLabel: Pachi UCT[2]
# WhiteVersion: 11.00 (Retsugen): Have a nice game!
# Xml: 0
```

```
#
```

#GAME	RES_B	RES_W	RES_R	ALT	DUP	LEN	TIME_B	TIME_W	CPU_B	CPU_W
0	B+R	B+R	B+R	0	-	199	1184.6	1213.9	0	0
1	B+R	B+R	B+R	1	-	236	1401.9	1458.2	0	0
2	W+R	W+R	W+R	0	-	278	1680.3	1650.7	0	0
3	W+R	W+R	W+R	1	-	223	1356.2	1339.4	0	0
4	W+R	W+R	W+R	0	-	178	1069	1062.5	0	0
5	B+R	B+R	B+R	1	-	236	1403.5	1449.9	0	0
6	W+R	W+R	W+R	0	-	212	1266.1	1260.9	0	0
7	B+R	B+R	B+R	1	-	186	1108.5	1144	0	0
8	B+R	B+R	B+R	0	-	301	1786.1	1817.6	0	0
9	W+R	W+R	W+R	1	-	237	1442.2	1431.5	0	0