Utveckling av minimax-baserad agent för strategispelet Stratego

Karl Stengård

Examensarbete för 20 p, Institutionen för datavetenskap,
Naturvetenskapliga fakulteten, Lunds universitet

Thesis for a diploma in computer science, 20 credit points, Department of Computer Science, Faculty of Science, Lund University

Utveckling av minimax-baserad agent för strategispelet Stratego

Sammanfattning

Stratego är en brädspel som inte är olikt schack, men som innehåller okänd information. Detta gör att existerande program spelar på nybörjarnivå. Syftet med detta examensarbete är att anpassa en minimaxalgoritm så att den passar för att spela Stratego, och därefter använda den för att konstruera en Strategoagent. Tester med existerande minimaxalgoritmer leder fram till utvecklandet av *p-e-minimax*. Denna algoritm använder sig av två olika värden i sina noder för att simulera att olika information finns tillgänglig för agenten och dess motspelare. M.h.a *p-e-minimax* konstrueras en Strategoagent, *Perspecto.* Denna agent tillämpar en hybrid arkitektur där en enkel strategisk sökning kompletterar *p-e-minimax*. Dessutom används en psykologisk modell för att få agenten att klara av spelet med de okända pjäserna.

Perspecto besegrar en kommersiellt program men förlorar mot en mänsklig spelare efter en jämn match. Det visar sig att det behövs en mer avancerad strategisk sökning och en mer svårförutsägbar psykologisk modell för att Perspecto skall kunna ha någon chans mot en skicklig mänsklig spelare.

Development of a minimax-based agent for the strategy game Stratego

Abstract

Stratego is a boardgame not very different from chess, but that contains hidden information. Because of this, existing programs play at beginner level. The purpose of this thesis is to adjust a minimax algorithm so that it passes the demands of Stratego, and then build a Stratego agent around it. Tests with existing minimax algorithms leads to the development *p-e-minimax*. This algorithm uses two different values in its nodes to simulate the different information available to the agent and its opponent. The name of the agent constructed around p-e-minimax is *Perspecto*. This agent uses a hybrid architecture were a simple strategic search is used in paralell with p-e-minimax. The agent also uses a psychlolgical model to handle the game of the hidden pieces better.

Perspecto defeats a commercial program but loses to a human player after an even game. This and the following tests shows that a more advanced strategic search and a more unpredictable psychological model is needed for Perspecto to have a chance aganist a skilled human player.

Förord

Tack till Eric Astor som tog över handledningen av examensarbetet efter halva tiden.

De som hjälpt mig med tips och diskussioner om olika problem är Jonatan Aronsson som gett tips om programmeringsteknik, Pontus Anrell som hjälpt till med användargränssnitt och Johan Lenander som satt sig in i teorin kring sökning. Peter Söderström har ställt upp som försöksperson mot *Perspecto* och skall ha tack trots att han var ofin nog att vinna.

Slutligen har Josefin Frank funnits med som stöd under hela processen och stått ut med många svordomar framför datorn.

Innehållsförteckning

1. Bakgrund

När jag började fundera på vad jag skulle skriva mitt examensarbete om, kom jag fram till att undersöka möjligheten att göra en AI till ett spel som innehöll dold information. Jag valde att undersöka *Stratego* eftersom det är ett spel som innehåller mycket taktik, och därför lämpar sig bra för att lösa med hjälp av sökning, den AI-metod som jag är mest förtrogen med sedan tidigare. *Stratego* är dessutom det mest välkända brädspelet som innehåller dold information.

När jag var 15 år spelade jag för första gången *Stratego* mot ett datorprogram. Det var Accolades *Stratego* från 1990, och jag blev väldigt besviken på hur lätt det var att slå programmet. Tidigare hade jag spelat schack mot datorer och hade då varit helt chanslös trots att jag ansåg mig vara betydligt bättre på schack än på *Stratego*. Hur kom det sig att datorn kunde vara så bra på ett spel men så dålig på ett annat? Då ingen i min närhet kunde svara på detta skrev jag ett e-post till Roland Davies, programmerare på Accolade:

Hi mr. Davies

My name is Karl and I recently bought the "stratego" game from Accolade. Altough the program looked nice and was easy to use, I soon found out that it played the game very bad. I havn't lost a single game aganist it. I wonder why the game performed so badly when other computer games I have like chess and five-in-a-row are impossible for me to beat.

Looking forward to hear from you. Karl.

En vecka senare kom svaret:

Hi Karl.

I am sorry to hear that you are disapointed. However, we at Accolade are quite aware that the game plays bad. It is very hard for a computer program to play a game where some information is hidden from it. Of course, we could make the program cheat and know the value of all the pieces on the board, but it is no fun playing against a cheater.

Besides, chess programs have been written since the 60's, while our stratego program is one of the first of its kind. We still hope that it could provide some challenge for new players.

Best wishes, Ronald

Detta var för femton år sedan, och det har gjorts andra strategoprogram sedan dess som spelar bättre. Problemet kvarstår dock: inget av de existerade programmen har en chans mot en mänsklig motståndare som kan spelet hyfsat.

2. Introduktion

Denna rapport beskriver ett försök att skapa en agent som kan spela spelet *Stratego*. *Stratego* är ett brädspel, inte helt olikt schack eller dam, där två spelare i tur och ordning flyttar pjäser i syfte att besegra sin motståndare. Precis som i schack har pjäserna olika rang och syfte. Då två pjäser möts i "slag" är det rangen som fäller avgörandet. Motsvarigheten till kungen i schack är i *Stratego* flaggan, som för sin överlevnad är beroende av skydd från de andra pjäserna. Det som skiljer *Stratego* från de klassiska brädspelen är existensen av dold information: i *Stratego* är identiteten hos motståndarens pjäser i början av spelet okänd. Detta gör att *Stratego* blir mer komplext för ett datorprogram att analysera. I schack och dam når de bästa programen världsmästarnivå, medan de i *Stratego* kan bli besegrade av nybörjare.

De flesta framgångsrika brädspelande datorprogram bygger på en och samma metod: minimaxmetoden. Denna metod bygger på att förutsäga motståndarens drag med hjälp av en sökning bland alla tillåtna drag. Man utgår ifrån att motståndaren alltid gör det bästa drag han har möjlighet till. Sökningen fortsätter tills dess den måste avslutas p.g.a tidsbrist. Oftast kan man söka 2 – 20 drag framåt i tiden, beroende på hur många drag som är möjliga i varje position, hur bra programmet är skrivet och hur snabb datorn är. När sökningen nått sin botten anropas en funktion som gör en värdering av den ställning sökningen nått fram till, en s.k evalueringsfunktion. Med minimaxmetoden använder datorn sin överlägsna beräkningskapacitet för att besegra en människa som oftast inte kan tänka lika många drag framåt.

Att använda *minimax* för att spela *Stratego* är svårare än i schack och dam. Hur skall man kunna förutse motståndarens bästa drag om man inte känner till vilka hans pjäser är? Hur skall man veta resultatet av ett framtida slag när man inte vet rangen på den slagna pjäsen? Dessutom spelas *Stratego* med 40 pjäser per spelare, och med så många pjäser blir antalet drag väldigt stort. Man kan alltså inte hoppas på att söka så många drag framåt i spelet.

En utgångspunkt för att finna lösningar på dessa svårigheter, att anpassa *minimax* så att den fungerar bättre för just *Stratego*, är *expectiminimax*[1], en algoritm som används i b.l.a *Backgammon*. Expectiminimax tar hänsyn till slumpens betydelse, och gör förusägelser om vad som statistiskt sett bör inträffa. Även om inget direkt slumpmoment finns i *Stratego* så kan de osäkra momenten simuleras som slumpmässiga processer.

1 Se [1], s 133-134

3. Problembeskrivning

Kortfattat består uppgiften för examensarbetet av att konstruera en mjukvaruagent baserad på *minimax*-algoritmen som skall kunna spela *Stratego*. Genom att göra detta hoppas jag kunna ge svar på följande frågor:

- Kan *minimax*-algoritmen användas för att spela *Stratego* på samma nivå som en skicklig mänsklig spelare?
- Vilken typ av *minimax*-algoritm lämpar sig bäst till att användas för *Stratego*?
- Är *minimax*-algoritmen det bästa sättet för ett program att spela *Stratego*, eller finns det något annat sätt som är bättre?

I mitt projekt valde jag att dela upp problemet i två delar.

Den första delen består i att anpassa de två algoritmerna *minimax* och *expectiminimax* så att de kan användas för att spela *Stratego*. De två algoritmerna skall sedan testas och deras styrkor respektive svagheter skall analyseras. Därefter skall jag använda denna analys för att konstruera en tredje algoritm som jag hoppas skall kunna överbrygga svagheterna hos de två andra. Denna tredje algoritm skall också testas, så att jag vet vilken algoritm som lämpar sig bäst.

I den andra delen av projektet skall jag använda den sökalgoritm jag kom fram till var den bästa för att konstruera en agent som skall kunna spela *Stratego* så bra som möjligt. Här kan det komma att bli aktuellt att komplettera sökalgoritmen med andra funktioner, som t.ex startuppställningsbibliotek. Agenten skall kunna spela bra både mot en mänsklig motståndare samt mot ett annat program och den skall aldrig ta mer än 10 sekunder på sig att göra ett drag.

4. Spelet *Stratego*

Stratego utvecklades 1960 av företaget Milton Bradley i USA. Ett år senare kom den första versionen ut. *Stratego* är alltså inget klassiskt spel som schack utan har kommit fram på senare tid. Efter 1960 har spelet kommit ut i många olika utföranden, men reglerna är oförändrade från orginalet. *Stratego* är populärt i både USA och Europa. I Holland, Belgien och Tyskland finns *Stratego*-föreningar och stora turneringar spelas ofta. Populariteten beror på spelets enkelhet; barn kan lätt lära sig att spela. Samtidigt finns det så väldigt många trick och strategier att använda, att det är svårt att tröttna.

Förutom att orginalspelet kommit fram i nya utföranden har en rad varianter av *Stratego* utvecklats. *Stratego legends, Star wars stratego, Sabotage* och *Stratego 4* är några av dessa. Dessa varianter har andra typer av pjäser och spelplaner än orginalet, men principen för spelet är den samma.

Två gånger har *Stratego* givits ut som datorprogram för PC. Första gången var 1990 av *Accolade* och andra gången 1999 av *Hasbro*. Båda programmens AI har så svag spelstyrka att de enbart lämpar sig för spel mot andra människor över nätverk. På senare tid har det dock producerats strategoprogram som finns att få tag på på nätet och som visserligen heter andra saker, då *Stratego* är ett registrerat varumärke, men som har samma regler som orginalet. Ett par av dessa program spelar helt okej, även om de lätt besegras av någorlunda avancerade spelare. Det senaste av dessa program är *The General* som är skrivet av Sean O'Connor.

Bild 1: Den variant av Stratego jag själv äger, tillverkat 1987.

4.1 Grundläggande regler

Reglerna här är de turneringsregler som används av International Stratego Federation (ISF).[2]

Stratego spelas på ett bräde om 10×10 rutor (se Bild 1). På de fyra raderna närmast respektive spelare sätter denna upp sina 40 pjäser. Dessa kan sättas upp hur som helst. På detta sätt blir bara de två mittersta raderna fria från pjäser. Här finns dock två "sjöar" om 2×2 rutor som blockerar all förflyttning. På detta sätt blir bara tre korridorer, en i mitten och två längs kanterna, öppna för förflyttningar i början av spelet. Vid spelets start vet ingen av spelarna något om den andres pjäser.

De flesta pjäser flyttar bara ett steg, och bara i fyra riktningar, ej diagonalt. Undantagen från denna regel är flaggan och bomberna som ej får flyttas, samt spejaren som får gå hur många steg som helst i raka linjer, som ett torn gör i schack. Den som flyttar en pjäs avslöjar alltså att denna pjäs inte är en bomb eller en flagga, och om man flyttar flera steg kan motståndaren vara säker på att man flyttat en spejare.

Förutom det man kan få reda på m.h.a förflyttningen vet ingen av spelarna något om den andres pjäser förrän någon försöker slå ut en motståndarpjäs. Då måste pjäsen med lägst rang tas ur spelet. Om två pjäser med samma rang möts tas båda ur spel. Flaggan kan bli slagen av samtliga pjäser. De flyttbara pjäserna har olika rang, från 10(lägst) till 1(högst). Även här finns dock undantag. Bomber tas ur spel enbart om de attackeras av en minör(rang 8). Alla andra pjäser som råkar på en bomb tas själva ur spel. Det andra undantaget gäller spionen. Spionen(rang 10) tas ur spel alla gånger den kommer i kontakt med en fientlig pjäs utom då den själv attackerar motståndarens fältmarskalk(rang 1), då tas den senare ur spel. Spionen kan också, precis som alla andra pjäser, slå ut flaggan.

Det finns också två specialregler som oftast saknar betydelse men som kan bli viktiga i speciella situationer. Enligt den första får man inte flytta samma pjäs tre gånger i rad över samma två rutor.[3] Enligt den andra får man inte jaga efter en motståndarpjäs i all oändlighet som man inte har möjlighet att fånga. Denna sista regel är dock svår att implementera och utelämnas oftast i mjukvara.

Spelet vinnes då man slår ut motståndarens flagga eller då motståndaren saknar möjlighet att utföra ett drag. Ett parti slutar oavgjort när båda spelare är överens om att inget avgörande kan nås. I alla tester mellan datorprogram har jag satt en maximal tidsgräns vid 300 drag. Om ett parti håller på så länge förklaras det vara oavgjort.

Vid notation av dragen i *Stratego* används samma system som i schack, d.v.s A5-A6, F6-

2 Reglerna finns i sin helhet i [3]

3 Enligt ISF är det tillåtet att flytta fem gånger över samma två rutor, men i alla datorprogram sätts stopp vid tre. Även i min implementering är max tre rutor tillåtet.

G6 o.s.v. Vid slag används “x” istället för bindestreck, vid misslyckat slag “o” och vid dubbelslag “xo”. I illustrationer av positioner markeras en känd pjäs med ett streck under sitt värde, och en pjäs som flyttats markeras med en prick.

4.2 Pjäsernas egenskaper och användning

Här följer information om de pjäser som använs i spelet, med beteckning eller rang inom parantes.

Flaggan (F) är den viktigaste pjäsen, men också den svagaste. Flaggan skall därför oftast skyddas med bomber så att den blir svårare att komma åt. Dessa bomber kan dock ibland dra oönskad uppmärksamhet åt flaggans håll. Ibland placeras flaggan därför helt oskyddad på någon ovanlig plats, t.e.x bakom en av sjöarna, för att vilseleda motståndaren.

Bomben(B) är en defensivt stark pjäs, då endast 5/33 rörliga pjäser (minörerna) rår på den. Bomberna skyddas bäst av lågt rankade pjäser som kan slå ut minörerna. En svaghet hos bomberna är att de lätt blir ett hinder för egna pjäser, då de ej kan flyttas. Man har sex stycken bomber från början.

Spionen(S) är en svag pjäs som slås ut vid attacker från samtliga andra pjäser. Spionen har bara ett syfte: att slå ut motståndarens fältmarskalk. Ofta kan blotta rädslan för spionen göra att motståndaren håller tillbaka sin fältmarskalk. Om spionen däremot blir utslagen kan marskalken röra sig utan hot från rörliga pjäser. Spionen är väldigt känslig för attacker från spejare, och måste därför hållas skyddad från dessa. Man har bara en spion.

Spejaren(9) är den vanligaste pjäsen, det finns hela åtta spejare på varje sida i början av spelet. Spejarens stora styrka är dess snabbhet, och den kan därför komma åt pjäser som är utom räckhåll för andra, t.ex de bakre linjerna hos moståndaren. Snabbheten kommer bäst till sin rätt mot slutet av partiet då få pjäser är kvar och spejaren kan utnyttja ytorna. En attack med en spejare innebär nästan säkert att den blir utslagen, så det är ofta en bra idé att avvakta i början. Dessutom kan det då uppkomma ett läge, då ens motståndare inte kan ta ens flagga p.g.a att den är skyddad av bomber och han saknar minörer. Hans enda hopp om seger blir då att slå ut alla rörliga pjäser, men det är svårt om det finns spejare kvar. Dessa kan oftast glida undan och på så sätt få till ett oavgjort resultat i spelet, då inget avgörande kan nås.

Minören(8) är den enda pjäsen som kan slå motståndarens bomber och är därför en offensivt stark pjäs trots sin låga rang. Att ha minörerna utspridda från början är ofta bra, då de i sådana fall snabbt kan kallas in då en bomb upptäcks. Man har fem minörer, och det finns sex bomber att slå ut, så det är viktigt att skydda minörerna från andra pjäser.

Sergeanten(7), Löjtnanten(6) och **Kaptenen(5)** är pjäser med medelhög rang. De har inga specialfunktioner som de med lägre rang och man har dessutom relativt många, fyra av varje sort. Dessa pjäser kan skickas fram i hopp om att slå ut en pjäs med lägre rang, eller avslöja någon av de väldigt högt rankade. Därför behöver man inte vara så rädd om pjäserna med medelhög rang och de kan anfalla pjäser som ännu inte flyttat trots risken för att dessa skulle vara bomber. De medelhöga pjäserna står ofta i första ledet eller som försvarare av bomber mot fientliga minörer. Sergeanten är enligt de flesta spelets minst värdefulla pjäs, men kan ändå slå ut 19/40, nästan hälften, av motståndarens pjäser.

Majoren(4), Översten(3) och **Generalen(2)** är pjäser med hög rang och det är ofta en övervikt av dessa som blir avgörande i *Stratego*. Dessa pjäser skall därför undvika att slå pjäser som inte flyttat, då dessa kan vara bomber. Den som kan få en pjäs som kan slå alla andra rörliga utan rädsla för att själv bli slagen av annat än en bomb har en stor fördel. Om man t.e.x har förlorat sina överstar medan motståndaren har överstar kvar, måste man skydda generalen så att denne fortfarande är kvar och kan hota motståndarens överstar. Varje spelare har tre majorer, två överstar och en general.

Fältmarskalken(1) är den högst rankade pjäsen, och det finns bara en. Fältmarskalken kan bli utslagen på tre sätt: Genom att den anfaller en bomb, genom att den anfalles av spionen eller genom att den stöter på motståndarens fältmarskalk. I det sistnämnda fallet tas båda pjäserna ur spel, så detta är ofta inte så allvarligt. Om fältmarskalken undviker att anfalla orörliga pjäser har han alltså enbart motståndarens spion att frukta.

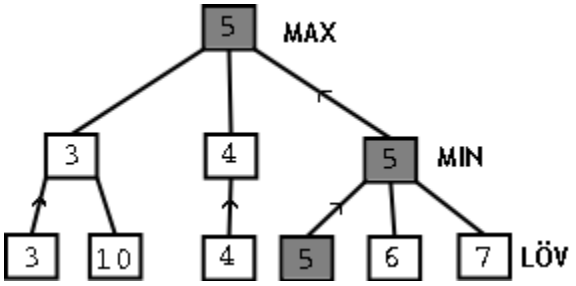
5. Minimaxmetoden

Idén att vissa typer av spel kan spelas av maskiner har funnits långt innan dessa maskiner existerat. Ett tidigt exempel på detta var "Turken", en schackspelande maskin som förevisades 1769 av Baron Wolfgang von Kempelen. "Turken" visade sig dock innehålla en gömd schackexpert som bestämde vilka drag den skulle göra. De första schackspelande datorprogramen konstruerades inte förrän på 50-talet.[4] Dessa använde sig av minimaxmetoden, som föreslagits redan 1944 av Von Neumann och Morgensen.

Minimaxmetoden är väldigt intuitiv och bygger på att man i varje ställning kan titta ett drag framåt och fråga sig: Vilket drag skulle jag själv göra om jag var min motståndare? *Minimax* kan sedan användas rekursivt och titta hur många drag som helst framåt om det finns tillräckligt med tid. För ett deterministiskt spel med två spelare där all information är känd, kan alltså *minimax* rent teoretiskt alltid göra det bästa draget och således alltid vinna om det är möjligt. Detta är dock bara praktiskt möjligt för väldigt enkla spel, som nim eller fyra-i-rad. Ett spel som schack skulle ta miljarder år att analysera på detta sätt även med en väldigt snabb dator. Därför måste *minimax*-sökningen avbrytas efter att en viss tidgräns har passerats, och det antal halvdrag sökningen nu hunnit med kallas för dess maximala sökdjup. Vid detta djup måste en evalueringsfunktion anropas, som värderar den aktuella positionen. Som exempel kan sägas att ett schackprogram som söker fyra halvdrag framåt spelar på nybörjarnivå, ett som tittar åtta halvdrag spelar på stark klubbspelarnivå och tolv halvdrag framåt innebär att programmet håller mästarnivå.

5.1 Sökträdet

Minimaxmetoden bygger på att ett spel kan beskrivas som ett träd. Varje nod i trädet beskriver en position, och varje båge beskriver ett drag. Trädets rot är den aktuella postionen, och bågarna som utgår från roten är de drag som finns att välja på.



Figur 1: Sökträd för minimax. Ett högt värde indikerar en för agenten fördelaktig position.

4 Se [1] s 141 ff

Tidskomplexiteten för *minimax*-sökning är $O(b^{djup})$[5], där b är antalet tillåtna drag i varje punkt, även kallat *förgreningsfaktorn.* Den genomsnittliga förgreningsfaktorn är olika för olika spel, som framgår av nedanstående tabell.

Spel	***Genomsnittlig förgreningsfaktor***
Go	350
Schack	35
Othello	10
Fyra-i-rad	7
Stratego	30(45*)

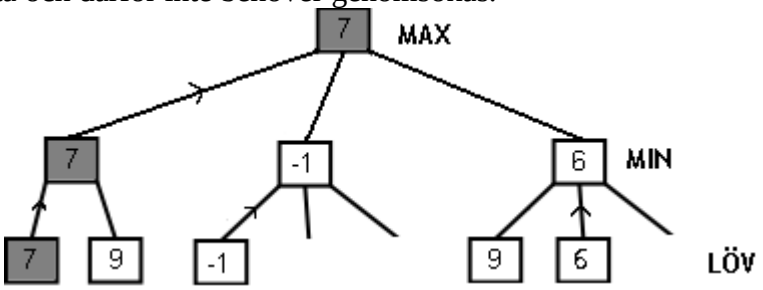
Tabell 1: Genomsnittlig förgreningsfaktor för olika spel.

*Spejarnas förflyttningar medräknade

Förgreningsfaktorn har stor betydelse för hur lätt en dator kan spela ett spel m.h.a *minimax*. I *Othello* är de bästa programmen överlägsna människor, i schack är människa och dator jämbördiga och i go spelar de bästa programmen på nybörjarnivå. I *Stratego* kombineras problemet med stor förgreningsfaktor med att all information inte är känd. Detta gör *Stratego* ytterst svårspelat med *minimax*, och existerande strategoprogram använder sig ofta av andra metoder.

5.1.1 Att klippa i trädet

Den stora tidskomplexiteten hos *minimax* gör att man ofta inte kan söka speciellt många drag framåt. En metod för att komma runt detta problem är *alpha-betaklippning* av sökträd. Alpha-beta är ett sätt att snabba upp *minimax*-sökningen, utan att resultatet av sökningen förändras. Detta kan göras genom att man kan visa att vissa noder i trädet är helt ointressanta och därför inte behöver genomsökas.



Figur 2: Alpha-betaklippning i sökträd

5 Se [1] s 126

Som synes i figur 2 har det stor betydelse att de bästa dragen genomsöks först. Eftersom max-funktionen redan hittat ett drag med värde 7 är det ointressant att söka vidare från de noder där ett sämre värde redan hittats. Trädet i figur 1 däremot hade inte alls gått att klippa i. Att sortera dragen med de bästa först är svårt att åstadkomma (om man visste behöver man ju inte söka alls!). Ofta räcker det dock att de sannolikt bästa dragen undersöks först, för att man skall få en avsevärd förbättring i jämförelse med vanlig *minimax*. Om detta kan göras, blir komplexiteten endast $O(b^{djup/2})$.Program som använder alpha-beta kan alltså söka dubbelt så många ply som de med *minimax*.[6]

5.2 Komplikationer baserade på slumphändelser

Fia med knuff, poker och backgammon är exempel på spel där slumpen spelar roll, men där spelarnas strategier också påverkar utgången. Här finns ingen strategi som garanterar vinst, målet är istället att hitta det drag som maximerar sannolikheten för vinst i spelet. Med en utvidgning kan mimimax användas även för dessa spel. För att göra detta introducerar man *slumpnoder* som beskriver en position och ett visst tärningsslag, ett draget kort eller annan händelse som inträffar med given sannolikhet. Utifrån en slumpnod kan sedan de möjliga dragen genereras. Värdet av en viss position fås genom att man summerar över alla slumpmässiga händelser med vikter beroende på sannolikhet.

Denna utvidgning av *minimax* kallas för *expectiminimax*. Algoritmens svaghet är dess tidskomplexitet som är $O((b*n)^{djup})$, där n är genomsnittligt antal slumpnoder per position.[7] Den totala förgreningen i varje positionsnod blir alltså väldigt stor.

Spel	***b***	***n***	***b*n***
Fia med knuff	3	6	18
Backgammon	20	21	420
Limit Hold'em	3	338	70

Tabell 2: Genomsnittlig förgreningsfaktor i spel med slumpmoment

För att råda bot på den höga förgreningen kan man även här använda sig av alpha-betaklippning, men detta måste göras med större försiktighet och med mer beräkningar inblandade än med vanlig *minimax*. Den inblandade evalueringsfunktionen måste också ge resultat som hamnar inom vissa fördefinierade gränser för att trädet skall kunna klippas.

Om expectiminimax tillämpas för *Stratego* behöver slumpnoderna bara användas vid själva slagen. Dessa utgör mindre än 10% av dragen och varje slag har bara tre möjliga utfall. Den totala förgreningsfaktorn i *Stratego* påverkas alltså inte speciellt mycket. En

6 Se [1] s 131
7 Se [1] s 135

större nackdel är att expectiminimax försvårar användandet av alpha-beta.

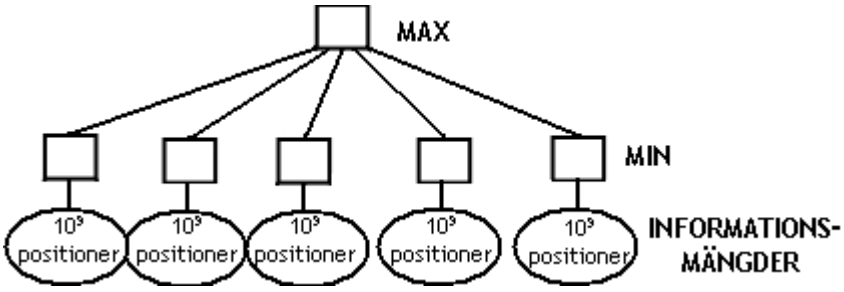
5.3 Komplikationer baserade på okänd information

Minimax och *expectiminimax* är välkända algoritmer som ingår i de flesta kurser i artificiell intelligens på universitetsnivå världen över. Deras motsvarighet för spel med okänd information, *IMP-minimax*, är inte lika känd.

IMP-minimax behandlar den okända informationen i ett spel genom att förutom noder använda s.k informationsmängder i sökningen. En informationsmängd är den mängd av positioner som kan finnas vid ett given situation i ett spel.

Algoritmen föreslogs först av Wilson år 1972, men då för spel med en enda aktiv spelare där denna spelare kan komma ihåg sina tidigare drag. Wilson visar att det i sådana spel är möjligt att hitta den bästa strategin på en tid som är linjär i förhållande till sökträdets storlek, d.v.s precis som med vanlig minimax.[8]

För spel med två eller fler aktiva spelare finns det dock inget annat sätt att hitta den garanterat bästa strategin än att jämföra alla möjliga strategier i sökträdet, något som är *NP*-svårt.[9]



Figur 3: IMP-minimax i inledningen av ett parti Stratego.

Som synes i figur 3 blir förgreningsfaktorn i *IMP-minimax* oerhört stor, och den blir därför omöjlig att använda i sin "rena" form, d.v.s utan att man inför begränsningar på vilken information i informationsmängderna som är intressant. Hur dessa begränsningar skall se ut är dock ett mycket svårt problem.

För ett spel med två spelare som Stratego är det heller inte säkert att en vanlig sökning hittar det bästa draget. Därför verkar *IMP-minimax* vara en algoritm som inte passar bra till *Stratego.*

8 Se [4]
9 Se [5], s 1

6. Problemdiskussion

För att kunna använda minimax för att spela *Stratego* måste en rad problem övervinnas. En fråga som kan vara bra att ta ställning till redan nu är vilken typ av *minimax*-algortim som är bäst att utgå ifrån. Här kan det vara bra att ta reda på hur andra har löst liknande problem.

Magisteruppsatsen *Multi-agent Stratego* från 2000 beskriver ett försök att skapa en *Stratego*-agent med ett multiagentsystem.[10] Författaren Caspar Treijtel bygger en S*tratego*-spelande prototyp där varje pjäs gör sin egen bedömning av den lokala miljön och handlar utifrån den. Pjäsernas förflyttningar evalueras av ett expertsystem som är helt reaktivt, d.v.s ingen sökning eller planering förekommer.[11] Vilken pjäs som i slutändan skall flyttas bestäms av en överordnad struktur som jämför de olika alternativen. Treijtel testar också sitt program mot en mänsklig spelare m.h.a en förenklad variant av *Stratego*. Den mänskliga spelaren vinner matchen, men först efter vissa besvär. I slutspelet visade sig dock Treijtels program alltför svagt.[12]

Multi-agent Stratego visar en alternativ väg att gå för att skapa ett strategoagent. Svagheten med ett multiagentsystem av den typ Treijtel förespråkar är det bristande samarbetet mellan pjäserna, samt svårigheten att få pjäserna att reagera mot globala förändringar på spelplanen. Det lokala synsättet i *Multi-agent Stratego* är dock intressant, kanske är det möjligt att överföra det till ett *minimax*-baserat program.

Multi-agent Stratego är det enda akademiska arbete jag hittat om *Stratego*. De flesta artiklar som skrivits om spel med okänd information fokuserar på kortspelen poker och *Bridge*. Därför finns det inte så mycket att utgå ifrån förutom det egna sunda förnuftet när jag skall bestämma vilken *minimax*-algoritm som utgör den bästa utgångspunkten för att konstruera en *Stratego*-spelande agent.

Den algoritm känns mest naturlig att börja med är *IMP-minimax*, eftersom den är den enda *minimax*-algoritmen som hanterar okänd information. Att använda denna algoritm utan att förenkla den är dock omöjligt p.g.a den stora förgreningsfaktorn (se avsnitt 5.3).

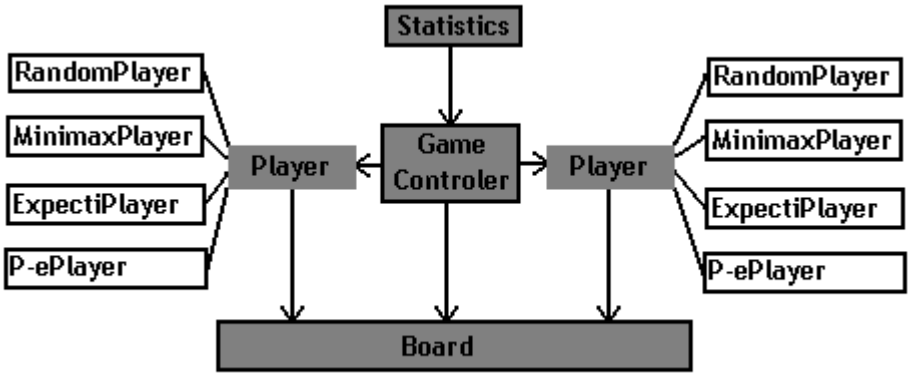
En metod för att kringgå problemen med *IMP-minimax* är att minimera den information man bryr sig om, och sedan behandla den okända informationen som en slumphändelse. På detta sätt kan man använda en algoritm som inte innehåller informationsmängder, och som därmed är enklare och snabbare. Det gäller bara att hitta en sådan algoritm utan att göra alltför stora generaliseringar om spelet.

10 Se [6], s 1
11 Se [6], s 28 ff
12 Se [6] s 47

7. Minimax för *Stratego*

För att en sökalgortitm skall kunna användas för att spela *Stratego* måste den uppfylla vissa kriterier. För det första måste algoritmen klara av att bestämma det bästa draget i de flesta vanliga stratego-situationer. För det andra måste sökningen kunna ske med en godtagbar hastighet. För det tredje måste algoritmens sätt att simulera de framtida dragen stämma någorlunda med verkligheten, detta eftersom den då blir lättare att följa och lättare att förbättra.

För att pröva kvalitén hos olika algoritmer kommer både ett teoretiskt resonemang och ett praktiskt test att genomföras. Det praktiska testet innebär att de olika sökalgoritmerna spelar matchserier i bäst av 100 matcher mot varandra. Om en match varar längre än 300 drag räknas den som oavgjord. För att göra detta möjligt implementerades ett *Stratego*-program i Java för att två agenter skall kunna spela mot varandra.



Figur 4: Testprogrammets struktur. Player är en abstrakt klass som kan ha en av fyra spelartyper som subklass. Player-klassen har inte insyn i hela Board, motståndarens pjäser förblir dolda.

Detta avsnitt beskriver de olika problem som man stöter på vid utvecklingen av sökalgoritmen. Dessa kan delas upp i fem delar: startuppställningen, draggenereringen, beräkningen av sannolikheter, förgreningsfaktorn och evalueringen.

7.1 Startuppställningar

Startuppställningen har stor betydelse för utgången av ett parti *Stratego*. Att komma fram till en bra startuppställning är ett svårt problem för ett program, eftersom kvalitén hos en uppställning beror på vad motståndaren väntar sig. Existerande strategoprogram använder sig ofta av en databas med fördefinierade startuppställningar. För att testa de

olika algoritmerna vill man spela serier av ett stort antal matcher. För att dessa matcher inte alla skall bli likadana måste man använda olika startuppställningar.

En möjlighet är att göra startuppställningarna helt slumpmässiga. På detta sätt finns ingen risk för att matcherna skall upprepas, då det finns 40!/6!8!5!4!4!4!3!2! = 10^9 olika startuppställningar. Problemet är att de flesta av dessa uppställningar är väldigt dåliga och placerar flaggan helt oskyddad. Matcherna avgörs ofta genom snabba attacker, och utgången beror mer på tur med uppställningen än på vilken sökalgoritm som används.

Ett bättre sätt är att endast låta vissa delar av uppställningen slumpas fram, en "halvslumpmässig" uppställningsmetod. Här står flaggan alltid någonstans på sista raden, omgiven av bomber. Minörerna, spejarna, spionen och de högst rankade pjäserna dras bort från den främsta linjen. Även om denna metod inte kan generera lika många ställningar, är den fortfarande tillräckligt varierad samtidigt som ställningarna blir mer balanserade och lika dem som bra mänskliga spelare använder.

7.2 Draggenerering

Då motståndarens pjäser är okända, kan man inte exakt veta vilka motståndardrag som är tillgängliga. Som beskrivs i avsnitt två, kan pjäserna flyttas på tre olika sätt:

- Pjäsen kan inte flytta alls (bomber och flagga, 7/40 pjäser).
- Pjäsen kan flytta obegränsat antal steg (spejare, 8/40 pjäser).
- Pjäsen kan flytta ett steg (alla utom ovanstående, 25/40 pjäser).

En exakt sökalgoritm skulle simulera olika drag-genereringar för alla de olika fallen. Detta skulle ta oacceptabelt lång tid. Därför förenklar jag och antar att alla okända pjäser kan flytta ett steg. Detta kan t.ex. göra att agenten blir överraskad av oväntade spejarattacker, men det är inget som kommer att ha avgörande betydelse.

7.3 Beräkning av sannolikheter

De okända pjäsernas rang är naturligtvis också okänd, och när en okänd pjäs är involverad i en attack är resultatet av attacken okänt. Tre resultat är möjliga: att anfallaren vinner, att förloraren vinner eller att båda blir slagna. Det bästa man kan göra är att räkna ut sannolikheter för de olika händelserna. Även här gör jag dock en förenkling: alternativet att båda pjäserna blir slagna (s.k. dubbelslag) kommer att ha sannolikheten noll. Förenklingen görs för att detta ofta inte har någon betydelse för resultatet av sökningen.

Att båda pjäserna kan slås ut är heller inte något man brukar räkna med när man som mänsklig spelare attackerar en okänd motståndarpjäs.

Sannolikheten för att en attack där den ena eller båda pjäserna är okända skall sluta med framgång (P), fås genom att summera över alla återstående okända pjäser. Här finns olika faktorer att ta hänsyn till:

- Mängden okända pjäser hos agent A(anfallare) eller F(försvarare), med en rang r som uppfyller villkoret B (flagga har r=0, bomb r=1 osv.) Förkortas O(A,B) eller O(F,B).
- Antal okända pjäser som hos F som någon gång flyttats under spelet. Förkortas m.
- Rangen hos en eventuell känd pjäs som är med i slaget, förkortas k.
- Mängden okända pjäser hos agent A eller F vars rang gör att attacken kommer att sluta med framgång då motståndarens rang är k. I denna mängd ingår även hälften av de pjäser som orsakar dubbelslag. Förkortas V(A,k) eller V(F,k).

Det finns fem olika basfall för hur sannolikheten skall beräknas.

Attack med okänd pjäs mot känd försvarare. Enkelt att räkna ut, den anfallande pjäsen måste kunna flytta för att kunna anfalla. Här är sannolikheten för ett lyckat anfall lika med de pjäser som kan slå ut försvararen delat med det totala antalet flyttbara pjäser hos anfallaren.

$$P = \frac{|V(A,k)|}{|O(A,r>1)|}$$

Attack med känd pjäs mot okänd försvarare som flyttats. Räknas ut med samma princip som ovan. Sannolikheten är lika med de antal pjäser som blir utslagna av den anfallande pjäden delat med det antal flyttbara pjäser som försvararen har.

$$P = \frac{|V(D,k)|}{|O(D,r>1)|}$$

Attack med känd pjäs mot okänd försvarare som ej flyttats. Detta är mer komplicerat,

då sannolikheten beror på hur många av försvararens okända pjäser som flyttats. Först måste man räkna ut det förväntade antalet rörliga pjäser hos försvararen som ännu inte flyttat och som kommer att bli slagna(förkortas s). Detta räknas ut genom att multiplicera de antal pjäser som kommer att bli slagna med kvoten de rörliga pjäser som ej flyttat och det totala antalet rörliga pjäser.

$$s = |V(D,k)| \cdot \frac{|O(D, r>1)| - m}{|O(D, r>1)|}$$

För att sedan beräkna sannolikheten måste även de stationära pjäserna (bomber och flagga) tas med i beräkningen. Om den anfallande pjäsen är en minör adderas både motståndarens bomber och flagga till *s,* annars adderas enbart flaggan. Denna summa divideras sedan med motståndarens totala antal pjäser som ej flyttats.

$$P = \frac{s + |O(D, r<2)|}{|O(D, r \geqslant 0)| - m}$$

Attack med okänd pjäs mot okänd försvarare som flyttats. Då både pjäserna är okända får man summera över alla den ena spelarens (här anfallarens) pjäser.

Attack med okänd pjäs mot okänd försvarare som ej flyttats. Samma som ovan, men precis som i fall tre får man ta hänsyn till antalet pjäser som flyttats.

De två sistnämnda fallen berör attacker med två okända inblandade. Man kan ifrågasätta att dessa någonsin kommer att bli aktuella, då en agent i alla lägen känner till värdet av sin egen pjäs. Det kan dock vara så, att man i en sökning vill ta reda på med vilken sannolikhet motståndaren tror sig kunna slå ut en pjäs. Då får agenten sätta sin egen pjäs som okänd för att kunna sätta sig in i motståndarens resonemang och göra en bedömning av vilket motståndarens nästa drag blir. På detta sätt kan en och samma attack ha två olika sannolikheter att lyckas, beroende på om man ser det i motståndarens perspektiv eller sitt eget.

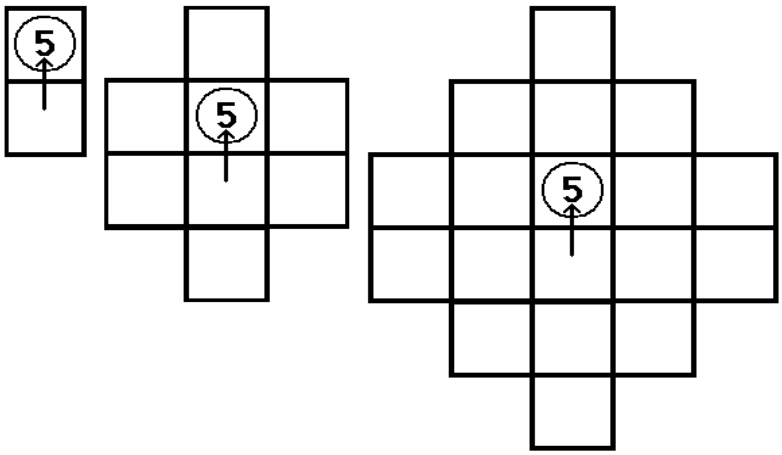
Ett annat problem då man beräknar sannolikheter är att motståndarens pjäser inte befinner sig där de gör av en tillfällighet. Pjäserna har blivit flyttade eller uppställda som resultatet av en strategi. I detta avsnitt får vi dock nöja oss med denna rent kombinatoriska beräkning av sannolikheter.

7.4 Förgreningsfaktorn

Den genomsnittliga förgreningsfaktorn för *Stratego* är ca 45 om alla drag tas med. Detta är alldeles för mycket om man vill att *minimax* skall söka mer än tre drag framåt. Eftersom jag inte vet vilken algoritm jag skall använda i min slutliga implementation vet jag inte heller om klippning i sökträdet är möjlig. Därför är det önskvärt om förgreningsfaktorn skärs ner. Spejarnas drag kan t.ex begränsas till att antingen flytta ett steg eller att gå så långt som möjligt i en riktning. En annan metod är att söka lokalt.

Lokal sökning innebär att man inte tittar på alla svarsdrag till ett drag. Enbart de drag som görs i närheten tas i beaktande. Lokal sökning är bara möjlig i ett spel där pjäsernas möjliga förflyttning är betydligt mindre än spelplanens storlek. I schack, t.ex, skulle en lokal sökning misslyckas då många av pjäserna flyttar tvärs över hela spelplanen. I *Stratego* är det bara spejarna som har möjlighet att göra dessa långa förflyttningar, och deras betydelse är begränsad. Med lokal sökning kan man dramatiskt reducera förgreningsfaktorn. Frågan är bara: Hur många drag kan sorteras bort utan att spelstyrkan påverkas negativt?

För att kunna bestämma vilka drag som skall tas med i den lokala sökningen definierar vi sökningens *sökområde* som det område runt det första draget där sökningen skall leta efter nya drag. Även drag utifrån, in i sökområdet skall tas i beaktning, liksom drag från sökområdet och ut. Figur 1 visar sökområden med olika radie runt ett drag. Hur stor skall denna radie vara? En bra utgångspunkt är att radien måste öka när sökdjupet ökar, eftersom pjäser då kan flytta flera steg och blanda sig i ett händelseförlopp.



Figur 5: Sökområde med radie 0, 1 och 2.

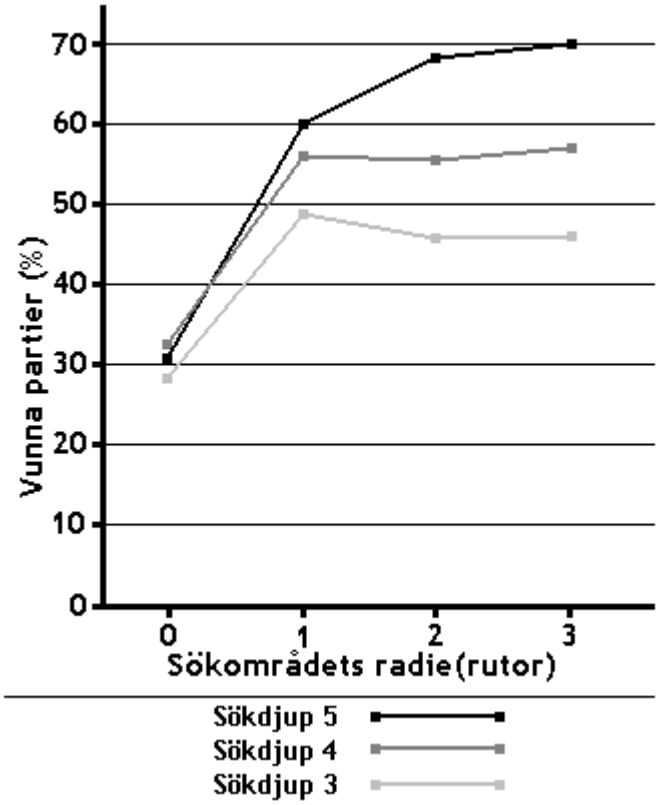
En annan sak att tänka på vid lokal sökning är att olika drag kommer att generera olika serier av svarsdrag. Det kan därför verka som om ett visst drag är bättre än ett annat, därför att man under sökningen undviker ett speciellt otrevligt svarsdrag. Denna effekt av den lokala sökningen är naturligtvis inte önskvärd, men den kan undvikas genom att vinsten av att göra ett visst drag jämförs med vinsten av att inte göra det. Denna skillnad

avgör om ett drag är bra eller dåligt. Detta tillägg i sökningen kallas *jämförelsesökning.* Jämförelsesökningen gör att alla förstadrag måste kollas två gånger, så att sökningen tar dubbelt så lång tid. Detta är ett pris väl värt att betala för att få ner förgreningsfaktorn.

Sökområdet bestäms av det drag som görs på nivå ett, och bibehålls på samma sätt genom sökningen. Genom att behålla sökområdet konstant kan man undvika att göra jämförelsesökningar på alla nivåer.

En lokal sökning kan aldrig avgöra om ett drag är tvunget eller ej, eftersom det kan finnas drag utanför det lokala sökrummet. Därför finns i alla ställningar också möjligheten att inte göra något alls(det passiva draget), och detta måste tas med i sökningen.

Vilket söksätt som är mest fördelaktigt att använda för *Stratego* kan bara avgöras genom ett test. Testet består i en serie av 200 matcher mot en agent som använder global sökning med sökdjup 3. All sökning sker med algoritmen *p-e-minimax*, som beskrivs i avsnitt 6.4.



Figur 6: Test av olika sökområden

Figur 6 visar att spelstyrkan ökar avsevärt vid utökande av sökområdets radie från 0 till 1. Utöver denna radie ökad den inte nämnvärt utom när sökdjupet ökas till fem. När

sökdjupet når fem simuleras tre egna drag vilket innebär att om en och samma pjäs flyttas riskerar den att hamna utanför ett sökområde med radie 1. Det verkar alltså vara en bra idé att låta sökområdet vara precis så stort att detta undviks.

En annan fråga är hur mycket den lokala sökningen minskar förgreningsfaktorn. Detta testades samtidigt som försöket ovan genomfördes. Alla data är från försöken med sökdjup 3. Tabell 3 visar resultatet.

Sökområdets radie	***Genomsnittlig förgreningsfaktor***
0	2.2
1	4.2
2	6.8
3	9.9
Global sökning	31.4

Tabell 3: Genomsnittlig förgreningsfaktor vid lokal och global sökning

Tabell 3 tar inte med de drag som genereras i roten, eftersom de är resultatet av en global sökning. I roten är förgreningen ca 60, då varje drag genereras två gånger p.g.a jämförelsesökningen. Spejarnas långa drag är inte medräknade.

Det står klart att lokal sökning är ett sätt att kringgå den stora förgreningsfaktorn hos *Stratego*, utan att spelstyrkan reduceras.

7.5 Evalueringen

En bra evalueringsfunktion är väldigt viktig för att en brädspelande agent skall vara framgångsrik. Det enklaste sättet att evaluera en ställning är att beräkna värdet av de pjäser varje spelare har. I schack, t.ex, kan en enkel evalueringsfunktion bestå i att ge varje en poäng för varje bonde, tre för varje löpare eller springare, fem för varje torn samt nio om spelaren har damen kvar i spel. I *Stratego* kan man göra likadant, men här finns ingen färdig teori som bestämmer vad pjäserna är värda. Man får därför utgå från vad man anser att pjäserna är värda när man själv spelar.

En evalueringsfunktion för *Stratego* kan göras nästan lika enkel som i schack, men i och med att pjäsernas värde är beroende av varandra får man göra vissa ändringar. Pjäsernas värde delas därför upp i två olika delar. Adderar man dessa värden får man pjäsens totala värde.
Rangvärdet höjs ett snäpp för varje värdefullare pjäs hos motståndaren som helt försvinner från spelplanen. Exempel: Agent B har förlorat sin general och alla sina

kaptener. Agent A's kaptener höjs därför ett snäpp och blir värda 35p. Minörerna höjs två snäpp och blir värda 20+20 = 40 p.

Specialvärdet finns bara kvar så länge pjäsernas speciella egenskap är användbar. Exempel 1: Om alla Agent A's bomber är utslagna förlorar Agent B's minörer sina 20 p. Exempel 2: Om Agent A's spion är slagen förlorar Agent B's fältmarskalk sin speciella egenskap (att kunna bli utslagen av spionen) och blir värd 30 p extra.

För att en agent skall sporras att aktivt söka efter motståndarens flagga sprids flaggans värde ut på de pjäser som misstänks kunna vara flaggan. En misstänkt pjäs värdeökning blir därför: flaggans värde * antalet misstankar pjäsen har / totalt antal misstankar. Alla okända pjäser på sista raden som inte rört sig får en misstanke, dessutom får de som står ett steg ifrån en bomb en misstanke. Man kan invända att man på detta sätt kan lura agenten genom att placera flaggan på annan rad än bakersta, långt ifrån bomber. Den som placerar här löper dock stor risk att få den utslagen av en mer slumpmässig attack, eftersom den blir svår att försvara.

Den evalueringsfunktion jag använder i de inledande testerna har följande värden på pjäserna:

Pjäs	***Rangvärde***	***Specialvärde***
Flagga	0	300
Bomb	0	25
Spion	5	35
Spejare	7	10
Minör	10	20
Sergeant	15	0
Löjnant	20	0
Kapten	25	0
Major	35	0
Överste	55	0
General	85	0
Fältmarskalk	125	-30

Tabell 4, Pjäsernas värden

Pjäsernas värden uppdateras inte under sökning, utan ändras bara efter varje faktiskt drag.

Evalueringsfunktionen ger också poäng för följande:

För varje rad som en pjäs (ej spejare) avancerar	1
Om en pjäs (ej bomb) är känd	-(pjäsens värde) * 0.2
Om en bomb är känd	-15

Tabell 5, Förändringar av pjäsvärden

För värdering av okända pjäser används ett genomsnitt av de kvarvarande okända. Om en okänd pjäs slås ut blir det dock problem med denna värderingsmodell. Då måste man också ta hänsyn till värdet på den pjäs som slår ut den okända. Värdeförändringen vid utslagningen blir summan av alla okända som kan slås ut av den aktuella pjäsen.

När alla utslagna pjäser räknas samman adderas en slumpfaktor mellan -2 och 2. Detta för att partierna inte skall fastna i upprepningar lika lätt.

Hittills i avsnittet om evaleringsfunktionen har jag inte gjort någon principiell skillnad mellan agentens egna pjäser och motståndarens, men eftersom agenten alltid känner till sina egna pjäser så tillhör alla okända pjäser motståndaren. När ett evalueringsvärde räknas ut används alltså alltid de riktiga värdena på agentens egna pjäser, även om dessa är okända för motståndaren. Detta gör att sökalgoritmen räknar med att motståndaren känner till vilka pjäser han kommer att slå ut, fast dessa är okända för honom. Detta är att överskatta motståndarens kunskaper! Bättre vore om evalveringsfunktionen returnerade två värden, ett för spelplanen som agenten känner till den och ett på spelplanen som agenten tror att motståndaren känner till den. I den sista varianten räknas även de pjäser som motståndaren inte känner till som dolda. För att detta skall kunna implementeras behöver vi dock en sökalgoritm som stöder att evalueringsfunktionen returnerar två värden istället för ett, precis som den stödjer att en attack kan ha två olika sannolikheter att lyckas.

8. Försök med tre sökalgoritmer

Detta avsnitt innehåller tre försök där olika sökalgoritmer testas. Det som är avgörande för testerna är agenternas spelstyrka, dels i förhållande till varandra och dels genom att de jämförs med en mänsklig strategospelares resonemang. Den viktigaste i testerna är spelstyrka per halvdrag i sökningen, inte per tidsenhet. Att göra algoritmerna snabbare kan göras då man vet vilken som är bäst att använda.

I delavsnitt 8.1 och 8.2 testas två kända algoritmer som ändrats något för att passa *Stratego*. Dessa behandlas kortfattat. Algoritmen i delavsnitt 8.3, perspektivistisk minimax, har utvecklats inom examensarbetet. Den ges därför större plats.

8.1 Vanlig *minimax*

I första försöket används en vanlig *minimax*-sökning. Denna algoritm används vanligen inte för spel med dold information, så den måste modifieras något. I de fall då en händelse kan ske på flera sätt, kommer denna algoritm att anta att det som är mest sannolikt inträffar.

Algoritmen består av en startfunktion och en rekursiv funktion. De ser ut som följer:

```
Function startMinimax(boolean redsturn, int maxply) returns a move
    for each move in generateAllMoves(redsturn) do
        if move.winProbability > 0.5 then
            move.make(win)
        else
            move.make(lose)
        int moveValue = recursiveMinimax(int ply = 1, maxply, !redsturn)
        move.retract()
        int passiveMoveValue = recursiveMinimax(int ply = 1, maxply, !redsturn)
        if (moveValue – passiveMoveValue) > bestMoveValue then bestmove = move
    end
    return bestmove
end
```

Figur 7: Minimaxalgoritmens startfunktion

```
Function recursiveMinimax(int ply, int maxply, boolean redsturn) returns an int
    if ply=maxply then return evaluate()
    for each move in generateLocalMoves(redsturn, lastmove) do
         if move.winProbability > 0.5 then
            move.make(win)
        else
            move.make(lose)
        int moveValue = - recursiveMinimax(ply +1, maxply, !redsturn)
        move.retract()
        if moveValue > bestMoveValue then bestMoveValue = moveValue
    end
    return bestMoveValue
end
```

Figur 8: Minimaxalgoritmens rekursiva funktion

Efter implementering testades algoritmen, med sökdjup fem, mot en motståndare som förflyttade sina pjäser helt slumpmässigt.

Vinster	86
Oavgjorda	14
Förluster	0

Tabell 6: Test av vanlig minimax

En agent som använder denna algoritm spelar väldigt offensivt med sin högt rankade pjäser. Ett anfall med en pjäs med högre rang än sergeant har nästan alltid mer än 50% chans att lyckas, därför görs sådana anfall så fort agenten får chansen. Detta leder till att de högt rankade pjäserna ofta går på bomber och förloras. Eftersom algoritmen bara kalkylerar med ett enda resultat av ett slag, blir strategin enkelspårig: Anfall med de högt rankade pjäserna och dra sig undan med de lågt rankade. Den spelare som bara gör slumpmässiga drag har dock så svag spelstyrka att *minimax*-agenten oftast vinner genom att ta sig igenom flaggans bombskydd med en minör. Detta sedan bomberna upptäcks av anfallande högrankande pjäser.

Minimaxalgoritmens fördelar är:

- Enkel att implementera och att testa
- Lätt och effektivt att klippa i sökträdet (bara att använda vanlig alpha-beta)
- Offensivt spel som kan leda till snabba segrar

Nackdelarna är.

- Förlorar snabbt många högt rankade pjäser
- Spelar överdrivet försiktigt med de lågt rankade pjäserna

En agent baserad på vanlig *minimax* skulle med sin enkelspåriga spelstil vara chanslös mot en mänsklig spelare eller mot de bästa programmen som finns på nätet. Det behövs antingen en mer avancerad sökalgoritm eller ett bättre sätt att använda *minimax* (man skulle kunna tänka sig att man evaluerar slagen istället för pjäserna.

8.2 *Expectiminimax*

Expectiminimax använder slumpnoder för att hantera händelser med okänt resultat. På detta sätt kan en agent se både för- och nackdelarna med ett slag med okänd utgång. Detta blir till priset av en mer avancerad algoritm som har längre exekveringstid.

```
Function startExpectiMinimax(boolean redsturn, int maxply) returns a move
    for each move in generateAllMoves(redsturn) do
        move.make(win)
        winMoveValue = recursiveExpectiMinimax(int ply = 1, maxply, !redsturn)
        move.retract()
        if move.winProbability < 1.0 then
            move.make(lose)
            loseMoveValue = recursiveExpectiMinimax(int ply = 1, maxply, !
redsturn)
            move.retract()
        else
            loseMoveValue = 0
      totalMoveValue = winMoveValue * winProbability + loseMoveValue * (1.0-
winprobability)
        int passiveMoveValue = recursiveExpectiMinimax(int ply = 1, maxply, !
redsturn)
        if (totalMoveValue – passiveMoveValue) > bestMoveValue then bestmove =
move
    end
    return bestmove
end
```

Figur 9: Expectiminimaxalgoritmens startfunktion

```
Function recursiveExpectiMinimax(int ply, int maxply, boolean redsturn) returns an
int
    if ply=maxply then return evaluate()
    for each move in generateAllMoves(redsturn) do
        move.make(win)
        winMoveValue = -recursiveExpectiMinimax(ply + 1, maxply, !redsturn)
        move.retract()
        if move.winProbability < 1.0 then
            move.make(lose)
            loseMoveValue = -recursiveExpectiMinimax(ply + 1, maxply, !redsturn)
            move.retract()
        else
            loseMoveValue = 0
      totalMoveValue = winMoveValue * winProbability + loseMoveValue * (1.0-
winprobability)
        if (totalMoveValue > bestMoveValue) then bestMoveValue =
totalMoveValue
    end
    return bestMoveValue
end
```

Figur 10: Expectiminimaxalgoritmens rekursiva funktion

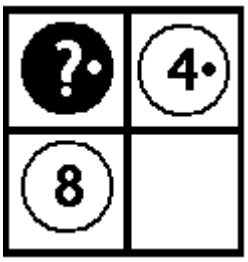
Efter implementering testades algoritmen, med sökdjup fem, mot en motståndare som använde vanlig minimax. Matchserien bestod av 100 partier.

Vinster	56
Oavgjorda	22
Förluster	22

Tabell 7: Test av expectiminimax

Med *expectiminimax* resonerar en agent mer likt en mänsklig spelare än en agent som använder vanlig *minimax*. Vid slag med egna högt rankade pjäser gör algortimen en riskanalys, som leder till att den då undviker att anfalla motståndarpjäser som ännu inte flyttat. Med de lägre rankade pjäserna spelar *expectiminimax* mer offensivt, speciellt om dessa är understödda av en närbelägen starkare pjäs. Dessa strategier är desamma som en mänsklig spelare använder sig av.

Trots dessa fördelar innehåller *expectiminimax* begränsningar som gör att denna agent kommer att göra misstag. I figur 11 visas ett exempel på detta.



Figur 11: Fientlig pjäs med två möjliga slag.

I figur 2 kan den fientliga pjäsen slå på två olika sätt. Antingen riktas slaget mot majoren(4) eller mot minören(8). Enligt *expectiminimax* kommer slaget att riktas mot minören då sannolikheten att vinna över denna är större än sannolikheten att vinna över majoren. I verkligheten är det mer sannolikt att slaget riktas mot majoren då denna flyttat och därför inte kan vara en bomb. Då *expectiminimax* bara använder en sannolikhet, och vid beräkningen av denna använder all känd information, tillskriver den motståndaren information som denna inte har och kommer därför att analysera många drag på ett felaktigt sätt.

Fördelar med *expectiminimax* är:

- Sättet att resonera är likt en mänsklig spelare
- Använder sina pjäser på ett förnuftigt sätt
- Det finns en algortim för att klippa i sökträdet, även om man inte kan klippa bort lika mycket som med vanlig *minimax.*

Nackdelarna är:

- Längre exekveringstid är *minimax*
- Gör misstag då den analyserar motståndarens drag

Expectiminimax skulle kunna användas i en enklare strategoagent om man ändrade den så att den kunde identifiera de situationer där dess fel blir för uppenbara. Det var när jag testade sådana ändringar som jag hittade ett mer generellt sätt att komma till rätta med *expectiminimax* svagheter. Detta ledde fram till en ny algoritm som jag kallar för *p-e-minimax.*

8.3 *P-e-minimax*

P-e-minimax är en utveckling av *expectiminimax* som är speciellt anpassad för att användas i en strategoagent. *P-e-minimax* är en förkortning av perspektivistisk expectiminimax, eftersom algoritmen bygger på att även motståndarens perspektiv måste tas med i beräkningen. I varje löv använder algortimen två evalueringsvärden, ett för agenten själv och ett approximativt för motståndaren. Dessa värden kommer sedan var för sig att påverkas av slumpnoderna, som använder sig av två sannolikheter per nod, ett för agenten och ett för motståndaren. Hur värdena räknas ut beror på omständigheterna när en pjäs slås ut.

Anfallande pjäs (A)	**Försvarande pjäs(B)**	**Eget evalueringsvärde**	**Motståndarens evalueringsvärde**
Egen känd	Fientlig känd	Värdet av B	Värdet av B
Egen känd	Fientlig okänd	Genomsnittet av alla okända som kan slås ut av A	Genomsnittet av alla okända som kan slås ut av A
Egen okänd	Fientlig känd	Värdet av B	Värdet av B
Egen okänd	Fientlig okänd	Genomsnittet av alla okända	Genomsnittet av alla okända
Fientlig känd	Egen känd	Värdet av B	Värdet av B
Fientlig känd	Egen okänd	Värdet av B	Genomsnittet av alla okända som kan slås ut av A
Fientlig okänd	Egen känd	Värdet av B	Värdet av B
Fientlig okänd	Egen okänd	Värdet av B	Genomsnittet av alla okända

Tabell 8: Evaluering av slag, anfallare vinner över försvarare

För att komma fram till motståndarens evalueringsvärden och sannolikheter i slumpnoderna används den information agenten har om sin motståndares pjäser, samtidigt som agentens egna pjäser sätts som okända. Vid konflikt mellan två okända pjäser används formlerna i avsnitt 7.3.

Vilket värde skall då användas för att avgöra vilket drag som är det bästa vid en viss nivå? Detta beror på om sökningen befinner sig i en min-nod eller en max-nod.

Vid en max-nod är det agentens egna drag som analyseras. Här kan värden från två olika drag returneras, om det bästa draget med motståndarens perspektiv blir annorlunda än det

med agentens eget. Om sökningen befinner sig i en min-nod är det dock enbart motståndarens perspektiv som avgör vilket drag som anses vara det bästa, och båda värdena från detta drag returneras.

```
Function startPEMinimax(boolean redsturn, int maxply) returns a move
    for each move in generateAllMoves(redsturn) do
        move.make(win)
        winMoveValue = recursivePEMinimax(int ply = 1, maxply, !redsturn)[1]
        move.retract()
        if move.winProbability[1] < 1.0 then
            move.make(lose)
            loseMoveValue = recursivePEMinimax(int ply = 1, maxply, !redsturn)[1]
            move.retract()
        else
            loseMoveValue = 0
      totalMoveValue = winMoveValue * winProbability + loseMoveValue * (1.0-
winprobability)
        int passiveMoveValue = recurivePEMinimax(int ply = 1, maxply, !redsturn)[1]
        if (totalMoveValue – passiveMoveValue) > bestMoveValue then bestmove =
move
    end
    return bestmove
end
```

Figur 12: P-e-minimaxalgoritmens startfunktion

```
Function recursivePEMinimax(int ply, int maxply, boolean redsturn) returns an int
array of length 2
        if (ply=maxply)
            myEval = evaluate(myBoard)
              oppEval = evaluate(opponentBoard)
            return [myEval, oppEval]
        end
        for each move in generateAllMoves(redsturn) do
            move.make(win)
            winMoveValue = - recursivePEMinimax(ply + 1, maxply, !redsturn)
            move.retract()
            if move.winProbability[1] < 1.0 or  move.winProbability[2] < 1.0
                move.make(lose)
                loseMoveValue = - recursivePEMinimax(ply + 1, maxply, !redsturn)
                move.retract()
            else
                loseMoveValue = [0, 0]
            myMoveValue = winMoveValue[1] * winProbability[1] + loseMoveValue[1]
* (1.0 - winProbability[1])
          oppMoveValue = winMoveValue[2] * winProbability[2] + loseMoveValue[2]
* (1.0 - winProbability[2])
            if (ply%2 = 0)
                 if (myMoveValue) > bestMyMoveValue
                       bestMyMoveValue =  myMoveValue
                if (oppMoveValue) > bestOppMoveValue
                      bestOppMoveValue =  oppMoveValue
          else
                if (oppMoveValue) < bestOppMoveValue
                      bestOppMoveValue = oppMoveValue
                      bestMyMoveValue = myMoveValue
                end
            end
        end
        return [bestMyMoveValue, bestOppMoveValue]
end
```

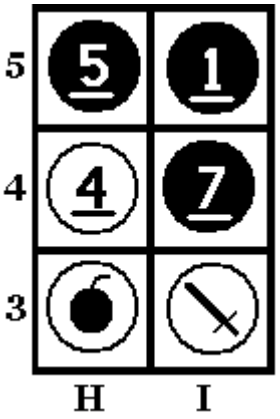
Figur 13: P-e-minimaxalgoritmens rekursiva funktion

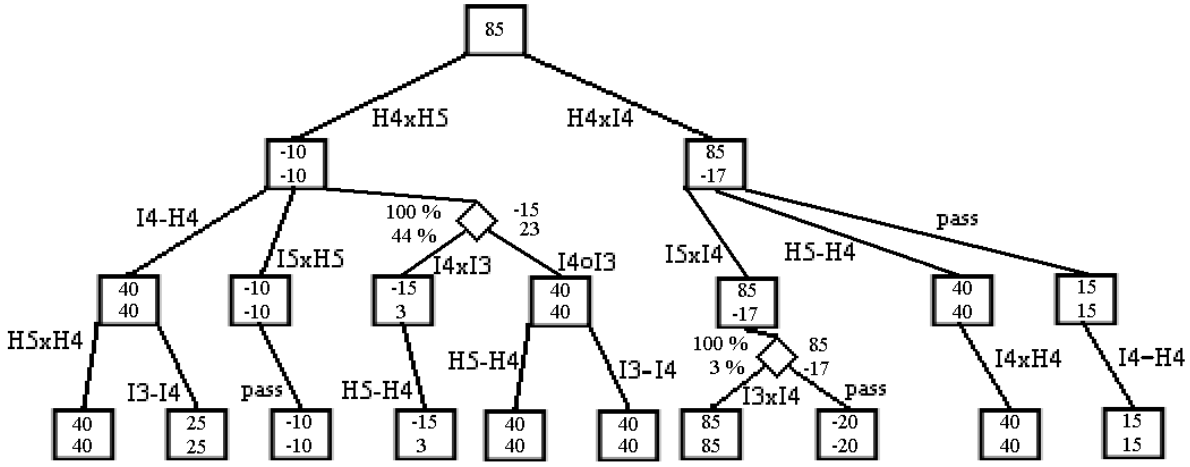
Även denna algoritm testades med sökdjup fem, mot två motståndare. En använde vanlig *minimax* och en *expectiminimax.* Matchserien bestod av 100 partier.

Vinster	53 / 61
Oavgjorda	22 / 19
Förluster	25 / 20

Tabell 9: Test av p-e-minimax. Första siffran anger resultat mot minimax, den andra mot expextiminimax.

P-e-minimax gör en mer omfattande analys än *expectiminimax*, och kommer att försöka utnyttja den information som agenten, men inte dess motståndare, har. Agentens vapen blir alltså de dolda pjäserna som den försöker använda mot de av motståndarens pjäser den känner till.





Figur 14: En ställning och dess sökträd med p-e-minimax

I noderna i figur 14 är det översta värdet det som agenten själv använder, det understa är ur motståndarens perspektiv. De två snedställda kvadraterna är slumpnoder. Många drag (t.ex de flesta passiva) finns inte medtagna. Evalueringen i löven är även den förenklad, här tas enbart hänsyn till pjäsernas omodifierade värden. Dessa är 95 för fältmarkskalk, 35 för major, 25 för kapten, 14 för sergeant och 40 för spion. Okänd pjäs som slås av sergeanten värderas till 22.

I den nedersta slumpnoden ser agenten att spionen kommer att slå fältmarkskalken, men

det finns ändå en slumpnod här för att undersöka motståndarens syn på det hela. Här tror sig agenten kunna lura sin motståndare i en fälla, eftersom den vet att sannolikheten att slå fältmarkskalken är 100%. Rent kombinatoriskt är den bara ca 3%.

De flesta slumpnoder i *p-e-minimax* delar upp sökträdet i två delar, en som visar hur det går om ett slag lyckas, och en som visar hur det går om det misslyckas. Den övre slumpnoden i figur 14 är ett exempel på detta. I slag där den anfallande pjäsen är okänd och den försvarande känd fungerar dock inte detta, eftersom slaget aldrig utförts om det hade misslyckats. Därför räknas ett sådant misslyckat drag som att pjäsen står passiv och inte gör något alls. Den nedre slumpnoden i figur 14 är ett exempel på detta.

Problemet med de misslyckade slagen är ett symptom på att *p-e-minimax* bara är en förenklad lösning på ett mer komplicerat problem. Egentligen är det inte bara själva slagen som beror på den okända informationen, utan också alla andra drag agentens motståndare gör. Egentligen borde varje min-nod vara en slumpnod, där motståndaren gör det drag som minimerar nodens värde med en högre sannolikhet än något annat drag, men inte med 100% säkerhet. En sådan algoritm skulle dock ställa väldiga krav på evalueringen och den funktion man skall använda för att räkna ut sannolikheterna.

Fördelarna med *p-e-minimax* är:

- Klart bättre spelstryka än de andra algoritmerna vid lika sökdjup.

- Algoritmens analys stämmer mer överens med verkligheten än de andras.

Algoritmen har dock en stor nackdel:

- Finns inget känt sätt att klippa i sökträdet vilket begränsar sökdjupet.

En agent som använder med *p-e-minimax* kommer att göra kortsiktigt korrekta drag, men då sökdjupet måste begränsas kan sökningen inte användas för längre dragserier.

P-e-minimax fungerade väldigt bra i testerna, och det verkar som de förenklingar den innehåller inte påverkar dess spel speciellt mycket.

9. Strategoagenten *Perspecto*

I avsnitt 7 behandlades de problem som uppkommer då man skall konstruera en minimaxalgoritm som skall passa för att spela *Stratego*, samt för de alternativ på algoritmer som finns. På grund av de resultatet av de jämförelser som gjordes i avsnitt 8.3 bestämde jag mig för att utgå från algoritmen *p-e-minimax* när jag konstruerar min strategoagent. Agenten hade från början arbetsnamnet *Ape* p.g.a dess (ouppnåeliga) mål att härma en människa. När utvecklingen av själva sökalgoritmen var klar bestämde jag mig dock för att namnet *Perspecto* var mer passande.

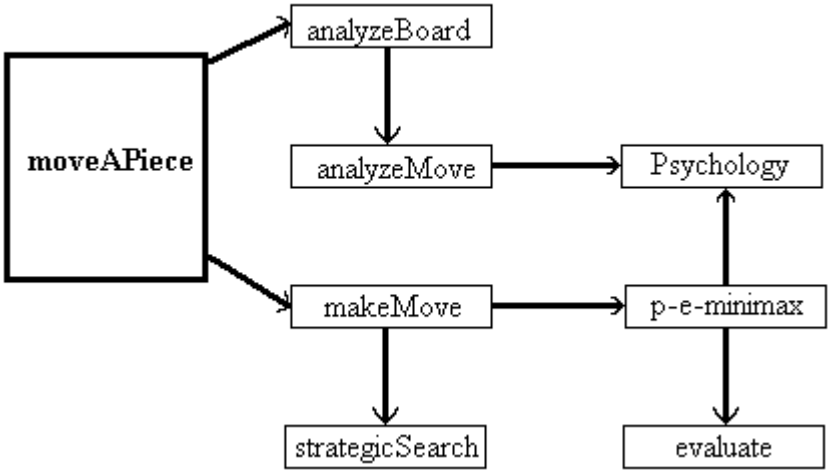
Även om sökalgoritmen, själva stommen i agenten, har testats med lyckat resultat innebär inte det att den spelar tillräckligt bra. I detta avsnitt beskrivs vilka olika förbättringar som kan göras för att komplettera sökalgoritmen.

9.1 Agentarkitekturen

En agent som skall spela checkers eller othello klarar sig bra med enbart en sökalgoritm, en evalueringsfunktion och en samordnare(controller) som styr in- och utmatning samt ser till att rätt information skickas till programmets olika delar. En strategoagent utrustad på detta sätt kommer dock att spela kortsiktigt, speciellt då den som här använder en sökalgoritm med mycket begränsat sökdjup. Man kan beskriva det som att sökalgoritmen har hand om taktiken (utslagning av motståndarens pjäser och skyddande av egna) men måste få hjälp med strategin (att skicka fram rätt pjäser mot motståndaren, och att skydda den egna flaggan).

En annan svaghet är att beräkningen av sannolikheter (se avsnitt 7.3) är kombinatorisk och inte tar hänsyn till att motståndaren har en avsikt med att placera viss pjäs där han gör. P.g.a detta tenderar agenten att bli lättlurad vid spel mot en människa eller mot en någorlunda starkt datorprogram. För att råda bot på denna svaghet är det nödvändigt att utföra en draganalys efter varje drag motståndaren gör, för att dra slutsatser om motståndarens spelstil och därmed göra mer korrekta beräkningar av sannolikheter.

Förutom att sökfunktionen behöver kompletteras med strategi och draganalys kan den också behöva trimmas och förändras. I avsnitt 7 och 8 användes sökfunktioner med fixt sökdjup, men ofta är det bättre att låta sökdjupet vara mer flexibelt beroende på hur intressant det aktuella draget är. På detta sätt kan sökningen bli djupare på de ställen det verkligen behövs.



Figur 15: Perspecto-klassens viktigaste metod moveAPiece och dess hjälpmetoder och hjälpklasser

I figur 15 visas hur *Perspecto* går tillväga för att flytta en pjäs med metoden *moveAPiece.* Denna metod använder i sin tur en rad hjälpmetoder för att ta reda på vilket det bästa draget är.

- **AnalyzeBoard** räknar alla brädets pjäser och förberäknar alla sannolikheter som sedan används i sökningen.
- **AnalyzeMove** tar reda på vilket drag motståndaren just gjort och använder klassen Psychology om draget påverkar de psykologiska variablerna.
- **Psychology** är en klass som har hand om de psykologiska variablerna och ändrar vissa sannolikheter i sökningen.
- **MakeMove** summerar de taktiska och strategiska värden som associerats med de tillåtna dragen för att se vilket drag som är bäst.
- **StrategicSearch** beräknar det strategiska värdet av dragen.
- **P-e-minimax** utför en sökning och beräknar det taktiska värdet av dragen.
- **Evaluate** utvärderar de slag som genomförs av p-e-minimax.

9.2 Strategisk sökning

I *Stratego* är de strategiska besluten ofta lika viktiga som de taktiska. Ett anfall på rätt ställe kan med tur och skicklighet göra att man vinner ett parti man håller på att förlora. Den vanligaste typen av strategiska beslut handlar om att förflytta pjäser från en del av brädet till en annan, där man tror pjäsen skall göra större nytta. Det är inte ovanligt att sådana förflyttningar tar 10 egna drag eller mer, och så långt är inte möjligt att söka.

Problemet med strategi uppstår också i klassiska brädspel som schack, där även de bästa programmen får lita till att deras mänskliga motståndare gör taktiska misstag för att kompensera för programmets bristande strategi. I de klassika brädspelen försöker man ofta bygga in strategi i evalueringsfunktionen. En sådan lösning skulle vara tänkbar även här. Det negativa är att en komplex evalueringsfunktion som tar lång tid att utföra kommer att göra sökningen långsammare. Strategin i *Stratego* är inte heller så invecklad som i t.ex schack, så kanske kan det hela lösas på ett mindre kostsamt sätt.

Vid programmering av robotar används ofta en arkitetur med reaktiva betenden i botten som t.ex ser till att roboten undviker hinder. Ovanpå detta kan man sedan lägga mer sofistikerade betenden, och därmed skapa en hybrid arkitektur.[13] På samma sätt kan man här lägga in en enkel strategisk sökning som styr lågnivå-beteendet, och ovanpå det en minimaxalgoritm som sköter de taktiska finesserna.

Den strategiska sökningen tittar enbart två egna drag framåt i en global sökning, så den är snabb i jämförelse med *minimax*sökningen. En fördel med denna metod är att den kan kombineras med spejarnas långa slag, som kommer bort i minimaxsökningen då den bara tittar på lokala drag.

Den strategiska sökningen börjar med att man för varje flyttbar pjästyp utforskar till vilka rutor på spelplanen denna pjäs skulle vilja gå, och vart den inte skulle vilja gå. Detta sker med en rad strategiska regler[14], t.ex

[8,fB,8] = Minör(8) dras till fientlig bomb(fB) med kraft 4
[7,fO,1] = Sergeanten(7) dras till fientliga okända (fO) med kraft 1
[5,f6<,-3] = Kaptenen(5) dras till fientliga pjäser med högre rang än 5(5<) med kraft -3

Dragningskraften på en pjäs räknas sedan ut med

$$\frac{1}{\sqrt{L^t}} \cdot kraft$$

Där L är antal steg mellan pjäserna (lägst 4) och t = antal pjäser motståndaren har kvar/10, dock lägst 1.

Alla dragningskrafter som verkar på en pjäs delas sedan upp i x- och y-koordinater och summeras. En positiv kraft i en riktning kommer alltid att motsvaras av en negativ i en annan rikting och tvärt om. På detta sätt tillskrivs varje drag en posiviv eller negativ poäng.

Spejarnas långa slag får poäng beroende på pjäsen de skall slå. De får ett poäng för varje misstanke om att deras mål är en flagga (se avsnitt 6.1.5) eller 0.1 poäng för varje steg målet tagit. Dessutom blir slaget värt 0.1 poäng per ruta spejaren flyttar.

13 Se [2] s 257-265
14 En lista på samtliga strategiska regler finns i appendix 1

```
Function StrategicSearch(booelan redsturn) assigns strategic values to all moves
    for each move in generateAllMoves(redsturn) do
        int force = move.getStrategicScore()
        move.make(win)
        bestforce=force
        for each move in generateAllMoves(redsturn) do
            int force2 = move.getStrategicScore()
            if force2 > bestforce
                bestforce=force2
        end
        move.retract()
        move.assignStrategicValue(force+bestforce)
    end
end
```

Figur 9: Strategisk sökning

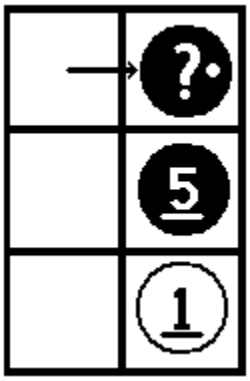
När alla poäng för draget självt är beräknade ger den strategiska sökningen ytterligare poäng genom att addera poängen för bästa efterföljade drag. Hittas inget drag som är bättre än draget självt adderas dess egen poäng istället. Denna funktion lades till för att agenten skall flytta bort pjäser som blockerar ett bra strategiskt drag.

De drag som innebär slag mot andra pjäser sorteras bort ur den strategiska sökningen. De strategiska poäng som genereras är också betydligt lägre än de som fås fram ur *minimax*-sökningen. T.ex värderas en framryckning med sergeant i början av spelet till ca 1, medan en minör rycker fram mot en bomb med en tillhörande misstänkt flagga (sannolikhet 1/8) på 6 rutors avstånd med en extra kraft på ca 4. Detta gör att den strategiska sökningen får betydelse enbart då inga bra taktiska drag hittas.

9.3 Pjäsanalys och draganalys

Genom att beräkna sannolikheter för attacker som involverar okända pjäser kan man göra om *Stratego* från ett spel i en dold miljö till ett spel i en icke-deterministisk miljö. Dessa båda miljöer har mycket gemensamt, eftersom alla slumpmässiga händelser är en form av dold information. Det motsatta förhållandet är dock mer osäkert: Är all dold information slumpmässig?

I *Stratego* är den dolda informationen ett resultat av de val som motståndaren gör. Dessa val görs inte slumpmässigt, en bra spelare har en tanke med hur uppställningen av pjäserna görs och hur de okända pjäserna flyttas. I figur 16 visas en ställning där det är svårt att ställa upp en sannolikhet för värdet på en okänd pjäs.



Figur 16: Är den okända pjäsen en spion?

En agent som bara räknar sannolikheter kombinatoriskt kommer att sätta sannolikheten att pjäsen i figur 16 är en spion till ca 3 %, och kommer att slå kaptenen med sin fältmarkskalk utan tvekan. Detta kan en smart motståndare utnyttja för att lura agentens kända pjäser i fällor. Vissa problem i *Stratego* handlar precis som i poker om att bluffa, syna och lägga sig. Problem som dessa är svåra för datorer att lösa då de har en psykologisk dimension som är svår att programmera.

För att kunna handskas med dessa psykologiska problem behövs en psykologisk modell. För att definiera en sådan modell behöver man först göra vissa antaganden som förenklar den:

- Motståndarens uppställning är helt slumpmässig, med undantag för placeringen av flaggan.

- En spelare använder enbart försåtliga drag då han närmar sig en motståndarpjäs som han vet värdet på. I alla andra fall kan rent kombinatoriska modellen användas.

Dessa två antaganden gör att man måste ta hänsyn till den psykologiska modellen då man i en sökning stöter på en konfrontation mellan en känd och en okänd pjäs, och enbart då den okända pjäsen har flyttats mot den egna pjäsen efter att denna blev känd. Den kända pjäsens sannolikhet för seger kommer då att förändras till det sämre, jämfört med vad som beräknats kombinatoriskt. Formeln för den slutliga sannolikheten blir

$$Py = P - P \cdot psy$$

Där *psy* är en av tre variabler i den psykologiska modellen, med ett värde mellan 0 och 1. Vilken av variablerna som används bestäms av de två inblandade pjäserna.

- Då en egen känd pjäs är inblandad i ett slag mot en av motståndarens okända, ökas sannolikheten för att den egna pjäsen skall förloras med variabeln *ownFear*. Detta är ett mått på hur försiktig agentens spelstil är. 0 innebär att agenten tror att motståndaren

skickar fram pjäser slumpmässigt, medan 1 innebär att agenten tror att varje okänd pjäs som skickas fram har högre rang än de egna kända pjäserna.

- Det är rimligt att även tillskriva motståndaren någon form av försiktighet. Detta görs med variabeln *estOppFear,* som fungerar på samma sätt som *ownFear* ovan. *EstOppFear* används då en känd motståndarpjäs är inblandad i slag mot en av agentens okända pjäser.

- Man kan anta att motståndaren i sin tur gör sig en uppfattning om agentens försiktighet. Denna uppfattning bestäms av variabeln *estOwnFear* och behöver inte stämma med det verkliga värdet. *EstOwnFear* används vid beräkningar ut motstådarens perspektiv istället för *ownFear.* Variabeln *estOwnFear* är alltså en uppskattning av motståndarens uppskattning av agentens försiktighet!

Två av variablerna är relativt enkla att bestämma. *Ownfear* sätts beroende på hur offensiv man vill att agentens spelstil skall vara. Ett lågt värde innebär en respektlös inställning medan ett högt värde indikerar en defensiv hållning. Mot en mänsklig spelare, eller mot en avancerad agent som kan läsa av motståndarens spelstil, kan det vara en bra idé att variera *ownFear* för att förvirra motståndaren, t.ex kan variabeln anta ett värde mellan 0.2 och 0.5. *EstOwnFear* kan också bestämmas enkelt. Ett rimligt antagande är att motståndarens bedömning av agentens försiktighet är ett mellanting av agentens faktiska försiktighet och motståndarens försiktighet.

EstOppFear är ännu svårare att bestämma. Här måste man hitta på ett sätt att läsa av vilken spelstil ens motståndare använder sig av. För att detta skall vara möjligt måste motståndarens samtliga drag analyseras. I detta projekt kommer en enkel draganalys att användas:

- Det inledande värdet på *estOppFear* är 0.3. Det kan aldrig understiga 0.0 eller överstiga 1.0.

- Om motståndaren attackerar en okänd pjäs som flyttat med en egen känd pjäs som har högre rang än löjtnant, minska värdet med 0.04.

- För varje attack agenten gör med okänd pjäs som flyttats mot en av motståndarens kända pjäser, minska värdet med 0.02.

- För varje möjlig attack som motåndaren kan göra med en känd pjäs med högre rang än löjtnant mot en okänd som flyttats, men som han avstår ifrån att göra, öka värdet med 0.04. Detta görs endast en gång per pjäspar under spelets gång.

Med ovanstående draganalys kommer agenten inledningsvis att försöka lura sin motståndares kända pjäser i fällor med högre rankade egna pjäser. Om motståndaren då

drar undan sina pjäser kommer *estOppFear* att öka, och agenten kommer istället att bluffa oftare för att jaga bort hotande motståndarpjäser.

Om man räknar noggrannare på ställningen i figur 17 ser man att vinsten för att slå kaptenen med fältmarkskalken är 25 om den okända pjäsen ej är spion och - 70 om den är spion. För att *Perspecto* skall vilja slå måste sannolikheten för vinst vara minst 0.74. Hur stor skall då *ownFear* vara för att Perspecto skall välja att avstå från slaget? Om man sätter in värdena i formeln ovan får man 0.74 = 0.97 – 0.97*ownFear,* d.v.s att *ownFear* skall vara minst 0.24 för att fältmarkskalken skall hålla tillbaka slaget.

9.4 Förbättringar av sökningen

Den strategiska sökningen, som behandlas i avsnitt 9.2, är till för att hjälpa *Perspecto* att flytta pjäserna till rätt positioner på planen. Därför kan evalueringsfunktionen i den vanliga sökningen koncentrera sig på att enbart utvärdera material, d.v.s vinst eller förslust av pjäser. I och med detta blir det onödigt att evalueringsfunktionen tittar på hela spelbrädet. Det intessanta blir enbart de gånger i sökningen då en pjäs blir utslagen. När detta sker sparas de två inblandade pjäserna i en lista som tas om hand av evalueringsfunktionen när sökningen bottnar. I listan värderas pjäser som slagits ut tidigt i sökningen högre en de som slås ut sent, vartannat halvdrag minskar värdena med 5%. Detta är för att få Perspetco att agera även om ett slag kan skjutas upp ett antal drag.

En annan förbättring gäller bedömningen av motståndarens drag i min-noderna. Ibland kan två eller flera drag vara lika bra, men p-e-minimax väljer ändå ut ett vilket leder till misstag. Därför är det bättre att låta alla motståndardrag som ligger inom tre poäng från det bästa vara lika sannolika och låta ett medelvärde av dessa drag skickas uppåt i trädet.

I de inledande försöken med olika sökalgoritmer i avsnitt 8 användes ett fixt sökdjup. Detta är dock ett oekonomiskt sätt att söka, eftersom vissa delar av sökträdet kommer att vara intressantare än andra. Dessutom riskerar sökningen att avbrytas i positioner där nya slag är möjliga och ställningen därför kommer att förändras drastiskt. För att undkomma detta använder *Perspecto* ett variabelt sökdjup som bestäms av följande regler:

- Halvdrag på djup 5 utförs bara om de är slag, flyttar en pjäs så att den kan bli slagen, flyttar den senast flyttade pjäsen eller är passiva.

- Halvdrag på djup 6 – 8 utförs bara om de är slag, om de flyttar den senast flyttade pjäsen eller är passiva.

- Halvdrag på djup 9 utförs bara om de är slag.

Trots det utökade sökdjupet används lokal sökning där sökområdet har radie 2.

10. Resultat

Den viktigaste egenskapen hos en brädspelande agent är dess *spelstyrka*. Spelstyrka är ett generellt mått på hur bra agenten spelar. Ju mer komplext ett spel är, destå svårare är det att mäta spelstyrkan, eftersom sannolikheten då är större för att en grupp spelare "slår varandra" d.v.s att deras olika spelstilar fungerar olika bra mot olika motståndare. Detta kan också vara fallet i *Stratego* , t.ex är det lätt att med hjälp av de psykologiska varablerna i *Perspecto* konstruera tre sådana spelare. I praktiken är det dock möjligt att genom tester mot flera olika motståndare göra en realistisk bedömning av en agents spelstyrka.

I detta avsnitt kommer tre olika tester av *Perspecos* spelstyrka att presenteras: ett parti mot en mänsklig motståndare, en serie om tio partier mot en annan agent och ett kvantitativt test mot alternativa versioner av *Perspecto*.

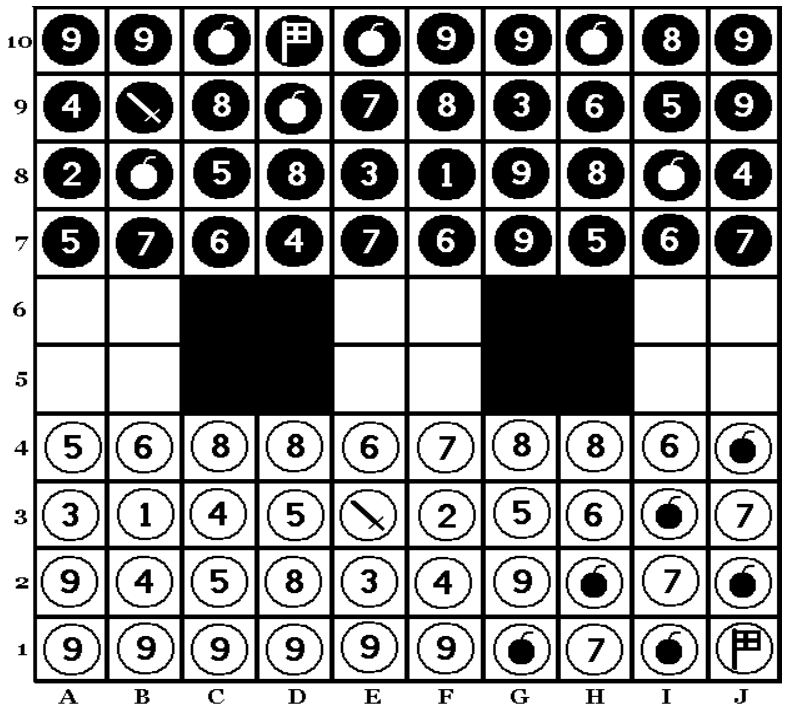
10.1 Spel mot en mänsklig motståndare

De flesta vuxna människor som spelar *Stratego* befinner sig på nivån strax över nybörjarstadiet. Ofta har man börjat som barn och sedan fortsatt spela då och då. I Tyskland och Holland finns strategoföreningar och turneringar spelas regelbundet, men i Sverige finns ingen sådan verksamhet. De flesta spelar helt enkelt för att det är kul.

När jag själv började spela *Stratego* var jag 11 år och spelade mest mot en kompis, Peter. Än idag händer det att vi tar ett parti. När *Perspecto* skulle testas undrade Peter om han kunde bli försöksperson och jag tyckte att han passade bra. Peter har spelat ungefär 100 partier *Stratego* i sitt liv, kanske hälfen av dem mot mig. Enligt egen utsago har han vunnit ca 60 % av dem. Han har dock aldrig tidigare spelat mot ett datorprogram.

Notationen som används är densamma som i schack fast med versaler som markerar kolumn. Ett lyckat anfall noteras med ett "x" medan ett misslyckat skrivs "o". Ett anfall där båda pjäserna slås ut noteras "xo". I bilderna på spelplanen visas en känd pjäs med ett streck under sitt värde och en pjäs som flyttat med en prick till höger på pjäsen.

Peter lottades att spela vit, och kommer att göra första draget.



Figur 17: Startuppställningen Peter vs Perspecto

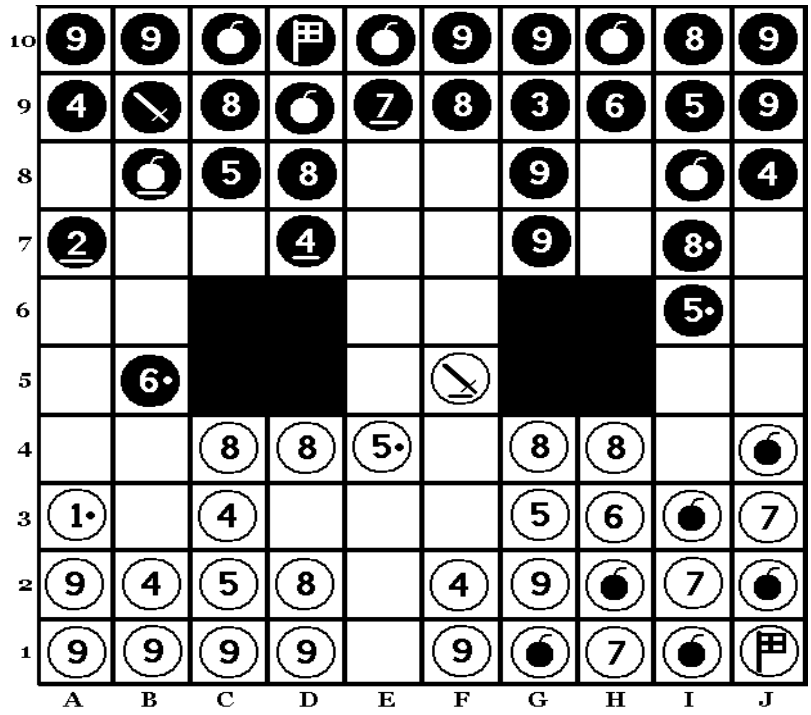
Peter: Min plan är inspirerad av den tyska schlieffenplanen för angreppet i väster under första världskriget. Alla de starka pjäserna är redo att gå till anfall i centern och på den vänstra flanken. De skall mötas på 7:e och 8:e raden bakom den vänstra sjön och sedan fortsätta mot rad 9 och 10. Minörerna och spejarna ställs upp i reserv och kan kallas in vid behov.

Den högra flanken är statisk och flaggan är skyddad av ett dubbla rader av bomber. Denna typ av försvar kan vara sårbart mot en mänsklig motståndare som ofta misstänker just detta försvar när han möter bomben på J4, men det borde vara utmärkt mot en dator.

Perspecto: Uppställningen är "halvslumpmässig", genererad enligt delavsnitt 7.1. Detta innebär att flaggan står skyddad och att de högsta pjäserna slumpas ut på rad 8 och 9. Vidare står de flesta spejarna och minörerna långt bak medan de mellanhöga pjäserna står långt fram.

Perspectos psykologiska variabler sätts enligt standardvarianten som beskrivs i delavsnitt 9.3. *OwnFear* är ett slumpvärde mellan 0.2 och 0.5. *EstOppFear* är 0.3 från början och *estOwnFear* hamnar mitt emellan de andra två variablerna.

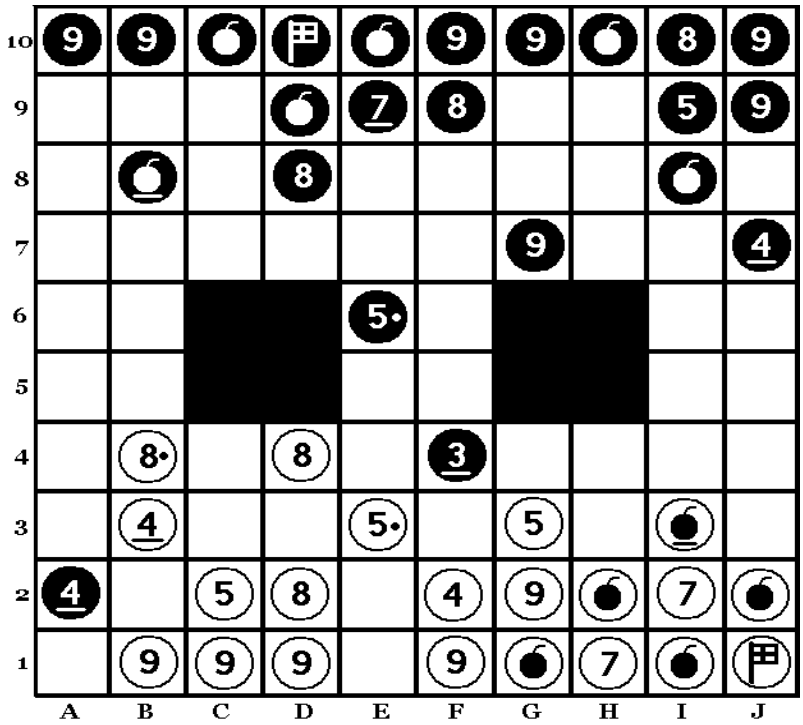
Drag	Peter	Perspecto	Kommentar
1	E4-E5	J7-J6	
2	F4-F5	B7-B6	
3	F5-F6	F7xF6	
4	E5-E6	F6-F5	Här slumpas till *ownFear* 0.42, ett högt värde. Ett lågt värde hade resulterat i F6xoE6.
5	E6xE7	D7xE7	
6	B4-B5	B6oB5	
7	A4-A5	F5-F4	
8	F3xF4	8F-7F(?)	Peter väljer här att avslöja sin general hellre än att riskera sin minör på G4. Perspectos svarsdrag beror på den strategiska regeln *[1,f=2,32]*, som gör att fältmarkskalken vill närma sig generalen. Peter misstänker direkt vilken pjäs han har mot sig.
9	B5-B6	A7-A6	
10	B6-B7	A6xoA5	
11	B7oB8	F7-F6	
12	A3-A4	F6-F5	*Perspecto* har nu möjlighet att jaga ifatt Peters general, men detta ligger 10 halvdrag framåt och bakom Perspectos horisont. F6-F5 beror åter på den strategiska sökningen.
13	F4-F3	F5-F4	*EstOppFear* ökas till 0.36, p.g.a att generalen glider undan.
14	A4-A5	F4xF3	*Perspecto* slår Peters general. Eftersom inga av pjäserna runt generalen har flyttat, misstänker *Perspecto* ingen spion där. Den kombinatoriska sannolikheten är alltför liten för att den skall spela in.
15	E3xF3	I7-I6	Perspectos fältmarkskalk slås av Peters spion.
16	E2-E3	J6-J5	
17	E3-E4	J5oJ4	
18	E4-E5	I6-I5	
19	E5-E6	E7-D7	Trots ett relativt lågt värde på *ownFear*, 0.31, vill Perspecto inte riskera sin major mot den okända pjäsen. Han hoppas istället slå den med sin överste.
20	E6-E7	E8oxE7	Båda spelarna förlorar varsin överste.
21	E1oE9	I5xoI4	
22	A5-A6	A8-A7	
23	A6oA7	C7-B7	Peter förlorar en överste, ett högt pris för att få avslöja perspectos general.
24	F3-F4	B7-B6	
25	B3-A3	H7-I7	
26	D3-E3	H8-H7	
27	E3-E4	I7-I6	
28	F4-F5	H7-I7	



Figur 18: Ställningen efter drag 28

Drag	Peter	Perspecto	Kommentar
29	C3-B3	I7-J7	Notera hur Perspectos strategiska sökning har gjort att kaptenen på I6 släppt fram minören.
30	B3-B4	B5oB4	
31	A3-A4	J7-J6	
32	A4-A5	J6-J5	
33	A5-A6	A9-A8!!	OwnFear slumpas till 0.40, men även vid 0.20 slår inte *Perspecto* i detta läge. *Perspecto* sökning kan se att generalen kommer att slås om den försöker fly till B7 (slaget ligger på djup 8). Eftersom *estOppFear* har ett såpass högt värde som 0.36 ser Perspecto en utväg: en bluff!
34	A6-A5	J5xJ4	Peter går på bluffen: A8 verkar trolig som spion, speciellt som den ställt upp intill generalen. Men han har en annan plan...
35	J3xJ4	I6-I5	
36	B4-B5	I5-I4	
37	J4-J5	I4oI3	
38	B5-B6	H9-H8	
39	A5-B5	H8-H7	

Drag	Peter	Perspecto	Kommentar
40	B6-B7!	A7xB7	Genom att offra sin major lyckas Peter får en chans att avslöja Perspectos "spion".
41	A2oA8	H7-I7?	Här tror *Perspecto* att Peter skall gå att lura på samma sätt igen...
42	B5-B6	C8-C7	
43	B6xB7!	C7-C8	Denna gång synar Peter bluffen.
44	B7-A7	A8-A9	
45	A7-A8	I7-I6	
46	A8xA9	B9xA9	Denna gång stod spionen på rätt ställe. Perspecto har nu uppnått överlägsenhet med sin överste på G9 och med sina majorer. Dessa kan nu ignorera *ownFear* när de flyttar, och nya strategiska regler kopplas in för att få dem att uppträda aggresivt.
47	A1xA9	I6-I5	
48	J5-J6	I5-I4	
49	J6-J7	J8xJ7	
50	F5-F6	I4xH4	
51	H3xoH4	G8-F8	
52	F6-F7	F8xF7	
53	G4-F4	G9-G8	
54	A9-B9	C9xB9	
55	B2-B3	G8-F8	Peter vill byta majorer mot majorer, så att han bättre kan utnyttja sin övervikt i kaptener. Perspectos överste kan han bara undvika.
56	B3-B4	F7oF4	
57	B4-B5	F8-F7	
58	B5-B6	F7-F6	
59	F4-F5	F6xF5	Perspectos överste avslöjas.
60	E4-E3	F5-F4	
61	B6-B7	D7-C7	
62	B7-A7	C7-B7	
63	A7-A8	B7-A7	
64	A8-A9	A7-A8	
65	A9xB9	A8-A7	
66	B9-C9	C8-C7	
67	C9-C8	C7-D7	
68	C8-C7	D7-E7	
69	C7-B7	A7-A6	P.g.a rangvärdet i evalueringsfunktionen (delavsnitt 7.5) vill inte *Perspecto* byta majorer.
70	B7-B6	A6-A5	
71	B6-B5	A5-A4	
72	B5-B4	A4-A3	
73	B4-B3	A3-A2	
74	C4-B4	E7-E6	



Figur 19: Ställningen efter drag 74

Drag	Peter	Perspecto	Kommentar
75	B4-B5	E6-E5	Peter är i en svår situation. Den enda öppna flanken är den vänstra, så Peter skickar fram en minör där.
76	B5-B6	F4-F3	
77	E3-E4	E5xoE4	
78	B6-B7	J7-I7	
79	B7xB8	D8-C8	Här upptäcker *Perspecto* hotet mot flaggan.
80	B8-B9	C8-C9	
81	B9xB10	J9oJ2	
82	B10xC10	C9xoC10	
83	D4-C4	I9-H9	
84	C4-B4	I10-I9	Perspectos flagga är väldigt hotad, men *Perspecto* skickar fram minörer mot Peters högerflank.
85	B4-B5	I9-J9	
86	B5-B6	J9-J8	

Drag	Peter	Perspecto	Kommentar
87	B6-B7	J8-J7??	Här förlorar *Perspecto* partiet. Den strategiska sökningen kan inte hitta något drag som skyddar den sårbara flaggan, och förlusten ligger 10 halvdrag bort, så den taktiska sökningen kan inte se den. E9-E8 var det enda drag som kunde räddat *Perspecto*.
88	B7-B8	A10-A8?	Horisonteffekt. *Perspecto* ser att flaggan är hotad, detta är det enda drag som kan få flaggans fall bakom horisonten. Draget gör också att Peter inte har några tvivel om var flaggan är.
89	B8-B9	A8-A9	
90	B9-B10	J7-J6?	Här skulle en mänsklig spelare chansat, kanske är Peters flagga på 1B eller 1F? Eftersom Perspectos sökning är lokal, påverkar inte situationen vid den egna flaggan dess offensiv.
91	B10-C10	J6-J5	
92	C10xD10		Perspectos flagga slås ut.

Efter partiet tyckte Peter att *Perspecto* spelat bra i början av partiet, men att spelet blev statiskt mot slutet. Han säger att han antagligen förlorat partiet om *Perspecto* skyddat flaggan bättre. Peter tror att hans eget spel troligen blivit bättre om han fått fler partier på sig att lära sig Perspectos spelstil.

Förlusten mot Peter var ingen tillfällighet, även om *Perspecto* kunde vunnit om händelserna tagit en annan vändning. Om man istället för att spela en match hade spelat en serie om tio, hade *Perspecto* kanske vunnit ett par matcher, men majoriteten hade förlorats. Dessutom skulle den mänskliga spelaren lära sig betydligt mer om sin motståndare från match till match än vad *Perspecto* med sin relativt primitiva draganalys kan.

Orsaken till förlusten i just detta parti är den strategiska sökningen. Denna är gjord för att få *Perspecto* att gå till anfall, och skulle man göra en sökning som även uppmanade till försvar finns en risk att pjäserna skulle bli passiva. En lösning skulle kunna vara att viga en viss typ av pjäs, t.ex kaptenen, till defensiva uppgifter.

Perspectos psykologiska variabler fungerade bra i denna match, med den framgångsrika bluffen i drag 33 som höjdpunkt. Dock behövs det en bättre draganalys, som t.ex kan tala om huruvida tidigare bluffar avslöjats eller ej.

Sammanfattningsvis kan man säga att *Perspecto* spelar tillräckligt bra för att ge de flesta mänskliga spelare en bra match, iallafall i de första partierna.

10.2 Spel mot en annan agent

De två bästa strategoagenter man kan hitta på nätet är *Reveal-your-rank!* av Raimonds Rudmanis[15] och *The General* av Sean O'Connor[16]. Demoversioner av dessa program finns att ladda ner gratis. Efter att ha testat de båda programmen kom jag fram till att *The General* var det klart starkaste. *The General* använder sig av minimax precis som *Perspecto*, medan *Reveal-your-rank!* bygger på ett expertsystem. För att ta reda på mer om *The General* gjorde jag en kort intervju med utvecklaren Sean O'Connor.

```
Kalle: What kind of algorithms are your AI based on?

Sean: It's basically a minimax routine with alpha-beta cutoff but you
have to be clever as it needs to think about 8 moves ahead and that
gives you a vast number of possibilities if you're not careful.

Kalle: How strong do you consider your AI to be vs humans and vs other
programs? Have you tested it aganist another AI?

Sean: I think a competent human will still win most of the time as it's
not a game that suits AIs very well as pieces are hidden. But, (IMHO !)
I don't think other Stratego AIs that I have seen come close to my one
and they seem to play an almost random game!

Kalle: How do the program handle the unknown information in the minimax
search tree?

Sean: It try to guess what your piece is. If it's moved it can't be a
bomb or a flag, if it's moved aggressively it might be a stronger piece
etc...

Kalle: Do the program search through all moves on the board, or just
local ones?

Sean: It looks for local moves and long range(scout) moves too.

Kalle: How deep does the search usually go before it is terminated?

Sean: It looks ahead 12 moves. I thought that would be good enough for
even moving a sapper(minör) most of the way diagonally across the board
to get to a bomb.
```

Det är helt klart så att *The General* använder en vanlig minimaxalgoritm och att detta tillåter betydligt djupare sökning än *Perspecto.* Det verkar också vara så att *The General* använder en mer avancerad pjäsanalys för att kompensera för den enklare sökningen.

För att få en uppfattning om de båda agenternas spelstyrka spelades en matchserie om 10 partier. Perspectos draganalys sparades inte mellan de olika partierna.

15 http://www.yellowgames.com/
16 http://www.windowsgames.co.uk/thegeneral.html

	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***Totalt***
Perspecto	1	1	1	1	1 /2	1	1	1 /2	0	1	8
The General	0	0	0	0	1 /2	0	0	1 /2	1	0	2

Tabell 10. Resultat av matchserien Perspecto - The General

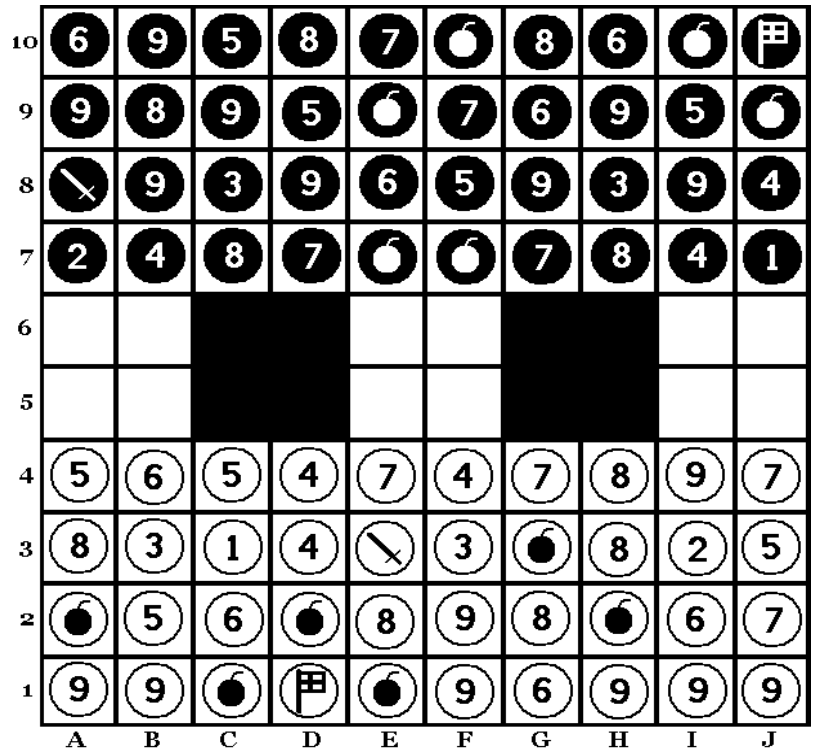
Sju av partierna slutade med seger för *Perspecto*, två slutade oavgjort och ett vanns av *The General*. De flesta partierna varade ungefär 100 drag och kunde vinnas av *Perspecto* genom att slå ut motståndarens flagga med en minör. Detta efter att materiell överlägsenhet uppnåtts i mitten av partiet.

Parti nr 4 var ett av de bättre partierna. Här använder *The General* en okonventionell startuppställning (programmet använder ett bibliotek av ca 40 uppställningar) som gör att *Perspecto* inledningsvis förlorar många pjäser.

The General: Den spelartyp som används är "Patrick" som enligt Sean O'Connor är den generellt bästa (*The General* har åtta olika spelartyper). Uppställningen som används är "center block" som placerar de flesta bomberna i mitten. På flankerna sätts de högt rankade pjäserna i frontlinjen.

Perspecto: Uppställningen är "halvslumpmässig", genererad enligt delavsnitt 7.1. Detta innebär att flaggan står skyddad och att de högsta pjäserna slumpas ut på rad 8 och 9. Vidare står de flesta spejarna och minörerna långt bak eller vid sjöarna medan de mellanhöga pjäserna står långt fram.

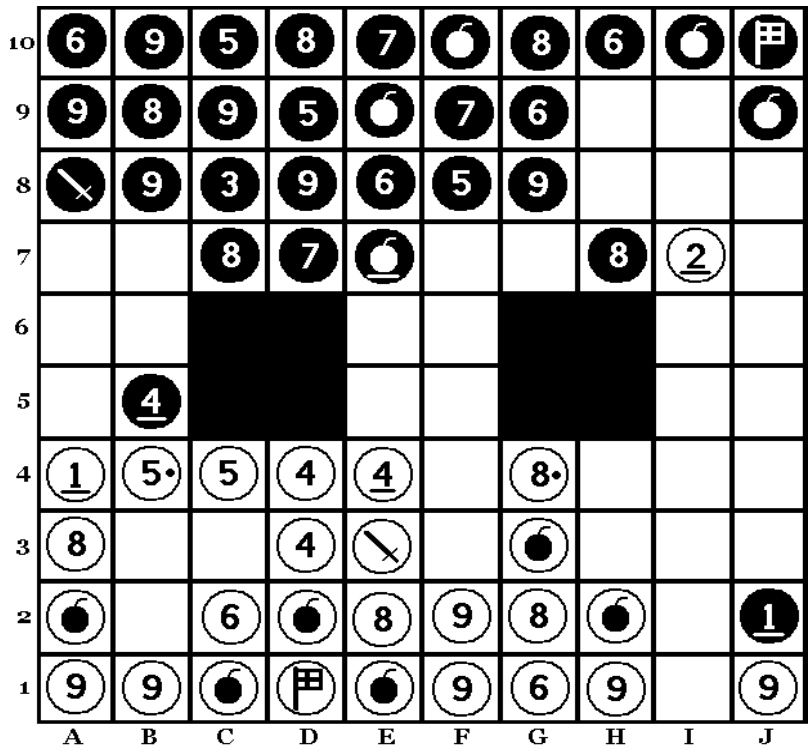
Perspectos psykologiska variabler sätts enligt standardvarianten som beskrivs i delavsnitt 9.3. *OwnFear* är ett slumpvärde mellan 0.2 och 0.5. *EstOppFear* är 0.3 från början och *estOwnFear* hamnar mitt emellan de andra två variablerna.



Figur 20: Startuppställningen, Perspecto vs The General

Drag	Perspecto	The General	Kommentarer
1	B4-B5	A7-A6	
2	B3-B4	A6-A5	
3	E4-E5	A5xB5	
4	B4-B3	B5-B4	
5	J4-J5	B4xB3	
6	C3xB3	J7-J6	Perspectos fältmarkskalk slår The Generals general, dock till priset av en överste och en löjtnant.
7	J5oJ6	I7-I6	*Perspecto* upptäcker The Generals fältmarkskalk.
8	A4-A5	I6-I5	
9	A5-A6	B7-A7	
10	A6oA7	I5-J5	
11	E5-E6	I8xoI4	
12	E6oE7	J5-J4	
13	J3oJ4	J6-J5	
14	F4-E4	J4-I4	
15	I3xI4!	J5-I5	*Perspecto* kan inte se någon fara för generalen och slår därför ut majoren, ett bra drag.
16	I4-J4	A7-A6	
17	G4-F4	A6-A5	

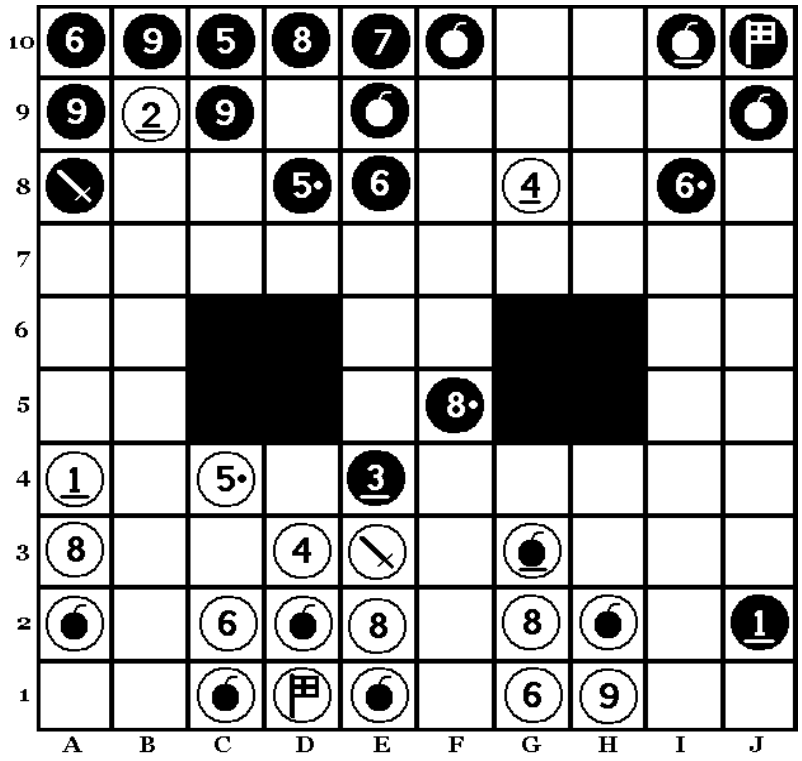
Drag	Perspecto	The General	Kommentarer
18	F4-F5	A5-A4	
19	F5-F6	I5-J5	
20	J4-I4	J5-I5	
21	I4-J5	J8-J7	
22	F6oF7	I5-J5	På grund av The Generals ovanliga uppställning har Perspectos inledande förluster varit stora.
23	J4-I4	J7-J6	
24	H4-G4	I9-I8	
25	G4-F4	J5-J4	
26	I4-I5	J6-J5	
27	F4-F5	I8-I7	
28	F5-F6	A4-A5?	*The General* gör ibland drag som det är svårt att se någon mening med.
29	F6xF7	G7xF7	
30	J2-J3	J4-I4?	*The General* tappar sitt skydd av sin major, som slås ut.
31	I5xJ5	F7-F6	
32	J5-J6	I4-I3	
33	J3-J4	I3xI2?!	The Generals fältmarkskalk slår en stationär pjäs. Detta är en stor risk att ta, men denna gång finns här ingen bomb utan en löjtnant.
34	J4-J5	I2xI1?!	Denna gång slås en av Perspectos spejare ut.
35	J6-J7	I7-I6	
36	J5-I5	F6-F5	
37	I5oI6	F5-F4	
38	E4xF4	I1-I2	
39	J7-I7	I2-I3?	*The General* lämnar en kapten helt åt sitt öde.
40	I7xI6	I3-J3	
41	B3-B4	G8-G7	
42	B2-B3	G7-F7	
43	F4-E4	F7oF3	
44	B4-A4	A5-B5	
45	B3-B4	H8-G8	
46	I6-I7	G8-G7	
47	F3-F4	G7-F7	
48	F4-F5	F7-F6	
49	F5xoF6	H9-H8	Eftersom båda sidors högsta pjäser avslöjats, kan överstarna agera utan rädsla för okända pjäser. Här slår de varandra.
50	H3-H4	J3-J2	
51	H4-G4	H8-G8	



Figur 21: Ställningen efter drag 51

Drag	Perspecto	The General	Kommentarer
52	G4-F4	G8-J8	
53	F4-F5	G9-G8	
54	F5-F6	G8-G7	
55	F6-E6	G7-F7	
56	F2oF7	H7-G7	
57	E4-E5	B5xB4	
58	A4xB4	H10-H9	
59	I7-H7	G7-G8	
60	H7-G7!	J2-J3	Med en “gaffel” hotar Perspectos general två pjäser.
61	G7xG8	J3-J4	*The General* gör många drag med sin fältmarkskalk längs J-linjen, långt bort från händelsernas centrum.
62	G8-G7	F7-F6	
63	E6xE7	D7xE7	
64	G7-F7	J4-J5	
65	F7xE7	H9-H8	
66	E5-F5	J5-J4	
67	F5xF6	J4-J3	

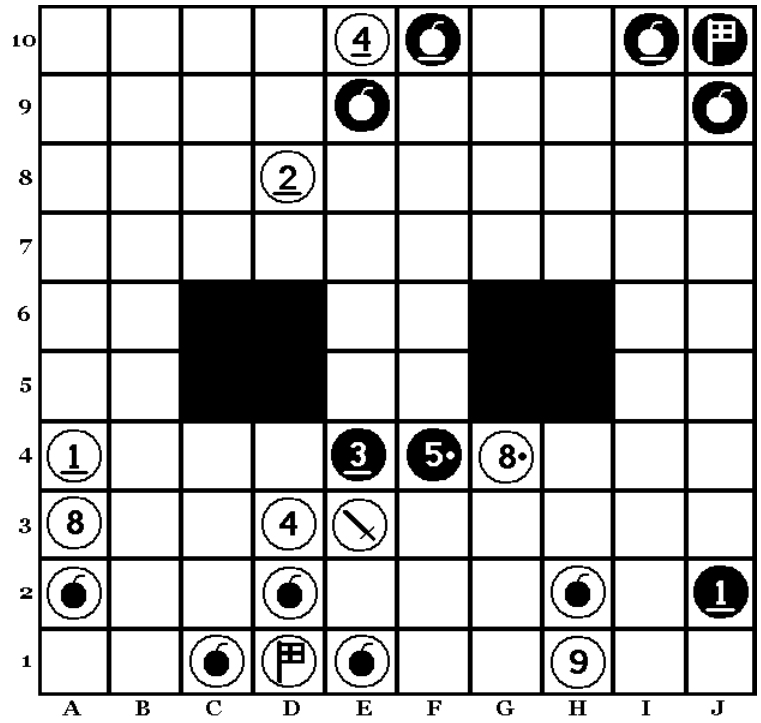
Drag	Perspecto	The General	Kommentarer
68	J1-I1	J8-J4	
69	J2oJ10	J4oD4	
70	F6-E6	F9-G9	
71	F1oF8	G9-H9	
72	E7-F7!	J3-J2	Perspectos taktiska sökning upptäcker ytterligare möjlighet till pjäsvinst.
73	F7xF8	D8-D7	
74	F8-F7	H9-I9	
75	F7-E7	J2-J1	
76	E7xD7	I9-I8	
77	B4-A4	B7xoB1	
78	C4-B4	I8-I7	
79	B4-B5	I7-I6	
80	B5-B6	I6-I5	
81	B6-B7	I5-I4	
82	B7xC7	I4-H4	
83	C7oC8	H4-G4	
84	A1-B1	G4oG3	
85	B1oB9	G10-H10	
86	D7-D8	C8-C7	
87	D8-C8	C7-D7	
88	E6-F6	D7-E7	
89	C8-B8	E7-F7	
90	F6-F5	F7-F6	
91	F5-F4	F6-F5	
92	F4-G4	F5-F4?	Här kunde *The General* fångat Perspectos major, men fångsten ligger 13 halvdrag framåt och bakom The Generals horisont.
93	G4-H4	F4-E4	
94	D4-C4	D9-D8	
95	B8xB9	H10-G10	
96	H4-I4	G10-G9	
97	I4-I5	G9-F9	
98	I5-I6	F9-F8	
99	I6-I7	F8-F7	
100	I7-H7	H8-I8	
101	H7-G7	F7-F6	
102	G7-G8	F6-F5	



Figur 22: Ställningen efter 102 drag

Drag	Perspecto	The General	Kommentarer
103	G8-G9	F5-F4	Perspectos major lockas först fram p.g.a en kombination av strategiska regler, men vid H7 ser den taktiska sökningen möjligheten att slå mot den misstänkta flaggan på F10.
104	G9-F9	F4-G4	
105	F9oF10	G4xG3	Flaggan visade sig vara en bomb.
106	C2-B2	G3xoG2	
107	B2-B3	C9-C7	
108	B9-B8?	C7-F7	Perspectos taktiska sökning räknar inte med långdrag, så *Perspecto* tror sig kunna göra en "gaffel".
109	B3-B4	F7-F2	
110	E2xF2	I8-I7	
111	B4-B5	I7-I6	
112	B5-B6	I6-I5	
113	B6-B7	I5-I4	
114	B7-A7	I4-I3	
115	A7xA8	I3-H3	
116	A8xA9	H3oH2	
117	A9xoA10	B10-A10	

Drag	Perspecto	The General	Kommentarer
118	B8-B9	E8-F8	
119	C4-B4	F8-G8	
120	B4-B5	G8-F8	
121	B5-B6	F8-F7	
122	B6-B7	F7-F6	
123	B7-B8	F6-F5	
124	B8-A8	F5-F4	
125	F2-G2	F4-F3	
126	A8-A9	F3-F2	
127	G2-G3	F2-G2	
128	G3-G4	G2xoG1	
129	A9xA10	D8-E8	
130	A10-B10	D10-D9	
131	B9-C9	D9-D8	Trots att majoren på B10 har möjlighet att slå ut flera misstänkta flaggor väljer *Perspecto* hellre att slå ut rörliga pjäser, eftersom dessa inte innebär någon risk.
132	C9-D9	E8-E7	
133	D9xD8	E7-F7	
134	B10xC10	F7-F6	
135	C10-D10	F6-F5	
136	D10xE10	F5-F4?	Här kunde *The General* ha blockerat den högra korridoren med sin fältmarkskalk och därmed hindrat minören från att komma igenom. Detta misstag hade *Perspecto* också gjort i samm situation och det visar hur svårt det är för en agent att spela bra slutspel.



Figur 23: Ställningen efter drag 136.

137	G4-H4	F4-G4	Nu finns bara en misstänkt flagga kvar, nämligen J10, vilket också är den riktiga flaggan!
138	H4-I4	G4-G3	
139	I4-I5	G3-G2	
140	I5-I6	G2-G1	
141	I6-I7	G1xH1	
142	I7-I8	H1-I1	
143	I8-I9	E4-D4	
144	I9xI10	D4-C4	
145	I10xJ10		The Generals flagga slås ut.

I detta parti visade sig den strategiska sökningen fungera bättre än i partiet mot Peter. Detta beror på att *The General* inte spelar lika målmedvetet i slutspelet och kraven på ett aktivt försvar blir lägre. Även om *Perspecto* förlorade många pjäser i början uppnåddes materiell överlägsenhet i mitten av partiet då *Perspectos* general slog ut många pjäser. Här visade sig *Perspectos* taktiska sökning fungera bättre än motståndarens.

10.3 Kvantitativa tester

Förutom att ta reda på vilken spelstyrka *Perspecto* har kan det vara intressant att veta vad som bidrar till denna spelstyrka. Därför har ett antal tester genomförts där olika ofullständiga varianter av *Perspecto* spelar emot varandra. Dessa tester är lätta att genomföra i stora mängder eftersom de kan göras helt automatiska. Sökalgoritm, strategisk sökning och draganalys testas var för sig.

10.3.1 Test av sökalgoritm

De tester som gjordes i avsnitt 8 visade att *p-e-minimax* var en bättre sökalgoritm än *expectiminimax* och *minimax*. Testerna utfördes dock helt utan strategisk sökning och draganalys, och sökdjupet varierades inte. Därför gjordes en större test med Perspecto, där bara sökalgoritm och sökdjup förändrades.

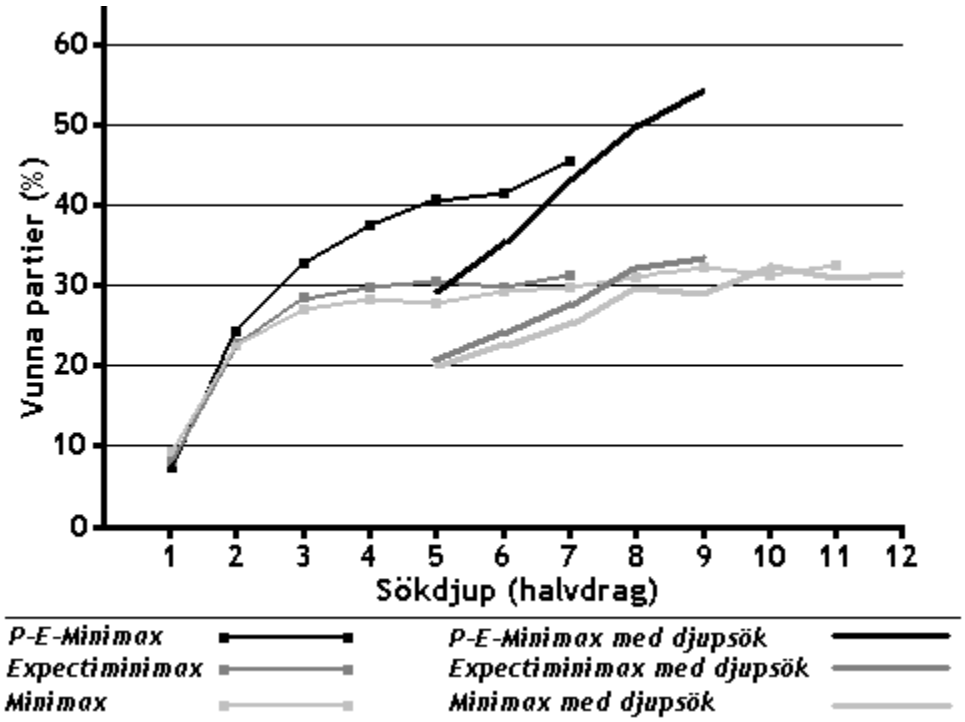
Samma tre sökalgritmer som i avsnitt 8 testades, med och utan variablet sökdjup (djupsök):

- *P-e-minimax* är den sökalgoritm som används i *Perspecto* och används här på precis samma sätt.
- *Expectiminimax* används på samma sätt som beskrivs i delavsnitt 8.2.
- *Minimax* används på samma sätt som i delavsnitt 8.1, fast här finns alpha-betaklippning implementerad vilket tillåter större sökdjup. Evalueringsfunktionen är också ändrad för att passa denna algoritm bättre: värdet av ett slag mellan två pjäser sätts till

  $$P \cdot v1 - (1 - P) \cdot v2$$

  där P är sannolikheten för att den egna pjäsen vinner, v1 är värdet av motståndarens pjäs och v2 är värdet av den egna pjäsen. Denna ändring gjordes för att kompensera för att *minimax*-trädet inte förgrenar sig vid slag.

Alla tester gjordes med en och samma motståndare: en snabbare variant av *Perspecto* med ett mindre i sökdjup, d.v.s 4-8 istället för 5-9. Detta gjordes för att testerna inte skulle ta alltför lång tid. Varje matchserie bestod av 200 partier, som utfördes med nollställd draganalys inför varje parti. Totalt utfördes 42 matchserier, vilket tog nästan 4 dygn på en 1.8 GHz processor.



Figur 24: Test av sökalgoritm. För algoritmerna med djupsökning används deras maximala sökdjup i diagrammet.

Figur 24 visar resultatet av testet. De viktigaste slutsatserna går att summera i fyra punkter:

1. P-e-minimax är den klart effektivaste algoritmen av de tre.

2. För att ett variabelt sökdjup skall bli bra måste 2-3 drag av vanlig sökning göras först, men sedan är den klart mer effektiv än plan sökning.

3. För minimax och expektiminimax planar vinstkurvan ut vid ca 1/3 vunna partier.

4. Även vinstkurvan för p-e-minimax visar tendenser till att plana ut, men ett par stegs extra sökdjup hade förbättrat *Perspecto* avsevärt.

10.3.2 Test av strategisk sökning

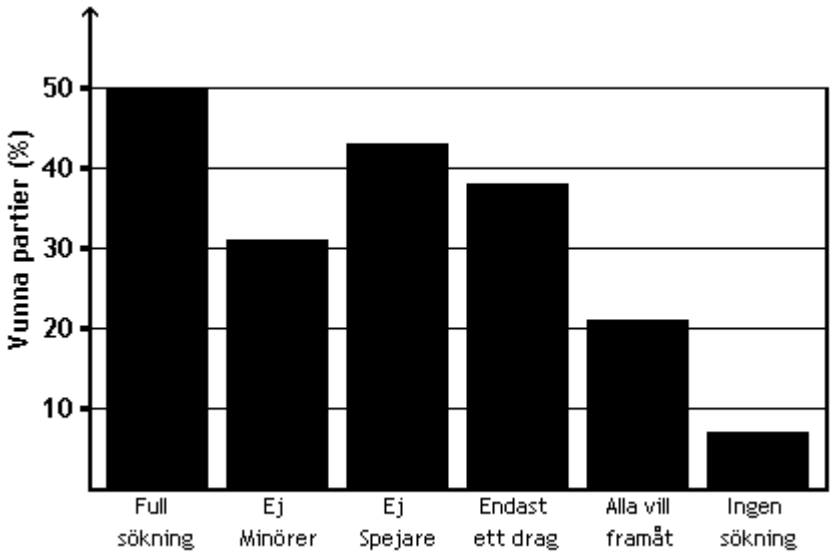
Totalt finns 21 st strategiska regler som styr hur pjäserna flyttar då minimax-sökningen inte hittar något tillräckligt bra drag. Dessa regler utformades så att de stämmer med hur en mänsklig spelare resonerar, och de gavs vikter utifrån hur pass stor betydelse de borde ha för utgången av spelet. Ett stort och omfattande test av varje regels faktiska betydelse

för Perspectos spelstyrka och dess funktion i grupp med andra regler ligger utanför syftet med detta arbete. Däremot kan det vara intressant att se hur mycket sämre *Perspecto* blir om man utelämnar den strategiska sökningen helt eller delvis.

Testerna genomfördes m.h.a en snabbare variant av *Perspecto* med ett mindre i sökdjup, d.v.s 4-8 istället för 5-9. Fem ofärdiga varianter av strategisk sökning testades:

- **Ej Minörer.** Här finns alla strategiska regler med utom de för minörer. För dessa används istället "alla vill framåt"-systemet.
- **Ej Spejare.** Här finns alla strategiska regler med utom de för spejare.
- **Endast ett drag.** I vanliga fall tittar den strategiska sökningen två egna drag framåt, här begränsas den till ett.
- **Alla vill framåt.** I testerna i avsnitt 8 användes ett poängsystem i evalueringsfunktionen som värderade pjäserna högre ju mer framskjutna de var. Här får detta system helt ersätta den strategiska sökningen.
- **Ingen sökning.** Här finns ingen strategisk sökning med överhuvudtaget.

Testerna genomfördes med matchserier om 200 partier vardera. Om inget avgörande nåtts efter 300 drag räknas partiet som oavgjort. I histogrammet nedan finns även den fulla sökningen med för jämförelsens skull. Det antas att den vinner 50% av partierna mot sig själv.



Figur 25: Test av strategisk sökning

Resultatet av testet var väldigt överraskande för mig. Det är helt klart så att den strategiska sökningen har klart större betydelse än jag tidigare trott. Testet av minimaxalgoritmerna visade att även en väldigt enkel algoritm kan uppnå 1/3 segrar mot *Perspecto*. I detta test räckte det med att plocka bort minörernas regler för att nå denna

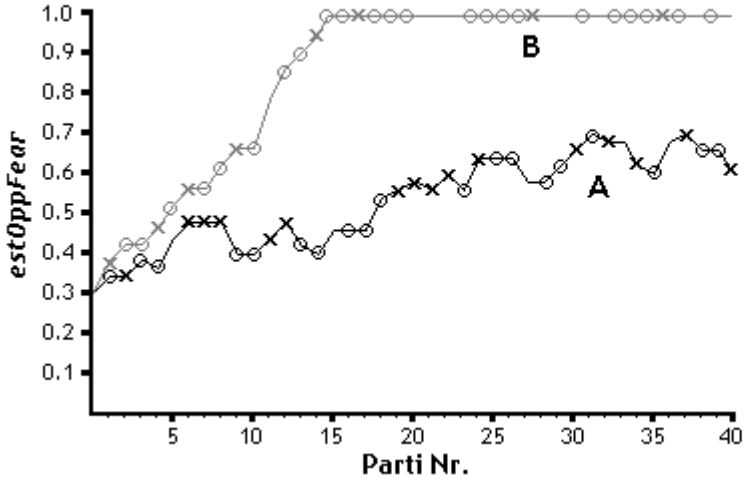
nivå. Anmärkingsvärt är också att den agent som inte använde sig av någon strategisk sökning alls inte lyckades vinna ett enda parti; de sju procenten kommer enbart från oavgjorda matcher (28 på 200).

10.3.3 Test av pjäsanalys och draganalys

I matchserien mot *The General* sparades inga psykologiska variabler mellan partierna, och detta har inte heller gjorts i några av de kvanitativa testerna i delavsnitt 10.3.1 och 10.3.2. I detta delavsnitt skall två tester utföras; det första skall undersöka om det är möjligt för *Perspecto* att lära sig en annan agents spelstil med draganalys och det andra skall avgöra hur stor betydelse pjäsanalysen har genom att testa olika varianter av den.

I det första testet spelar *Perspecto* med samma psykologiska varabler som i partierna mot Peter och The General, d.v.s att *OwnFear* är ett slumpvärde mellan 0.2 och 0.5, *EstOppFear* är 0.3 från början och *estOwnFear* hamnar mitt emellan de andra två variablerna. Perspecto spelar med 4-8 i sökdjup. Perspectos två motståndare spelar med samma sökdjup, men de psykologiska variablerna ser annorlunda ut.

- Motståndare A spelar med statiska psykologiska variabler. OwnFear = 0.35, estOppFear = 0.3 och estOwnFear = 0.33. Dessa ändras aldrig.
- Motstådare B spelar också med statiska variabler, och använder en extremt försiktig spelstil. Här är ownFear = 1.0, estOppFear = 0.3 och estOwnFear = 0.65.

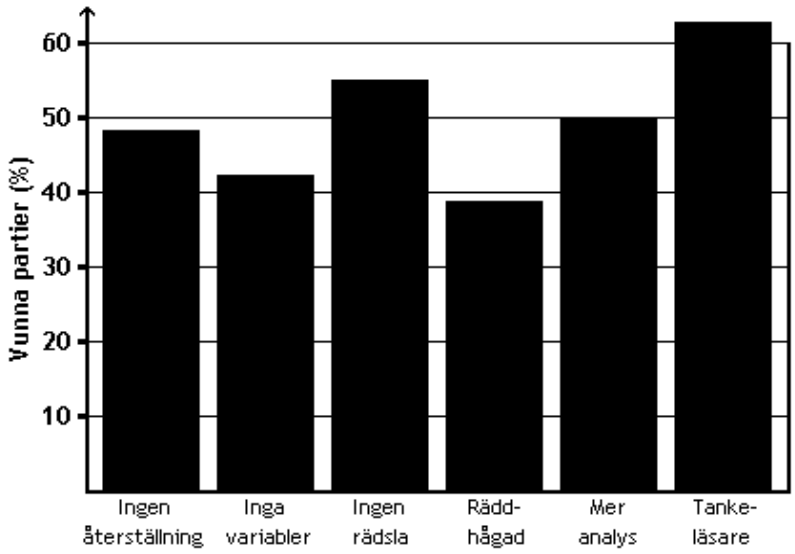


Figur 26: Test av draganalys. Inlärning av estOppFear mellan partier. "O" markerar en vunnet parti, "X" ett förlorat.

Perspecto lyckas här efter bara ett fåtal partier förstå skillnaden mellan motståndare A och B. Man kan också tydligt se hur Perspectos vinstfrekvens ökar efter att det korrekta värdet på estOppFear uppnåtts. Tyvärr ökar estOppFear även i matchserien mot motståndare A, trots att estOppFear passerar 0.35 redan under fjärde partiet och fortsätter att öka under hela matchserien. En annan nackdel är att ändringen per parti är oväntat liten. Helt klart är det så att reglerna för hur estOppFear skall ändras borde vara mer sofistikerade.

Det andra testet är ett rent spelstyrke-test, där *Perspecto* spelar med 4-8 i sökdjup och med standardvärden på de psykologiska variablerna, d.v.s att *OwnFear* är ett slumpvärde mellan 0.2 och 0.5, *estOppFear* är 0.3 från början och *estOwnFear* hamnar mitt emellan de andra två variablerna. Testerna består av matchserier med 200 partier i varje. De psykologiska variablerna återställs mellan varje parti. De olika varianter som skall testas mot Perspecto är:

- **Ingen återställning.** Här återställs inte estOppFear mellan varje parti. Annars är de psykologiska variablerna samma som för *Perspecto*.
- **Inga variabler.** Här är alla de tre psykologiska variablerna 0.
- **Ingen rädsla.** Här är ownFear 0, de andra variablerna är samma som för *Perspecto*.
- **Räddhågad.** Här är ownFear 0.8, de andra variablerna är samma som för *Perspecto*.
- **Mer analys.** Här förändras estOppFear tre gånger mer än normalt, d.v.s med 0.12 och 0.06 istället för 0.04 och 0.02.
- **Tankeläsare.** Denna agent "fuskar" genom att den alltid vet värdet på Perspectos variabler. EstOppFear = perspectos ownFear och estOwnFear = perspectos estOppFear. OwnFear är ett slumpvärde mellan 0.2 och 0.5.



Figur 27: Test av pjäsanalys

I detta test bekräftas Perspectos oförmåga att få fördel genom att analysera en motståndare över flera partier. Inte heller metoden att öka förändringen av variablerna hade någon effekt.

Att "Ingen rädsla" lyckades bra i förhållande till "räddhågad" var överraskande. Det verkar som om ett alltför högt värde på ownFear minskar spelstyrkan kraftigt. Att "ingen rädsla" lyckas nå över 50% mot *Perspecto* är dock inte underligt. Det beror på att Perspectos värde på estOppFear sämre motsvarar motståndarens ownFear i jämförelse med vice versa.

"Tankeläsaren" lyckades vinna över 60% av matcherna, men jag hade väntat mig att den skulle vinna mer. Detta visar på att de psykologiska variablerna har mindre betydelse för spelstyrkan än väntat.

11. Slutsatser

Resultatet av detta examensarbete blev ett helt annat än det jag väntade mig när jag började. Då tänkte jag mig att det bara var snäppet svårare att programmera en AI till *Stratego* än till *Othello*. Redan från början kändes det dock som att varje gång jag kom närmare en lösning på problemet förflyttade sig lösningen själv längre bort. De största svårigheterna har varit rent programeringstekniska, och jag har flera gånger gjort om programmets struktur för att det hela skall bli lättare att överblicka och för att underlätta felsökning. Så här i efterhand vet jag att det varit klokare att först använda en "minivariant" av *Stratego* på kanske 4x4 rutor, och inte gå över till stort bräde förrän allting fungerade på det lilla.

Detta avsnitt delas upp i två delavsnitt där det första utgör en utvärdering av resultatet och försöker ge svar på de frågor som ställdes i problembeskrivningen. Det andra delavsnittet beskriver möjliga förbättringar av *Perspecto.*

11.1 Utvärdering

Resultatet av testerna som redovisades i avsnitt 10 visar att *Perspecto* kan spela jämnt mot medelgoda mänskliga spelare och förmodligen är en bättre strategospelare än något annat datorprogram i världen. Det är dock långt kvar innan det kan mäta sig med en verkligt skicklig mänsklig spelare. De kvantitativa testerna visar också att den strategiska sökningen har mycket större betydelse än jag tidigare trodde, och det är också här *Perspecto* uppvisar de största bristerna i sitt spel. Mot skickliga mänskliga spelare har naturligtvis också pjäsanalysen större betydelse än mot andra datorprogram, och här blir den analys som används i *Perspecto* alltför primitiv.

I problembeskrivningen ställdes tre frågor om *minimax*-algoritmen och *Stratego*. Dessa skall jag här besvara med hjälp av de erfarenheter jag fått under examensarbetet.

- **Kan *minimax*-algoritmen användas för att spela *Stratego* på samma nivå som en skicklig mänsklig motståndare?**

De misstag som *Perspecto* gjorde i matchen mot Peter visar att mycket återstår att göra innan ett datorprogram kan tävla med bra mänskliga spelare. Trots att *Perspectos* visade sig vara en taktiskt bra spelare, var dess strategiska spel svagt, speciellt i slutspelet. Det psykologiska spelet fungerade i den enda match som spelades, men hade antagligen blivit försutsägbart i en matchserie.

Trots det negativa resultatet av matchen tror jag att det är möjligt att med en bättre pjäsanalys och bättre strategisk sökning skapa en agent som vinner mot en stor majoritet av de mänskliga spelarna. Detta exjobb har till 90 % handlat om att utveckla en

minimaxalgoritm, trots att denna algoritm bara utgör ena halvan av det *Perspecto* behöver för att vinna.

- **Vilken typ av *minimax*-algoritm lämpar sig bäst till att användas för *Stratego*?**

Av de tre algoritmer som testats har *p-e-minimax* visat sig överlägsen. Detta visas också av partierna mot *The General* som ju också använde sig av *minimax.* Resonemanget i avsnitt 8.3 visar att *p-e-minimax* gör en del felaktiga antaganden. Dessa har dock inte visat sig leda till några större misstag i sökningen.

Lokal sökning är ett utmärkt sätt att kringå den stora förgreningsfaktorn i *Stratego.* Den typ som användes i *Perspecto* skär ner förgreningsfaktorn från ca 30 ner till ca 7, vilket kompenserar för att ingen alfa-betaklippning finns utvecklad till *p-e-minimax.*

- **Är *minimax*-algoritmen det bästa sättet för ett program att spela *Stratego*, eller finns det något annat sätt som är bättre?**

De två strategoagenter som jag studerat närmare och som inte använder någon typ av *minimax*-algoritm är *Stratesys* av Caspar Treijtel och *Reveal-your-rank!* av Raimonds Rudmanis. Dessa använder expertsystem. Ingen av dessa program uppvisar speciellt hög spelstyrka. Trots att *Stratego* innehåller okänd information skiljer mycket av taktiken inte från ett spel som schack, där pjäser måste samverka för att fånga motståndarens pjäser i fällor. Sådana taktiska problem är svåra att lösa utan sökning.

11.2 Framtida förbättringar

Alla som någon gång har konstruerat en brädspelande agent vet att det är ett problem som inte har något slut. Ju bättre ens agent blir på att spela, ju bättre blir man själv på att hitta förbättringar av den. Någon gång måste man emellertid sätta stopp, och skjuta fram ytterligare förbättringar till någon gång i framtiden. I detta avsnitt presenteras några förslag till sådana förbättringar.

- **P-e-alfabeta.** Att utföra alfabetaklippning i *p-e-minimax* skulle vara betydligt svårare än i expectiminimax, speciellt efter de ändringar som gjordes av min-noderna i avsnitt 9.4. Den förbättring av sökdjupet som skulle vara resultatet av en sådan klippning skulle heller inte förbättra agentens spelstyrka på något avgörande sätt. Om man kom på ett bra sätt att utföra klippningen på, skulle det dock vara ett enkelt sätt att förbättra sökningen.

- **Min-noderna blir slumpnoder.** Egentligen är alla motståndarens drag i spel med okänd information resultatet av denna information. Därför kan man aldrig med säkerhet veta vilket motståndarens bästa drag i en situation är. En utveckling av *p-e-minimax* borde därför sätta en slumpnod i varje min-nod. Det svåra blir att med hjälp

av värdena i nodens barn-noder få fram sannolikheter för hur troliga de olika dragen är.

- **Spelteori i pjäsanalysen.** I enkla spel med okänd information kan man ofta spelteoretisk bestämma en optimal strategi. Dessa strategier innehåller alltid slumpmässiga beteenden för att göra spelarens val svåra att förutsäga. Att tillämpa spelteori för att räkna ut optimala bluffrekvenser i *Stratego* skulle göra en agent väldigt mycket starkare, speciellt mot en mänsklig motståndare.

- **Strategisk analys innan varje drag.** Innan varje drag kan en analys av spelplanen göras. Denna analys skall avgöra hur pass skyddad den egna flaggan är, huruvida den egna sidan har materiellt övertag eller ej, o.s.v. Analysen används sedan för att förändra de strategiska reglerna och få vissa pjästyper att skydda flaggan eller attackera oftare. En bra utförd sådan analys skulle göra en agent väldigt mycket starkare.

Referenslista

[1] Russel, Samuel och Norvig, Peter. *Artificial Intelligence; A modern approach*. Prentice hall, Upper Saddle River, New Jersey, 1995

[2] Murphy ,Robin R. Introduction to AI Robotics. A Bradford Book, 2000

[3] http://www.edcollins.com/stratego/isf-rules.htm (verifierad 26 april 2005)

[4] Wilson, Robert. *Computing equilibria of two-person games from the extensive form.* Management Science, 18(7):448-460, 1972

[5] Blair, Jean R S, Mutchler David, van Lent Michael. *Perfect Recall and Pruning in Games with Imperfect Information*. Computational Intelligence, 1996.

[6] Treijtel, Caspar. *Multi-agent Stratego*. Delft University of Technology, 2000

Appendix 1: Strategisk sökning

Perspectos strategiska sökning, som beskrivs i delavsnitt 9.2, baserar sig på ett antal regler som styr hur de olika pjäserna dras till andra pjäser. I detta appendix finns en fullständig lista över de olika reglerna.

Spionen
[S, f1,40] Spionen dras till fientlig fältmarskalk med kraft 40.

Spejaren

P.g.a spejarens förmåga att flytta långt behöver den inga strategiska regler.

Minören

[8, fB, 16] Minören dras till fientlig bomb med kraft 16.
*[8, fX, pB*16]* Minören dras till fientlig okänd pjäs med kraft 16 * sannolikheten för att pjäsen är en bomb.
[8, fO,-2] Minören dras till fientlig okänd pjäs med kraft -2
*[8, fX,80 * pF]* Minören dras till fientlig okänd pjäs med kraft 80 * sannolikheten för att pjäsen är en flagga

Sergeanten

[7,fO,1] Sergeanten dras till fientliga okända med kraft 1

Löjtnanten

[6,fO,1] Löjtnanten dras till fientliga okända med kraft 1

Kaptenen

[5,fO,0.5] Kaptenen dras till fientliga okända med kraft 0.5
[5,f<5,-6] Kaptenen dras till fientlig högre pjäs med kraft -6

Majoren

[4,f>4,8] Majoren dras till fientlig lägre pjäs med kraft 8
[4,f<4,-10] Majoren dras till fientlig högre pjäs med kraft -10

Översten

[3,f>4,8] Översten dras till fientlig lägre pjäs med kraft 8
[3,f=4,16] Översten dras till fienlig major med kraft 16

[3,f<3,-16] Översten dras till fientlig högre pjäs med kraft -16

Generalen

[2,f>4,4] Generalen dras till fientlig lägre pjäs med kraft 4
[2,f=4 | f=3,16] Generalen dras till fientlig major eller överste med kraft 16
[2,f=1,-20] Generalen dras till fientlig fältmarskalk med kraft -20

Fältmarkskalken

[1,f>4,4] Fältmarkskalken dras till fientlig lägre pjäs med kraft 4
[1,f=4 | f=3,16] Fältmarkskalken dras till fientlig major eller överste med kraft 16
[1, f=2, 32] Fältmarkskalken dras till fientlig general med kraft 32

Allmänna strategiska regler

*[<8, fX(op), 80 * pF]* Alla pjäser med högre rang än minör dras till fientlig okänd pjäs med kraft 80 * sannolikheten för att pjäsen är en flagga. Den okända pjäsen måste stå öppen (ej omgiven av fientliga orörliga pjäser).

Varje egen pjäs som flyttat får alla strategiska krafter multiplicerade med en faktor som beror på vilken rang de har. För minörer och spioner är faktorn 2. För sergeanter, löjtnanter, kaptener och majorer är den 1. Pjäser med högre rang än major har faktorn 2.