Opponent Modeling in Texas Hold'em

Nicholas Piël

June 30, 2006

Begeleider: Maarten van Someren

Nicholas Piël <npiel@science.uva.nl>
Fannius Scholtenstraat 71 II
1051 EV Amsterdam

Abstract

Er altijd al interrese geweest naar onderzoek binnen het poker spel maar dit spitste zich meestal toe op gesimplificeerde versies van het spel. Dit heeft als groot nadeel dat de charme en de meest interresante eigenschappen van het spel verdwijnen. Wanneer je in het poker spel goed wilt kunnen presteren moet je niet alleen je eigen kansen goed in kunnen schatten maar ook die van anderen en de eventuele acties die andere spelers zullen doen. In mijn afstudeer project richt ik me dan ook op het volledig poker spel en probeer een systeem te maken dat andere spelers kan modeleren en hun kaarten probeert te voorspellen.

Contents

Chapter 1

Introduction

Poker is in de Verenigde Staten een zeer populaire spel[1]. Er zijn daar niet alleen veel actieve spelers die er hun geld mee verdienen maar ook televisie kanalen die 24 uur per wijden aan het poker spel. Ook in Nederland begint Poker een sterke opmars, hoewel het volgens de huidige Nederlandse kansspel wetgeving alleen in het Holland Casino mag worden gespeeld zijn er al veel Nederlanders online aan het spelen en beginnen ook verscheidene tv programma's aandacht te geven aan het poker spel.

Daarnaast is poker een interresant domein voor de Kunstmatige Intelligentie. Als spel bevat poker namelijk enkele interresante eigenschappen zoals bedrog, risico inschatten, tegenstanders modeleren en kansen berekenen. Het is dan ook een uitstekende omgeving om verschillende leer methodieken te testen.

Door de grote complexiteit die de mogelijkheden en de onzekerheid in het poker spel met zich meebrengen is het tot dusver niet zonder meer mogelijk om een optimale strategie te berekenen. Het puur doorekenen van het spel zoals bijvoorbeeld nu bij 'schaken' gebeurt (tot een bepaalde diepte) is bij poker op dit moment niet mogelijk. Over het algemeen zijn er twee anpakken in het wetenschappelijk onderzoek naar het pokerspel. Een waarbij het pokerspel ondergeschikt wordt gemaakt aan het onderzoeksmodel en een ander waar juist de regels en complexiteit intact worden gehouden en het model ondergeschikt is aan de werkelijkheid. De eerste aanpak is de oudste en dateert al uit 1944 waar Von Neumann en Morgenstern[4] al experimenteerden[2]met een versimpelde poker versie. Maar ook recentelijk krijgt deze aanpak nog veel aandacht bijvoorbeeld bij het onderzoek naar POMDP's[7]. De tweede aanpak begint de laatste tijd meer op te komen en wordt voornamelijk gedreven door interesse in Machine Learning, het is meer toegespitst op een practische implementatie dan op een

[1] Of moeten we zeggen sport? Aangezien het proffesioneel wordt uitgevoerd door enkele top spelers.

[2] Ik vind het overigens interresant om te zien dat de grondleggers van de huidige computer technology hun wortels hebben in de AI. Dit soort bevindingen geven me steeds meer en meer de neiging om te zeggen dat AI niet een klein broertje is van de Informatica maar dat het juist eerder andersom is. AI is de werkelijke wetenschappelijke richting terwijl Informatica meer het instrument is.

wiskundig theoretische onderbouwing zoals bij de eerste aanpak.

Het meeste succes op dit gebied en met deze laatste aanpak heeft de *University of Alberta Computer Poker Research Group* onder leiding van Jonathan Schaeffer (auteur van Chinook, de wereld kampioen *Man-Machine Checkers*[6]). Hun 'Pokibot' staat bekent als een van de beste pokerbots[3]. De aanpak die ik hanteer lijkt sterk op hoe het modeleren van tegenstanders wordt gedaan bij Loki[3] (voorloper van Poki) maar waar bij Loki ervoor wordt gekozen om de potientiele hand te benaderen en niet te berekenen ivb met computationele kosten. Heb ik een manier gevonden om deze berekening vlot en correct uit te voeren. In Loki wordt het biedsysteem deels bepaald door regels geabstraheerd uit 'Expert Kennis', het voordeel van dergelijke aanpak is dat je hierdoor snel een redelijke goed presterend systeem hebt. Het nadeel van dergelijke vaste regels is dat ze statische zijn, lastig te onderhouden en vatbaar voor exploitatie van de tegenstander. Ik wil dan ook geen gebruik maken van zogenaamde expert kennis eventueel ten koste van de prestatie op korte termijn.

Maar voordat ik het ga hebben over mijn aanpak in hoofdstuk 3 zal ik eerst wat dieper ingaan op het poker spel en wat de vereisten zijn voor een goede poker speler in hoofdstuk 2. In hoofdstuk 4 bespreek ik dan de implementatie en in hoofdstuk 5 sluit ik af met de resultaten.

[3]Afkorting van pokerrobot, ofwel een kunstmatige intelligente agent die in staat is om te kunnen pokeren.

Chapter 2

Poker

2.1 Texas Hold'em

Poker is kaart spel waarbinnen veel varianten bestaan. Texas Hold'em is de meest gespeelde variant en wordt over het algemeen beschouwd als de strategische meest complexe variant waarbij de verhouding tussen geluk en kunnen het kleinst is[1]. Het kan worden gespeeld met 2 tot 10 spelers waarbij het de bedoeling is om de beste HAND[1] te maken. De kaarten zijn gerangschikt van 2-9 en dan J,Q,K,A. Een A kan eventueel ook tellen als 1 om zo een STRAIGTH te kunnen maken. Een volledige hand bestaat uit totaal 5 kaarten waar eventueel een combinatie mee gemaakt kan worden. Deze combinaties zijn in grote lijn net zo gerangschikt als bij Yathzee maar zie voor de duidelijkheid tabel2.1. In de eerste ronde van het spel krijgt iedere speler 2 gesloten kaarten. De twee spelers (BLINDS) die na de dealer zitten (links van de dealer aangezien het spel met de klok mee gaat) moeten voordat ze hun kaarten zien bepalen of ze mee willen doen met het spel door de BLIND te betalen, dit is om zo hun voordeel wat ze hebben doordat ze als laatste aan zet zijn te minimaliseren. De derde speler links vand de dealer mag dan beginnen met inzetten op zijn kaarten, waarna de andere spelers volgen. In de biedingsronde heb je drie mogelijkheden: BET (het verhogen van de inzet) / CALL (het meegaan met iemand anders zijn inzet) / FOLD (het niet meegaan en dus opgeven voor deze ronde.

Wanneer iedereen ofwel gecalled of gefold heeft worden er drie kaarten OPEN op tafel gelegd de zogenaamde FLOP. Deze drie kaarten behoren tot iedereen en het is nu al mogelijk voor elke speler om een HAND te maken. Na de FLOP volgt weer een bied ronde en wordt er nog een kaart open op tafel gelegd, de TURN. Ook nu volgt weer een bied ronde, en kan iedere speler uit de 6 kaarten een HAND van 5 kaarten maken. Na de TURN komt de RIVER, de vijfde en laatste openkaart wat gevolgd wordt door de laatste bied ronde, SHOWDOWN genaamd. In de SHOWDOWN wint degene met de beste kaarten. Het is natuurlijk mogelijk

[1]Een pokerterm wat simpelweg betekent 5 kaarten. In dit verband betekent het de beste 5 kaarten uit de in totaal 7. Kijk voor meer pokertermen achter in de Glossary

Table 2.1: Overzicht van kaart combinaties van goed naar slecht

Naam	Kaart Combinatie
StraightFlush	5 opeenvolgende kaarten in dezelfde kleur (bijvb: Ah, Kh, Qh, Jh, Th)
FourOfaKind	4 dezelfde kaarten (bijvb: Ah, Ad,As,Ac)
FullHouse	een combinatie van 3 en 2x dezelde (bijbv: Ah, As, Kh, Kd, Kc)
Flush	5 kaarten in dezelfde keur (bijvb: 2h, 5h, Jh, Qh, Ah)
Straight	5 opeenvolgende kaarten (bijvb: Ac, Kh, Qd, Js, Th)
ThreeOfaKind	3 dezelfde kaarten (bijvb: 2h 2s 2d Ah Kd)
TwoPair	2x 2 dezelfde kaarten (bijvb: 2h 2c 4d 4s Ah)
Pair	2 dezelfde kaarten (bijvb: 2h 2d 5h 8s 5c)

dat alle andere spelers al hebben opgegeven waardoor de showdown nooit bereikt wordt.

Qua grote van het inzetten bestaan binnen Texas Hold em meerdere varianten zo heb je de No-Limit waarbij geen limiet wordt gesteld aan wat je mag inzetten en de Limit variant. In mijn project heb ik me gericht op de Limit variant. In de limit variant hoef je minder rekening te houden met het instellen van de grote je inzet en het beoordelen van deze inzet van andere spelers. Over het algemeen wordt de No-Limit variant als lastiger en moeilijker beschouwd dan Limit Texas Hold'em.

2.2 Goed Spel

Nu we de regels hebben besproken wordt het tijd om te gaan kijken naar wat de ingredienten zijn van een poker pro. Schaeffer indentificeert[5] de volgende zes eigenschappen van een 'world-class poker speler':

Hand-Strength Hoe sterk is je huidige hand. De simpelste implementatie kijkt alleen naar de huidige kaarten en de publieke kaarten op tafel. Een meer geavanceerdere versie kijkt ook naar de board positie en de bied geschiedenis van de andere speler.

Hand-Potential Hoe sterk kan je hand worden wanneer er meer kaarten bijkomen. Ook hier kijkt de meest simpele implementatie alleen maar naar de kaarten op tafel en de eigen kaarten en kun je dit net als bij Hand-Strength uitbreiden in complexiteit.

Betting-Strategy Wanneer moet ik inzetten, wanneer moet ik opgeven. De minimale strategie kijkt alleen naar de sterkte van de hand. Een betere strategie kijkt ook naar PotOdds, bluffen, en het model van de tegenstander.

Bluffen Bluffen geeft je de mogelijkheid om slechte handen toch nog nuttig te gebruiken. Een minimaal bluff systeem bluft met een bepaalde

willekeurigheid. In de praktijk wordt er vooral gebluft wanneer je de mogelijkheid hebt een hele goede hand te krijgen (en net mis te lopen).

Onvoorspelbaarheid Maak het tegenstanders moeilijk om een correct model van jou te vormen door een dynamische speelstijl te hanteren. Hierdoor zullen tegenstanders een verkeerd model van je verkrijgen wat dan geexploiteerd kan worden.

Opponent-Modeling Geeft een inzicht in wat de PocketCards zijn van een tegenstander. Opponent Modeling is essentieel voor goed spel en is datgene wat het onderscheidt maakt tussen een speler die gebruik maakt van een "optimal strategy" en een "maximizing stratey". De optimalespeler maakt zijn beslissingen aan de hand van spel-theoretische kansen de maximalizerende-speler houdt rekening met de sub-optimale eigenschappen van de tegenstander en zal zijn eigen spel aanpassen om dan deze eigenschappen te exploiteren.

Van de bovenstaande vereisten zit alleen de 'onvoorspelbaarheid' niet verwerkt in mijn huidige speler. De overige eigenschappen zitten voornamelijk impliciet verwerkt in mijn pokerbot.

Chapter 3

Aanpak

3.1 Theoretische Aanpak

3.1.1 Overzicht

Ik heb voor mijn aanpak wat betreft het modeleren van tegenstanders gekozen voor een model gelijk aan *Loki,* ofwel tegenstanders worden gedeeltelijk gemodeleerd door gewichten die toegewezen worden aan hun mogelijke PocketHands. Ik heb hiervoor gekozen omdat een dergelijke aanpak transparant is, (in tegenstelling tot bijvoorbeeld een ANN) de keuzes die het systeem maakt zijn duidelijk en gedragen zich niet als een blackbox. Daarnaast omdat een gewichten systeem makkelijk is aan te passen zo kunnen we meerdere gewichten matrixen achter elkaar plakken. Ook is een dergelijk systeem snel. Niet alleen het leren maar ook het uitvoeren.

In mijn aanpak kijk ik naar de distributie van acties (bet/call/fold) van de verschillende spelers en pas aan de hand van deze gegevens het model van de speler aan. Met dit spelers model kijkt de speler dan naar zijn eigen potentiele hand en die van de overige spelers en bepaalt aan de hand daarvan wat zijn eigen actie zal zijn.

Daarnaast maakt mijn speler (die ik van nu af aan *Yoki* zal noemen) in tegenstelling tot *Loki* wel gebruik van zogenaamde ExposedPocketHands. Dit zijn handen die aan het eind van een ronde zichtbaar worden doordat meerdere spelers tot het eind van de showdown overblijven. Wat *Yoki* doet, is dan de ronde terug bekijken en de score van de verschillende spelers berekenen waarop ze hebben gecalled of geraised. Deze informatie gebruikt het dat in de vervolg rondes om de intentie van een speler beter te kunnen begrijpen.

Voordat ik dieper inga op het modeleren van tegenstanders zal ik eerst uitleggen hoe mijn huidige simulatie in elkaar steekt en hoe ik verschillende speel stijlen kan genereren waarover *Yoki* dan kan modeleren.

3.1.2 Simulatie

Het modeleren van tegenstanders kan op verschillende manieren. Zo zou je bijvoorbeeld kunnen gaan kijken naar statistische data van door mensen gespeelde partijen. Online is veel data te vinden waarop je direct zou kunnen beginnen met dataminen. Het grote voordeel van een dergelijke aanpak is dat je metteen kan beginnen en dat je geleerde modellen zeer relevant zijn aangezien ze gebasseerd zijn op het echte spel. Het grote nadeel hiervan is dat wanneer je dit wil omzetten naar acties die door je pokerbot gebruikt kunnen worden je een groot probleem blijft houden. Ik heb daarom gekozen voor 'self-play' simulaties, een simulatie waarbij de bot speelt tegen andere bots en dan daar van moet proberen te winnen. Het grote voordeel is dat je metteen iets krijgt wat werkt. Het nadeel is dat er een grote bias zit in de acties van andere spelers. Wanneer je bot goed speelt in een simulatie betekent dit nog niet dat het in de 'echte wereld' goed zal presteren. Een ander nadeel is dat het implementeren van een volledige poker simulatie (we willen immers niks afdoen aan de complexiteit van het spel) veel tijd kost.

We willen immers dat de simulatie correct werkt, genoeg informatie verzamelt waarop we kunnen modeleren en snel genoeg is om informatie binnen een niet al te lange tijd te kunnen verzamelen.

3.1.3 Speler Modellen

Behalve de volledige simulatie van het spel zelf, willen we ook spelers hebben die het spel spelen. Ik heb gekozen voor relatief eenvoudige spelers die naar hun potentiele hand kijken en aan de hand daarvan een actie kiezen. Door te spelen met parameters kun je al snel verschillende spelers modeleren. In de rest van dit verslag kan je de volgende 3 speel modellen tegen komen:

TIGHT Een speler die voorzichtig speelt. Snel FOLD en alleen BET wanneer hij zijn kansen hoog inschat.

LOOSE Een speler die los speelt wanneer er al een kleine kans is dat hij kan winnen; BET hij. En opgeven doet hij alleen bij bar slechte kaarten.

DEFAULT Een speler die tussen bovenstaande spelers inzit.

3.2 Hand Evaluatie

De kern van elke poker speler is het evalueren van zijn eigen POCKETCARDS in combinatie met die op het BOARD. Op het eerste gezicht lijkt dit misschien een futiliteit maar ik durf te stellen dat de kwaliteit van de simulatie valt met hoe goed je je eigen hand kan inschatten. Ook voor de hand evaluatie geld dat deze correct en snel moet zijn. Hij moet correct zijn dat hij niet alleen altijd de juiste waarden teruggeeft maar ook dat hij dit doet zonder een bepaalde bias. Hij moet snel zijn omdat dit het hart van het model is. Van alle functies die worden aangeroepen wordt de hand-evaluatie het meest aangeroepen.

Het gaat hierbij niet alleen om het vergelijken van twee handen met elkaar maar ook om bijvoorbeeld te kijken naar situaties zoals deze:

- PocketCards: [5h, 4h],
- Flop: [2h, Ah, As]

In deze situatie ziet er het op het eerste moment niet al te gunstig uit totdat je iets beter kijkt, dan zie je dat een willekeurige 3 je een straat geeft en dat wanneer er een harten bijkomt je een flush hebt. Volgens de regels die in veel poker boeken beschreven staan zou je outs moeten tellen[2]. In totaal heb je 4 drieën die je een straat geven en 13-4-1 = 8 harten die je een flush geven. In totaal heb je dus 8 kaarten die je waarschijnlijk de beste Hand geven. De kans dat de volgende kaart 1 van de 12 wordt is dan: $\frac{52-5}{12} = 1/3$ Wanneer je ook nog eens gezien hebt dat in de PreFlop bied ronde niemand heeft ingezet zou dit kunnen betekenen dat sowieso niemand een A heeft en je daar dus niet bang voer hoeft te zijn.

Stel nu dat iemand in de vorige ronde wel zwaar heeft ingezet ingezet en dat de pot nu 100 groot is. Daarnaast staat in deze ronde er een bed tegen je uit van 20. Wat doe je, call je die 20? Volgens de regels van de PotOdds zou je zeker moeten callen. Immers een call van 20 tegen over 100 betekent dat alles wat een grotere kans heeft dan 1/5 om te winnen zich uiteindelijk zou terugverdienen.

Hoewel bovenstaande een aanpak is die zeer gebasseerd is op regels en ik al eerder heb verklaard dat ik geen regels toe pas zit een dergelijke iets niet expliciet in mijn hand evaluatie functie. Maar het zit er wel impliciet in (los van de PotOdds).

Wanneer ik deze waardes in mijn hand evaluator gooi dan zie dat de verwachtingswaarde voor deze flop gemiddeld 2714 en dat de verwachtingswaarde voor de speler 2196 is. Deze score is evenredig met de rangorde van de handen en lagere scores betekent dus betere kaarten. Dit wordt berekend door alle mogelijke extra kaarten bij langs te gaan en dan het gemiddelde te nemen van deze scores. Ik zal de preciese werking van dit verder uitleggen in het hoofdstuk over de implementatie.

Chapter 4

Implementatie

4.1 Algemene informatie

In dit hoofdstuk beschrijf ik hoe ik het idee heb geimplementeerd en waar ik tegen aan ben gelopen tijdens de implementatie. Over het algemeen heb ik het de simulatie en het modeleer systeem geimplementeerd in Python. De eerste HandeEvaluator heb ik geimplementeerd in Prolog, de 2e in puur Python, de 3e in C en de 4e in een combinatie van Python en C. De reden van deze verandering is voornamelijk de snelheid van het uitvoeren van de HandEvaluatie functie. Zoals al eerder vermeld is het van groot belang dat dit snel gebeurt aangezien de hoeveelheid calls naar deze functie groot is. Immers wanneer we de FLOP weten en we willen weten wat onze eigen potentiele hand kan zijn, dan hebben we in totaal 47*46 mogelijke 7-kaart handen. Binnen elke hand van 7 kaarten heb je $\begin{pmatrix} 7 \\ 5 \end{pmatrix} = 14$ mogelijke combinaties. In totaal moet je dus 30268 handen evalueren wanneer je je eigen potentiele hand sterkte wil weten.

Het probleem wordt nog groter wanneer je niet de pocket cards weet zoals dat het geval is wanneer je de score van je tegenstander wil berekenen. Je moet dan voor elke mogelijke pocket card $\begin{pmatrix} 49 \\ 2 \end{pmatrix} = 1176$ de score berekenen. Wat betekent dat je in totaal 35.595.168x je hand evaluatie functie moet aanroepen.

In mijn allereerste prolog implementatie duurde het evalueren van 1 hand zo'n 0.1 seconde. Dit zou betekenen dat het uitrekenen van de score van een potentiele hand 41 dagen zou duren. De prestaties van de pure python code heb ik helaas niet gemeten. De C-code daarentegen is zeer efficient. Ik implementeer *Cactus Kev's Poker Hand Evaluator[9]* die door slim gebruik van hash tabellen extreem snel is. Daarover heen pas ik een patch[8] toe die de prestatie nogmaals zo'n 2.7x verbeterd door de binary search in het algoritme te vervangen door een 'PerfectHash' het resultaat is een bizar snel algoritme dat een 7-kaarts hand kan evalueren in 0.0000893 seconden. Het totaal doorrekenen kost dan nog 'maar' 3180 seconden ofwel 53 minuten.

Dit is uiteraard nog steeds veelte langzaam om gebruikt te kunnen worden

in een simulatie en is dan ook de reden dat ze bij Loki deze waarde benaderen en niet bereken. Op dit moment kan ik de potentiele waarde van een flop berekenen in 0.05 seconden[1]. Hoe ik dit voor elkaar heb gekregen zal ik verder uitleggen in het hoofdstuk van de Implementatie.

4.2 Implementatie van de simulatie

De implementatie van de simulatie is gedaan in Python. Het is een volledige simulatie van Texas Hold'ehm inclusief BLINDS, SITIN, SITOUT, SPLITPOTS, instelbare BED sizes en maxima. Het idee hier achter is dat door een de simulatie een zo volledig mogelijk spel te laten simuleren we *Yoki* later makkelijker kunnen koppelen aan een ander spel. Daarnaast erven alle spelers de functie over van de Player class zodat eventuele nieuwe spelers zonder pijn en moeite kunnen worden geimplementeerd. In figuur 4.1zijn de verschillende onderdelen te zien van de simulatie. In de 'game' module zitten de regels van het spel. De 'board' module houdt de status van het spel bij. De 'deck' module is de set kaarten. De 'player' module bevat de status van elke speler en moet worden geimplementeerd door alle overige spelers. De 'handevaluator' module berekent de score van de verschillende kaarten en deze score kan eventueel aangepast worden de 'moduler' module die zich bezig houdt met het modeleren van spelers.

4.3 Implementatie van de hand evaluatie

Zoals al eerder besproken is, is de hand evaluatie functie de meest belangrijke functie en ik heb dan ook een aanzienlijk gedeelte van mijn tijd besteed aan het optimaliseren van deze functie. De truuk tot het snel kunnen uitrekenen van de potentie van een hand is het van te voren berekenen en dan deze waarde opslaan in een hash tabel.

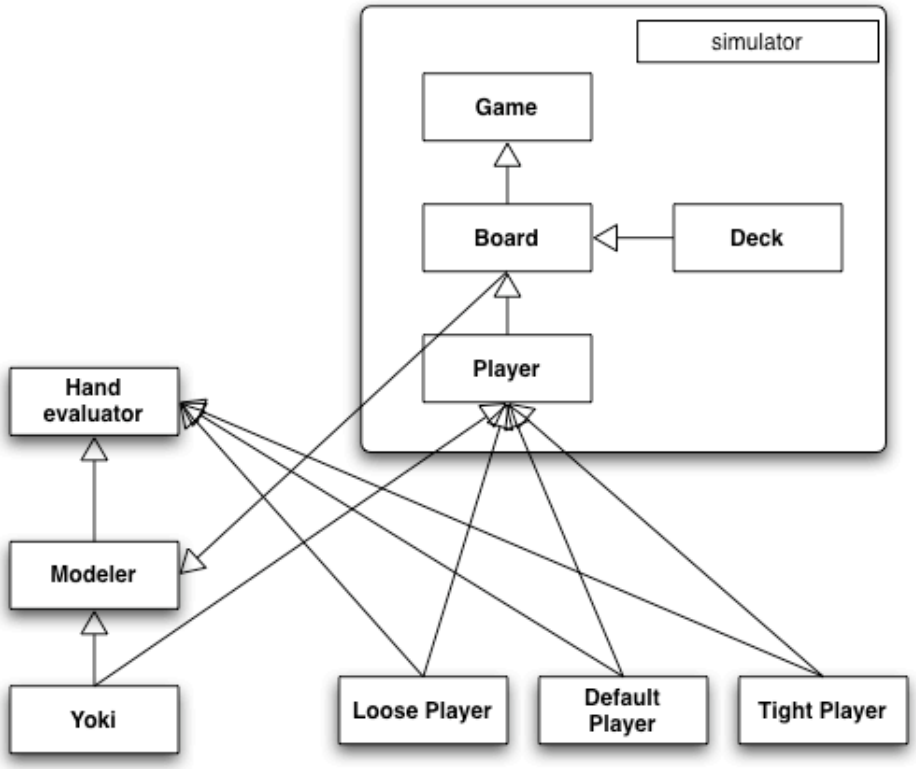
Maar, dit gaat niet zonder meer. In totaal zijn er 132600 mogelijke flops met elk 1176 mogelijke pocket cards. Dit betekent in totaal zo'n 160miljoen mogelijke combinaties. Het berekenen van deze waarde zou gedaan kunnen worden in zo'n 4 uur en het zou qua geheugen zo'n 305mb[2] nodig moeten hebben. Op zich zou dit te doen zijn alleen helaas niet voor de TURN, je hebt dan zo'n 15 gigabyte geheugen nodig en zo'n 8 dagen om te rekenen. Zo veel geheugen heb ik niet en 8 dagen rekentijd is te veel voor een project van maar 24 dagen.

Op een of andere manier moest ik het aantal mogelijkheden terugdringen. Ik heb dit gedaan door de kleur van de kaarten gedeeltelijke te negeren. Wanneer je de kleur van de kaarten negeert heb je niet 52 verschillende kaarten in een deck maar 13. Met deze 13 heb je maar 13*12 mogelijke pocketkaarten. Wanneer je binnen deze pocketkaarten ook nog eens de dubbele wegstreept houdt je maar

[1] Al deze resultaten zijn behaald op mijn Powerbook G4 867mhz uit 2001. Wanneer ik de test zou draaien op de machines in Euchlides zou dit een versnelling opleveren van bijna 10x.

[2] Aangezien de waarde die het moet bevatten opgeslagen kan worden in een 'short' wat maar 2 bytes nodig heeft.

Figure 4.1: Overzicht van de simulatie



91 over (A2 is immers gelijk aan 2A). Hetzelfde kan je ook doen voor de FLUSH en de TURN.

Het mooie aan dit is dat wanneer je de FLUSH kaarten negeert, de overgrote meerderheid van combinaties gewoon blijft bestaan. De enige die je niet meer kunt maken is de FLUSH en de STRAIGHTFLUSH. Om dat op te vangen heb ik de volgende truuk bedacht. De score van een HARTENFLUSH KONING-HOOG is gelijk aan de score van een SCHOPPENFLUSH KONING-HOOG. Voor een FLUSH maakt de specifieke kleur niet uit zolang er maar 5 kaarten de zelfde kleur hebben. Met deze informatie gaan we verder. De 455 mogelijke FLOPS hebben op dit moment allemaal nog geen KLEUR, we stellen nu een bepaalde KLEUR in als de FLUSH KLEUR, zeg maar de kleur die mogelijk een FLUSH maakt. Dan doen de overige kleuren niet toe. We kunnen deze kleur bij de FLOP op 7 ($2^3 - 1$) verschillende manieren encoderen[3]. Wat betekent dat we in totaal 3158 (7*455) mogelijke flops hebben. Een dergelijke truuk kunnen we ook uithalen voor de TURN. Het is overigens wel interresant dat waar eerst vooral de FLOP moeilijk was om uit te rekenen door de extra diepte van twee onbekende kaarten. Is het nu de TURN door de extra combinaties en dus de groter benodigde hash tabel[4].

4.3.1 Het opslaan van de BOARD waarden

Het opslaan van de boardwaarden gebeurd door over alle mogelijke flops in combinatie met alle mogelijke pocketodds te enumereren. Het is soms mogelijke dat bepaalde combinatie niet mogelijk zijn, daarnaast is het zo dat de tabel niet gelijk verdeeld is. Sommige waarden tellen zwaarder omdat ze vaker voorkomen. Hier zal rekening mee moeten worden gehouden bij het terug vragen van deze waarden. Python code die de waarde berekent en opslaat voor alle mogelijke flops staat beschreven in Algorithm: 1

4.3.2 Het Terugvragen van de BOARD waarden

Het terug opvragen van de score van een board gaat dan als volgt:

1. Creeer een hash van het board
2. Bekijk de suit signature van het board
3. Haal deze waarde uit de hash tabel
4. Normaliseer de waarden

[3] [1],[2],[3],[1,2],[1,3],[2,3],[1,2,3] waarbij het getalletje binnen de haken aangeeft welke kaart de FLUSH KLEUR is.

[4] In mijn huidige implementatie heb ik 75mb nodig voor het hashtabel van de TURN en 8mb voor die van de FLOP

Algorithm 1 Hoe de FLOP waarden worden opgeslagen

```
def enumerateflop():
 """ Enumerate all possible cards
 """
 floptable = {}
 for flop in allflopcards():
  x,y,z = flop
  fhash = primes[x] * primes[y] * primes[z]
  flopvariations = []
  for flopvariation in [[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]:
   flopcards = setflush(flop, flopvariation)
   if flopcards:
    pocketscore = []
    for pocket in allpocketcards():
     pocketvariations = []
     for pocketvariation in [[0],[1],[2],[1,2]]:
      xx, yy = pocket
      phash = primes[xx] * primes[yy]
      pocketcards = setflush(pocket, pocketvariation, flopcards)
      if pocketcards:
       pocketvariations.append(getouts(pocketcards, flopcards))
      else:
       pocketvariations.append(0)
     pocketscore.append(pocketvariations)
     p.update(1)
    flopvariations.append(pocketscore)
   else:
    flopvariations.append(0)
  floptable[fhash] = flopvariations
 pickle.dump(floptable, open("flopresults.dat", 'w'))
```

4.3.2.1 Creeer een hash van het board

Dit wordt gedaan door de kaart rank om te zetten naar een priemgetal en dan deze priemgetallen met elkaar te vermenigvuldigen. Het voordeel hiervan is dat A2 en 2A dezelfde waarde krijgen en dat je zeker weet dat je een hash krijgt die niet wordt gedeeld door andere kaarten.

4.3.2.2 Bekijk de suit signature van het board

Wanneer we een board binnen krijgen bekijken we de suit signature. Bijvoorbeeld: [“Ah”, “Kh”, “As”] heeft een signature van [1,2,0] wat betekend dat de eerste 2 kaarten de zelfde suit hebben.

4.3.2.3 Haal de waarden uit de hash tabel

Met de hash en de suit signature kunnen we al komen bij de lijst van de 91 verschillende pocket hands en ook weer hun suit signature (4 mogelijkheden. Wanneer we alleen maar een waarde willen weten voor een specifieke pocket waarde zijn we snel klaar en kunnen we deze waarde simpelweg aflezen.

4.3.2.4 Normaliseer de waarden

Wanneer we de verwachtingswaarde willen weten van het board dan moeten we ervoor zorgen dat we de waarden normaliseren. Immers de kans op AA is veel kleiner dan AK, en de kans op weer AKs (in een en dezelfde KLEUR) is weer kleiner. Na het normaliseren kunnen we de waarden eenvoudig sommeren. Overigens wordt bij het normaliseren van de board kaarten ook rekening gehouden met de eigen POCKETCARDS en zullen de kansen in overeenstemming worden aangepaset.

4.4 Implementatie van het modeleren

Bij het modeleren van de tegenstander maak ik gebruik van 2 verschillende soorten statistieken. De eerste is de verhouding van het bieden tegenover het callen en het folden. En de tweede kijkt naar de zogenaamde EXPOSEDPOCKETCARDS om dan achteraf te berekenen hoe een speler gespeelt heeft.

4.4.1 Frequency Distribution modeling

Bij deze aanpak wordt er gekeken naar hoe de verschillende speler acties zich verhouden tot elkaar. Van een voorzichtige speler verwacht ik dat hij in verhouding meer FOLD dan een speler die losjes speelt. De distributie van deze acties pas ik dan daarna toe op gewichten die ik vermenigvuldig met de mogelijke PocketHand kaarten. Het resultaat hiervan is dat acties van een speler in de voorgaande ronden aangeven hoe de speler speelt. Aan een actie in de huidige ronde kan dan geschat worden wat de waarde van de hand is.

Algorithm 2 Het creeeren van de gewichten

```
    def createModelWeights(betFrequency, action):
        """ When given a bet frequency create a list of weights
            and return them
        """
        fold, call, bet = betFrequency
        total = sum(betFrequency)
        fold = int(364.0 / total * fold) # set length
        call = int(364.0 / total * call)
        bet = int(364.0 / total * bet)
        # create raise weights
        rweights = zeros([364], dtype=Float32)   # 91*4
        cursor = 364 - bet
        for i in range(bet):
            x = float(i)/bet
            rweights[cursor] = 1.0 - 1.0 / (1 + math.e ** (10*(x - 0.5)) )
            cursor += 1
        if action == "raise":
            return rweights
        # create fold weights
        fweights = zeros([364], dtype=Float32)
        for cursor in range(fold):
            x = float(cursor) / fold
            fweights[cursor] = 1.0 / (1.0 + math.e ** (10*(x - 0.5)) )
        if action == "fold":
            return fweights
        # create call weights
        cweights = ones([364], dtype=Float32)
        return cweights - fweights - rweights
```

Bijvoorbeeld, wanneer de acties van een speler zich verhouden als 1:1:1. Betekent dit, dat wanneer de speler verhoogt hij dit waarschijnlijk doet voor de 30% beste kaarten. Wanneer *Yoki* een speler in een ronde ziet verhogen past hij de gewichten aan voor de top 30% kaarten aan zodat de kans hier op groter is. De kans op de andere kaarten worden verlaagd. Hierdoor zal de schatting wat de PocketCards zijn van een speler veel exacter worden.

Het aanpassen van de gewichten doen we met de zogenaamde sigmoid functie om een mooi gelijdelijke curve te krijgen. Hoe ik dit precies doe kun je vinden in algoritme:2

4.4.2 ExposedPocketCard Modelling

Bij deze aanpak word nadat een tegenstander zijn pocketcards zichtbaar zijn geworden de ronde opnieuw doorgerekend maar dan vanuit het oogpunt van de

tegenstander. Er wordt een gemiddelde bijgehouden van de waarden waarop de speler inzet of called. Het idee hier achter is dat dit als het ware de intentie van de speler weergeeft en misschien in grotere mate betrouwbaarder is dan de verdeling van acties.

Chapter 5

Conclusie

5.1 Resultaten

Het modeleren van tegenstanders lukt aardig goed zoals te zien is in figuur 5.2. Deze gewicht verdeling is gemaakt door *Yoki* en laat duidelijk een verschil zien in de drie speel stijlen. Helaas ben ik tijd te kort gekomen om te kijken of ik deze kennis ook op een bepaalde manier kan exploiteren. Maar dit resultaat is in iedergeval al bemoedigend. Ik vind het overigens zeer jammer dat ik net tijd te kort kom om de resultaten van dit modelern nog mee te nemen in dit verslag. Ik denk dat het implementeren van een manier om een geleerd model toe te kunnen passen nog ongeveer een dag kost. Ik ben dan ook reuze benieuwd hoe *Yoki* die dus gebruik maakt van correcte hand evaluatie functie in tegenstelling tot *Loki* zal presteren tegen deze bot.

Daarnaast is het ook fijn om te weten dat de evaluatie functie goed werkt en dat de simulatie (na flink debuggen) goede resultaten geeft zoals te zien is in figuur 5.2. In dit figuur kun je zien dat het losse spelers model overduidelijk beter presteert dan de andere modellen.

5.2 Vervolg onderzoek

Er is nog veel te doen. Ten eerste moet er voor gezorgd worden dat *Yoki* goed gebruik kan maken van de door hem geleerd modellen. Daarna is het interresant om te kijken hoe het bieden van *Yoki* kan worden verbeterd. Je kan hierbij denken aan MonteCarlo simulaties of misschien is de state space zelfs wel terug te brengen tot iets wat behapbaar is met een POMDP.

Ook aan de dynamiek binnen een ronde kan nog genoeg gedaan worden. Hierbij valt zeker meer te doen met de PostPocketOdds Modelling.

Figure 5.1: Modeleren van Tegenstanders door Yoki

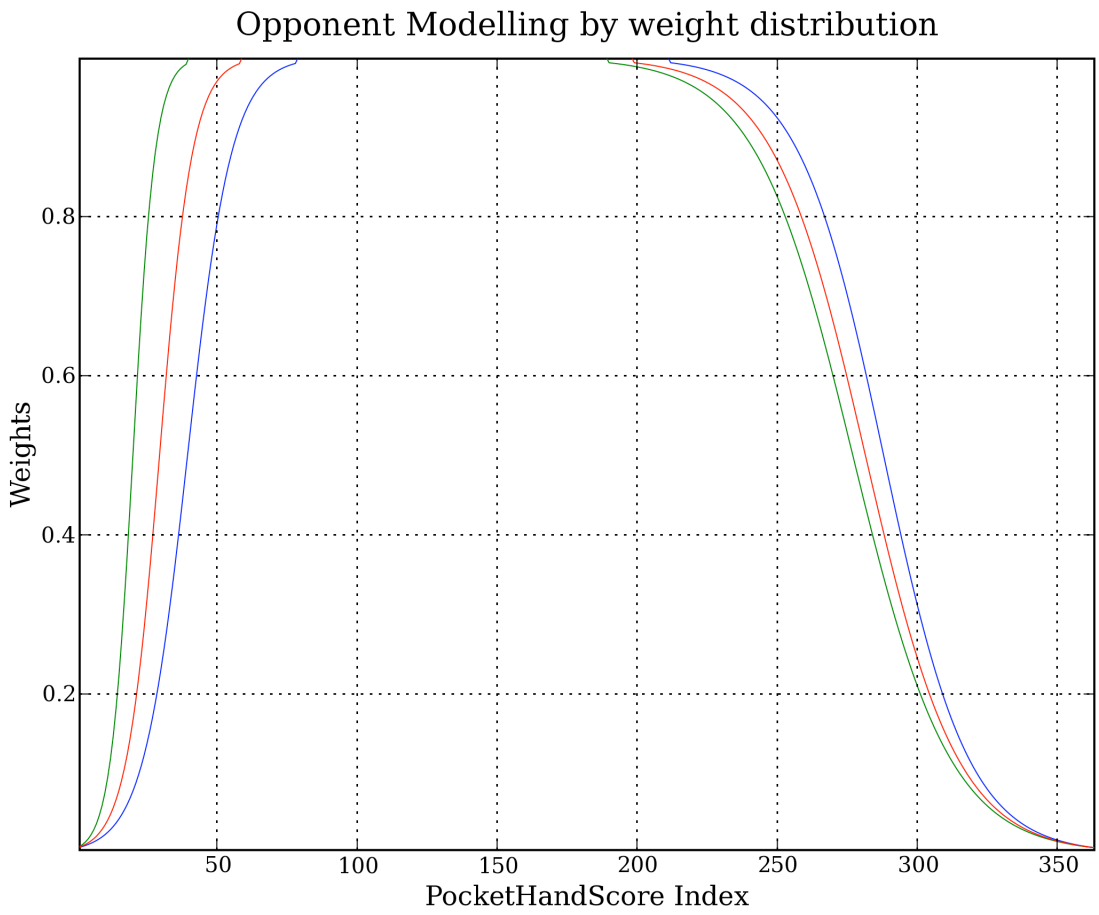
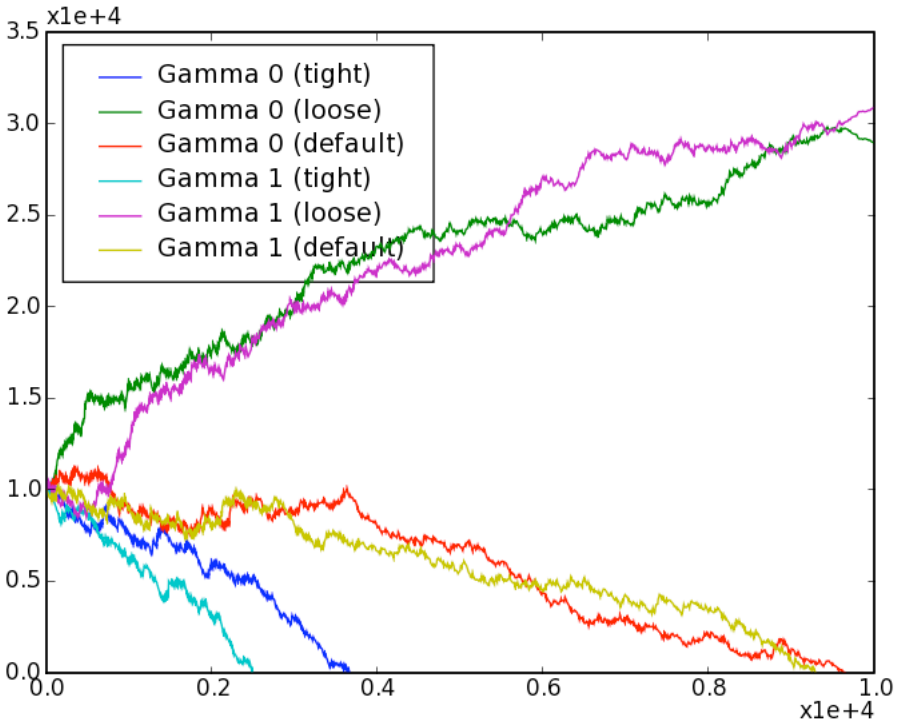


Figure 5.2: Een simulatie van 4 verschillende spelers modellen



Glossary

pocketodds De kans die de kaarten zijn toebedeelt die je aan het begin krijgt gedeelt.

pocketcards Je eigen (verborgen) kaarten.

board De publieke kaarten

hand Een combinatie van 5 mogelijke kaarten

flush Alle kaarten in de zelfde kleur

straigth 5 opeen volgende kaarten

straigthflush Een straat in een kleur

kleur De kleur van een kaart (schoppen / harten / klavers / ruiten)

sitin Een plek bezitten en actief meedoen

sitout Een plek bezitten maar niet actief mee doen aan het spel

suit Zie kleur

flop De eerste 3 open kaarten

turn De 4e open kaart

river De laatste open kaart

dealer De speler die moet beginnen met bieden

potodds De kans dat de inzet zich terug verdient door de grote van de pot. Immers een grote pot rechtvaardigt het om meer risico to nemen.

blinds De spelers die eerst een van te voren bepaald bedrag moeten inzetten voordat ze mogen spelen.

blind Het bedrag dat minimaal moet worden ingezet. Meestal bestaat er ook nog een small en bigblind. Waarbij de bigblind wordt gestelt voord de laatste speler en de small blind voor de 1na laatste speler.

showdown De laatste bied ronde

Bibliography

[1] Darse Billings. Computer poker. Department of Computing Science, University of Alberta, 1995.

[2] Theodore Cage. In *Insiders secrets to playing texas hold'em poker online*, 2004.

[3] Denis Pap Duane Szafron Darse Billings, Jonathan Schaeffer. Opponent modeling in poker. Department of Computing Science, University of Alberta, 1998.

[4] O Morgenstern J von Neumann. Theory of games and economic behaviour. Princeton, Princeton University Press, 1944.

[5] Lourdes Pena Duane Szafron Jonathan Schaeffer, Darse Billings. Learning to play strong poker. Department of Computing Science, University of Alberta, 1999.

[6] Paul Lu Martin Bryant Jonathan Schaeffer, Robert Lake. Chinook: The world man-machine checkers champion. In *AI Magazine, vol. 17, no. 1, pp. 21-29*, 1996.

[7] Frans Oliehoek and Nikos Vlassis. Dec-POMDPs and extensive form games: equivalence of models and algorithms. IAS technical report IAS-UVA-06-02, University of Amsterdam, Intelligent Systems Labaratory, Amsterdam, The Netherlands, April 2006.

[8] Paul Senzee. Some perfect hash. http://senzee.blogspot.com/2006/06/some-perfect-hash.html, June 2006.

[9] Kevin Suffecool. Cactus kev's poker hand evaluator. http://www.suffecool.net/poker/evaluator.html, 2005.