Massachusetts Institute of Technology

Poxpert+, the intelligent poker player
v0.91

Meshkat Farrokhzadi
6.871 Final Project
12-May-2005

Joker's the name, Poker's the game.
Chris de Burgh – Spanish train

Introduction

As though it is a sign of our childhoods, all of us still have a tendency toward playing games. Some of us however, take another step further and become experts in one or more games. The most fascinating games among all are probably the mind games. Mixed with logic, psychology and science many of these games require tremendous dedication, concentration and studying to master. Chess and bridge are two examples of these kind of games. On the other frontier of games, there's gambling. A huge source of income, a big industry and a luxurious entertainment for many, gambling has been around since the oldest of times.

On the borderline of these two categories of games, there's Poker; partly gambling, partly mathematics and partly psychology, Poker is about using the brain to guide the luck to the gates of fortune. In this paper, I tried to analyze the process of thought of a skillful, professional poker player and verbalize and capture the underlying rules in an expert system, the Poxpert. Poxpert is an effort to mimic, capture and perhaps improve, if possible at all, the thinking process of a professional player.

First I will talk about the task and what it is that the program does, then I would briefly talk about how I gathered the "knowledge" and how I deducted the rules, and finally I will discuss the underlying structure of the program. There are some examples of how the program works, a graph containing the rules and a discussion about the shortcomings and possible areas for future work.

The task

Poxpert (short for poker expert) will act in one round of betting of the Texas Hold'em All flavor of Poker. Texas Hold'em is the most popular poker game, mainly because a player has to have some set of skills to be successful and good players develop variety of spectacular and sometimes astonishing strategies to outplay their opponents.

The game is played with a typical 52 card deck. Texas Hold'em all is played with as little as two players (which is called heads up) and the maximum number of players is usually eleven. Poxpert is designed to play in a five player game, but by using the general table characteristics (amount to call, etc…) it can play in different setups without specifying number of players. The hand rankings are similar to the other poker games. The typical Hold'em game has five different stages.

Pre-flop

Each player is dealt two cards face down. The first betting round begins with the first player to either putting in enough money to "Call" the blind bet, or putting in more to "raise" or folding his hand. The betting goes around the table in order until it reaches the first player. The three main options based on the position are: calling (or checking if the amount to call is zero), raising (or betting if there has been no bets before) or folding which is throwing away the cards and not participating in that round of play, however the player loses all the bets that she has made in that round (all the bets that are in the "pot").

The dealer which identifies the position is a actually a button that says "dealer" and is passed around the table after each hand. It signifies where the dealing is done from and which player has to act.

Flop
After the first round, three cards are dealt and turned face up in the middle of the table. These are community cards used by all the players. Then another betting round starts.

The Turn
When the betting is completed, the dealer turns a fourth card face up in the middle of the table. The third round of betting takes place. The dealer turns over another card making four community cards. This fourth card is called "the turn" or sometimes "fourth street". Betting occurs again.

The River
The dealer turns over the fifth and last community card. This is called "the river" or "fifth street". Betting begins for the last time.

Showdown
To determine the winner, the players may use any combination of their two hole cards and the five cards on the "Board" (Table) to form the highest five-card hand. The winning hand is determined based on the usual rankings for Poker games with straight flush and four of a kind being the best hands.

Poxpert will take part in the game and it can play at any different stage of the game. However, it can not play continuously and does not use a playing history for making its decision. Although this is a major limitation of the program, for the purpose of this project, it helped in focusing on the rules, gaining insight about the representation choices and paying enough attention to other important parts like the hand scoring function.

<u>Gathering the knowledge</u>
I used several sources for acquiring the information. First, there was my own personal experience as an amateur poker player. My experience comes from playing in tournaments, reading books and watching professional players in action. When working on this project however, I tried to refine my previous findings and add to them, so I went back to the books and I tried to watch the important events like "World Series of Poker" which is held annually in Vegas with over five million dollars worth of winnings. Interestingly enough, year after year, the same faces appear at the final tables of WSOP, the same professional players keep winning large sums of money. Well, could there be any better indication that Poker is in fact a game of skills and not only luck?

I also used the web to search for previous works that people have done. I was unable to access anything worthy that could help in building the rules or implementing the hand score function. Most of the online resources are not open source, even the University of Alberta that runs a game group had commercialized its poker program. Nevertheless, I was able to find some discussions about the hand rankings, some statistics about the winning hands and some previous statistical work in the field.

The basic and significant part of the job was to focus on deducting the rules. There are several different factors playing at the same time, and it's not easy to divide the actions between the variable combinational space. Also, rule deduction from the ambiguous book descriptions or player recommendations is not trivial. Many professional players when faced with the questions about how they decide to act would say that they just act "intuitively" and they just know what to do at any given time. A common expression is that they have a good "read" on others, but nobody knows how exactly this works. So one of the challenges was to extract concrete and useful facts out of the stories and legends (every poker player wants to tell the story of his worst loss and his best victory).

After writing down the basics, I had to choose the variables and find the connections between them. This task appeared to be one of the most time consuming and challenging issues I faced in this project. The main difficulty appeared when I was trying to implement what I had in mine in Joshua. Having no prior experience with this programming language I spent so many hours trying to update the value of a variable and changing my representations. At last I learnt that not every tool is suitable for solving every problem. Joshua is most capable of dealing with situations in which facts are established once and no changes happen thereafter. It provides the user with a decent snapshot of any given situation. In poker however, one usually needs to capture different variable states at different points of time. Updating and replacing variables proved to be almost impossible in Joshua. This major problem prevented me from implementing some of the functionalities that I had in mind and it also affected the program interface, which may ask redundant questions at times. The challenge was to not let this shortcomings affect the intelligence of the program. Although the interface is not as fancy as it could, I'm willing to claim with that the performance is almost as good as other implementations that I had in mind.

The program solving paradigm

I used Joshua and forward/backward chaining rule-based system to implement this program. The advantage of this method is in its power of firing different rules and coming up with relativistic answers. Having a certainty factor helped a lot in defining the problem, so for example in some cases the program can produce different actions and various rules may trigger identical or contrasting actions, but the certainty factor differentiates between the correct and incorrect decisions to a great extent.

The rule-based system seems somehow like a natural way of solving the problem of modeling the expert's thought process. Poker professional players usually look at different factors to make their decisions, for example if they have a lot of chips and their opponent is short on money (short-stacked) they will try and push their opponent to make aggressive calls. Although the hand score is the primary criterion for making the decisions, but as just shown, expert players' actions are not solely based on their hand strength and the possibility of winning the pot. Good players never rely on good hands as the only source for winning the chips. Bluffing is a well established way for winning the pots by scaring the other players and pretending to have a very strong hand.

However, there are certain shortcomings in dealing with the more abstract concepts in a rule-based system. There was hardly anyway in which I could model a player and change that model later. The rule based system asks for the truth and false claims. Once the facts are in the database, one can hardly alter them. This was a major limitation of this model, because in the real world, assertions change frequently; a player changes her strategy every now and then and your confidence goes up and down; the less confident you become, the less likely it is that you bluff. In Joshua, changing a variable in the database affects all the rules and facts that were dependent on that fact. This rule-based system lacked the notion of time, something essential to the game of poker, in which a good player puts the facts on top of each other to make a model and modifies the constituent assertions when they don't fit into the model anymore.

One of the main parts of the knowledge base is the function that calculates player's hand score. Basically, this function gives the probability of winning the pot, conditioned on the hole cards and the cards on the board. Having so many difficulties in Joshua, I wrote this function is Java which can be used to calculate the hand's strength which is an input for the main program. I could've probably implemented my own rule-based system in another language with fewer limitations, but I have to admit that struggling with Joshua taught me a lot; especially because I had to think about how to choose my variables and represent the knowledge within the limitations of the environment.

The case-based reasoning could also be used as one of the methods for solving this problem. However, the number of cases that one has to deal with in order to have a decent action in a card game like Poker is almost infinite. In determining the hand's score and for the two pocket cards, I used the statistical results from winning bets in over millions of games. Capturing these many cases in a system could be very time consuming and costly in the processing time. However, the cased-based reasoning could be adequately used for improving the program's performance. The program can use its failure cases for "learning" and not repeating its mistakes. Also, they could be used as a source of feedback to the programmer for refining the rules.

Another representation that I considered was the frames. Frames are fancy, they provide as close a model as possible for showing how we think and they are as fuzzy and as vague as every human decision making process could be. The problem with the frames is not their representation power, but it's the fact that there is no practical way of implementing a system with the frames representation. Although interesting; the lack of tools and previous experience that I faced with made frames one of the issues for further study and considerations. One of my ideas was to use frames for representing the players and strategies, rule based system for inference and reasoning and cased-based reasoning for automated testing and refinements. Whether this idea is practical or not could be a subject for another discussion.

<u>What does the program know?</u>
Poxpert knows how good every hand is. It uses the "handScore" function to find out the probability of winning based on its hole cards (faced-down) and the board cards. It knows the basics of the games, including the winning hands, the possible actions, the pot size and the

minimum amount to call. Poxpert asks for a variety of information and uses those inputs to deduct new facts.

The program tries to adapt a strategy based on its hand score, the amount of bets, the amount of money on the table (pot size), its chip count and some other factors. Once it chose a strategy it searches through the possible actions with relative certainties and picks the action with yields the highest result. If it chooses to raise, it also decides on the amount of raise. For example if the betting is low and Poxpert has an average or poor hand and a good position on the table (being the last person to call), it usually decides to bluff by raising the pot to a considerable amount (5 times the big blinds). In real world, this strategy usually works, at Pre-flop stage this method typically leads to a strategy commonly known as "stealing the blinds". The other players who have worse positions always have to act before you and so the program uses its position edge to intimidate the others.

Poxpert knows about these little advantages, namely chip advantage, position and hand advantage. It also knows about the betting patterns and their relevance with the table stage, for instance if the betting amounts are relatively high and the dealer has just dealt the river card, there's highly unlikely that anybody would be bluffing and more than likely someone has a really good hand. The program also knows about what is known as "draws" and waiting for a draw and it uses this piece of knowledge in both the "handScore" function and the main body of code. A draw happens when a player has four cards belonging to a suit and is waiting for the fifth card to hit and make a "flush". A player in this situation would hardly get out of the pot unless after the "turn" or fourth street.

How does it work, what's under the hood

There are three possible actions at any given time, folding, raising or calling. Other actions like checking can be included in calling with zero amount of money and betting is raising as the first player. So at the very top level the Poxpert selects at least one of these actions together with a certainty factor corresponding to that action. Several rules may result in firing the "fold" rule for example in which the certainty of "folding" goes up.

The rules may even result in different actions which actually resembles the challenges that every poker player has to face. Sometimes there is no single right or wrong action in the game and the player has to be random and make a decision. The program in these cases presents the player with different options and their corresponding certainties. Usually there is enough discrepancy between the certainties to allow the player pick the best choice easily. Therefore, close values (in rare cases) are not an indication of program's bad behavior; they in fact reflect the real nature of the game.

Some rules, for example many folding rules use the hand strength as the main criterion for decision making. Although the final decision is usually made based on a combination of different variables, for example let's look at rule action-fold1:

```
(defrule action-fold1 (:backward :certainty 0.9 :importance 250)
  if  [and [hand-score poxpert ?score]
      (> ?score 0.1) (< ?score  0.3)
      [seat poxpert ?seat] (> ?seat 1) (< ?seat 5)]
  then [action poxpert fold])
```

In English, this rule says that if the hand score is between .1 and .3 (relatively weak hand) and the program's position is 2, 3 or 4 (bad position), then the program is supposed to fold with a 0.9 certainty. This is a classic situation in which calling would more than likely result in more losses because bad position means that the players after Poxpert may raise the pot and then the program has to fold which results in losing the original bet.

Let's explain another rule, action-raise1:

```
(defrule action-raise1 (:backward :certainty .90 :importance 225)
  if [and [hand-score poxpert ?score](> ?score .7) (< ?score .9)
     [amount-to-call poxpert ?coc] [big-blind table ?bb]
     (> ?coc (* 2 ?bb))]
  then [action poxpert raise])
```

Again, this rule says that if the hand score is between .7 and .9 (relatively strong hand) and the amount to call is more than two times the amount of big blinds then the program should raise. The reason behind this rule is that if the amount to call is more than two big blinds, someone has already raised the pot and therefore the probability of everybody else folding and decreasing potential winnings is relatively small (they're more than likely to call if they have raised before). So raising here is a safe action.

As the final example let's look at cost-of-calling2:

```
(defrule cost-of-calling2 (:backward :certainty 1.0 :importance 170)
  if [and [amount-to-call poxpert ?cost] [pot-size table ?pot]
     [big-blind table ?bb] [chip-count poxpert ?money]
     (>= ?cost (max (* .5 ?pot) (* 5  ?bb) (* .1 ?money))
     (<= ?cost (max (*  2 ?pot) (* 10 ?bb) (* .3 ?money))]
  then [cost-of-calling poxpert moderate])
```

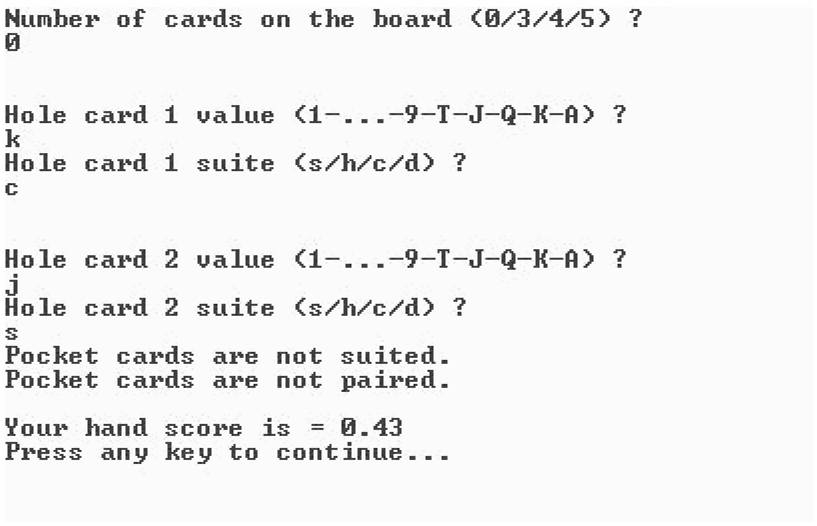
This rule is used to determine whether the cost of calling is high, average or low for the program. The cost of calling depends on four different variables, the amount to call, the pot size (total bets already on the table), the program's fortune (chip count) and the amount of big blinds (minimum bets). This rule takes the maximum of the last three factors scaled by different amounts and compares them to find the amount of betting. The variables and scaling factors are deducted from rules and experience, for example any bet more than ten percent of program's total chip count is always considered an expensive call.

Walking through an example

To start the trace one has to use the following command:

```
(ask [action poxpert ?x] #'print-answer-with-certainty)
```

The program tries the first rule, action-fold0. It is always wise to check for folding situations first since it's almost always safer to fold and lose nothing than to take the risks with a bad hand and lose a lot. The rule needs the hand score to decide how good the hand is, now suppose we are dealt an average hand, King of clubs and Jack of spades. Now we use the handScore function to find the score. For the hole cards without any cards on the board, I used a statistical study of the winning hands that is well established among the poker players and is available online [1][2]. For the other stages when there are faced up cards on the table, the program calculates the possible hands and the best possibilities for winning the pot based on available combinations of the different hands and produces an output. Here's a snap shot of the program input and outputs.



```
Number of cards on the board (0/3/4/5) ?
0


Hole card 1 value (1-...-9-T-J-Q-K-A) ?
k
Hole card 1 suite (s/h/c/d) ?
c


Hole card 2 value (1-...-9-T-J-Q-K-A) ?
j
Hole card 2 suite (s/h/c/d) ?
s
Pocket cards are not suited.
Pocket cards are not paired.

Your hand score is = 0.43
Press any key to continue...
```

Figure 1 – handScore function

So the handScore is 0.43, let's get back to the main program.

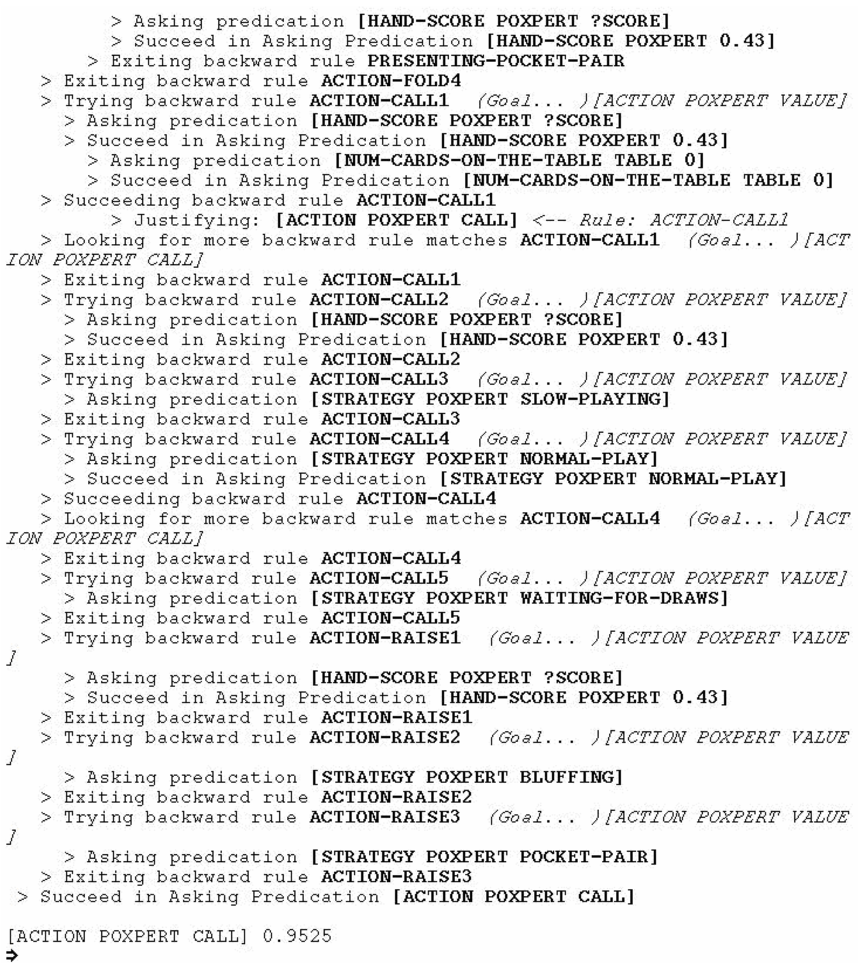
```
What is Poxpert's hand score? .43
What is the current number of cards on the flop?
```

This is a moderate hand with probability of winning less than half, the program exits from the first three folding rules because the hand score is not too low, the fourth folding rule however checks to see if the amount of calling is too expensive and the hand score is not that great then the program wants to fold. Now the program wants to know how much is the calling cost, this fact comes from the table positioning and the player actions who are sitting before the program. The program therefore first checks to see if Poxpert is the first player to call, this happens if this a pre-flop table and Poxpert is at position three or if the table is at any other stage and the Poxpert is at position two (corresponding rules are "how-much-to-call1 and how-much-to-call2"). Let's assume that the program is at position four and we're at pre-flop stage, the big blinds are 50 dollars, Poxpert has 2500 dollars, the player before Poxpert has just raised to 250 dollars and the total pot size is 400 dollars. These facts all come from a hand of Poker and are usual inputs to the program and the factors that every player considers in trying to make a call.

```
What is the current number of cards on the flop? 0
What is Poxpert's position? 4
What is the Poxpert's cost of calling? 250
What is the table's pot size? 400
What is table's big blind? 50
What is Poxpert's chip count? 2500
```

The program now realizes that since the cost of calling is relatively cheap (from "cost-of-calling3") because the amount of bet is not bigger than five big blinds or ten percent of Poxpert's chips. The folding rule therefore does not suit this occasion since the cost of calling is not expensive. The program then checks through the calling rules where it has to pick a strategy (obviously, there was no need to pick a strategy if the program was folding) and based on it's hand score, table position and calling-cost it picks the "normal-play" strategy. This actually makes sense since the Poxpert has an average hand and there are still two more cards to come, folding with an average hand and this pot size is not the right thing to do and raising is just out of question since another player might re-raise. The program checks all the other rules and succeeds in firing action-call1. The final result is very reasonable, a snapshot of the tracing can be seen in figure 2.

```
[ACTION POXPERT CALL] 0.9525
```



```
          > Asking predication [HAND-SCORE POXPERT ?SCORE]
          > Succeed in Asking Predication [HAND-SCORE POXPERT 0.43]
        > Exiting backward rule PRESENTING-POCKET-PAIR
    > Exiting backward rule ACTION-FOLD4
    > Trying backward rule ACTION-CALL1  (Goal... )[ACTION POXPERT VALUE]
      > Asking predication [HAND-SCORE POXPERT ?SCORE]
      > Succeed in Asking Predication [HAND-SCORE POXPERT 0.43]
        > Asking predication [NUM-CARDS-ON-THE-TABLE TABLE 0]
        > Succeed in Asking Predication [NUM-CARDS-ON-THE-TABLE TABLE 0]
    > Succeeding backward rule ACTION-CALL1
          > Justifying: [ACTION POXPERT CALL] <-- Rule: ACTION-CALL1
    > Looking for more backward rule matches ACTION-CALL1  (Goal... )[ACT
ION POXPERT CALL]
    > Exiting backward rule ACTION-CALL1
    > Trying backward rule ACTION-CALL2  (Goal... )[ACTION POXPERT VALUE]
      > Asking predication [HAND-SCORE POXPERT ?SCORE]
      > Succeed in Asking Predication [HAND-SCORE POXPERT 0.43]
    > Exiting backward rule ACTION-CALL2
    > Trying backward rule ACTION-CALL3  (Goal... )[ACTION POXPERT VALUE]
      > Asking predication [STRATEGY POXPERT SLOW-PLAYING]
    > Exiting backward rule ACTION-CALL3
    > Trying backward rule ACTION-CALL4  (Goal... )[ACTION POXPERT VALUE]
      > Asking predication [STRATEGY POXPERT NORMAL-PLAY]
      > Succeed in Asking Predication [STRATEGY POXPERT NORMAL-PLAY]
    > Succeeding backward rule ACTION-CALL4
    > Looking for more backward rule matches ACTION-CALL4  (Goal... )[ACT
ION POXPERT CALL]
    > Exiting backward rule ACTION-CALL4
    > Trying backward rule ACTION-CALL5  (Goal... )[ACTION POXPERT VALUE]
      > Asking predication [STRATEGY POXPERT WAITING-FOR-DRAWS]
    > Exiting backward rule ACTION-CALL5
    > Trying backward rule ACTION-RAISE1  (Goal... )[ACTION POXPERT VALUE
]
      > Asking predication [HAND-SCORE POXPERT ?SCORE]
      > Succeed in Asking Predication [HAND-SCORE POXPERT 0.43]
    > Exiting backward rule ACTION-RAISE1
    > Trying backward rule ACTION-RAISE2  (Goal... )[ACTION POXPERT VALUE
]
      > Asking predication [STRATEGY POXPERT BLUFFING]
    > Exiting backward rule ACTION-RAISE2
    > Trying backward rule ACTION-RAISE3  (Goal... )[ACTION POXPERT VALUE
]
      > Asking predication [STRATEGY POXPERT POCKET-PAIR]
    > Exiting backward rule ACTION-RAISE3
  > Succeed in Asking Predication [ACTION POXPERT CALL]

[ACTION POXPERT CALL] 0.9525
➔
```

Figure 2 – Snapshot of the tracing through the example

The scope of the program

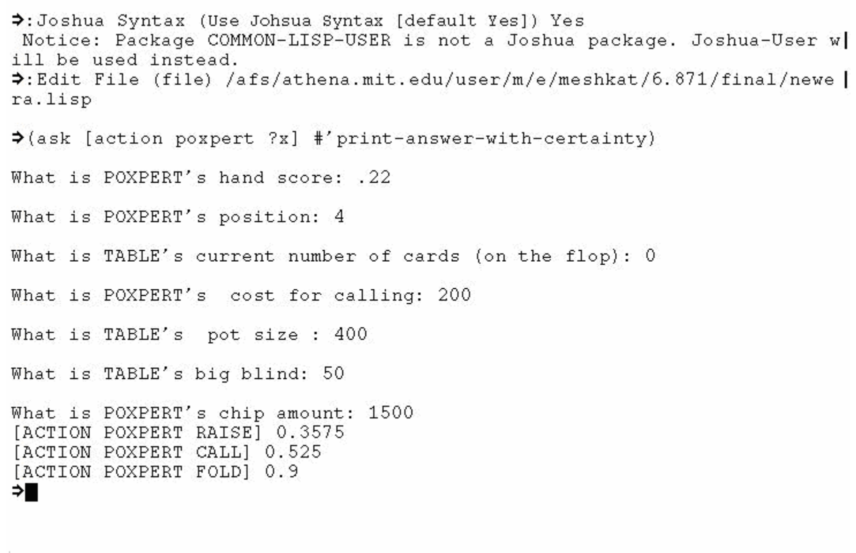
Poxpert participates in one round of betting of the Texas Hold'em version of poker. It doesn't keep a history of previous betting and other players' actions. Therefore all the inferences that it makes are based on the current hand only which is a limitation of the current version of the program.

Because of the nature of the rule-based system, the program would reproduce the same results for the same setups. Although some random and concurrent actions may be suggested, in general case the program would suggest the same actions (and presumably the user would also select the action with the highest score). A consequence of this is that any smart player or another poker bot that saves the history of the games after a number of rounds (about a thousand rounds at least to exhaust all the rules and observe the repetitions and show downs) may be able to identify Poxpert strategies and decision making pattern. That person then could adapt her strategies accordingly to beat the program.

The program is designed to work best playing against five other opponents but can also be used in different settings. The rules have to change slightly if the number of players increases to ten. The program cannot make profiles for its opponents and therefore tries to make an estimate of the table strength by looking at its position, the amount of bets, raises, table stage and other factors which turns out to be just as efficient in this setting.

What went well, what went badly

The rule adaptation turned out to be efficient and the performance of the program is satisfactory most of the times. I was very excited to see the interaction of different rules that could result in proposing multiple actions with different certainties since I was struggling to include the notion of randomness in Poxpert. Figure 3 demonstrates one of the occasions that program makes multiple suggestions and it correctly points out that folding is the most suitable action.



```
➔:Joshua Syntax (Use Johsua Syntax [default Yes]) Yes
 Notice: Package COMMON-LISP-USER is not a Joshua package. Joshua-User w|
ill be used instead.
➔:Edit File (file) /afs/athena.mit.edu/user/m/e/meshkat/6.871/final/newe |
ra.lisp

➔(ask [action poxpert ?x] #'print-answer-with-certainty)

What is POXPERT's hand score: .22

What is POXPERT's position: 4

What is TABLE's current number of cards (on the flop): 0

What is POXPERT's  cost for calling: 200

What is TABLE's  pot size : 400

What is TABLE's big blind: 50

What is POXPERT's chip amount: 1500
[ACTION POXPERT RAISE] 0.3575
[ACTION POXPERT CALL] 0.525
[ACTION POXPERT FOLD] 0.9
➔█
```

Figure 3 - sample interaction with the program, producing multiple results with different certainties

The experience of working with Joshua was both frustrating and rewarding. The language is very limited in its capabilities to reflect problems with a time varying nature, but it's very effective and informative in dealing with the typical decision making problems. I had to change the way my rules worked several times, I changed the variables and the way questions were posed to the user. First I aimed to have a much more user-friendly system in which user only had to enter the amount of bets and positions and the program would extract all the other information from this set of data. But working with database variables and using Lisp variables in body of Joshua statements without messing the previous statements proved to be a very hard if not impossible task. Therefore I had to change the system to ask the questions more directly. I also implemented the handScore function in Java due to the aforementioned limitations of Joshua.

It's always inspiring and rewarding to see the whole system working and producing reasonable results which takes away the memory of sleepless nights wrestling with Joshua and browsing through manual pages. The challenge taught me that the first step in modeling every abstract concept is probably the most important, and that is selecting the presentation.

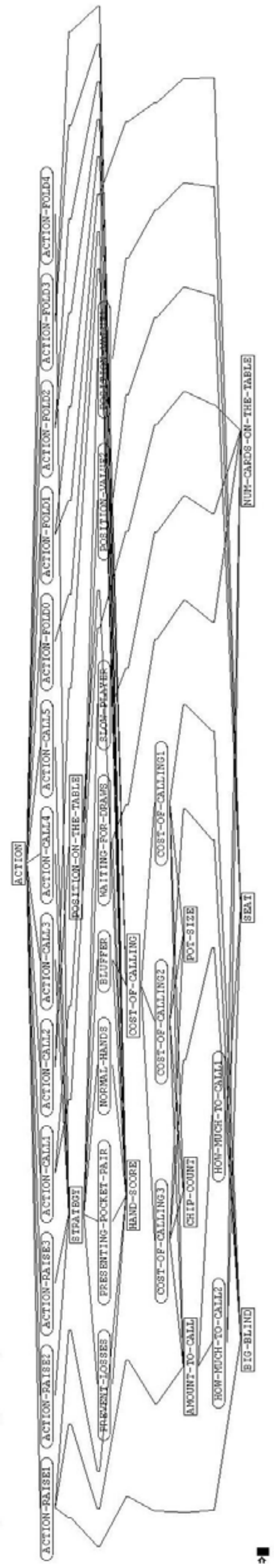
The rules have comments that describe why they act in a certain way. Tracing option can be chosen to walk through an example since the rules are complex and one example requires going through many pages of explanation.

Acknowledgments
Special thanks to Professor Davis for answering my last minute questions and inspiring me by giving suggestions about the Poxpert and of course this year's Mine Sweeper's contest.

References:

1- *http://www.pokerinspector.com/screen2.php*
2- *http://teamfu.freeshell.org/poker_hands.html*

Appendix A – Rules Graph



Appendix B- Joshua Rules

```
;; List of rules
;; rules for action are followed by rules that determine strategies
;; and rules to determine if calling is expensive or not and also
;; rules that decide about the amount of raises and the position edge
;; on the table
;; For starting the program use:
;; (ask [action poxpert ?x] #'print-answer-with-certainty)


;; rules for folding
(defrule action-fold0 (:backward :certainty .95 :importance 251)
  if [and [hand-score poxpert ?score] (< ?score .1)]
  then [action poxpert fold])

(defrule action-fold1 (:backward :certainty 0.9 :importance 250)
  if  [and [hand-score poxpert ?score] (and (> ?score 0.1) (< ?score  0.3))
      [seat poxpert ?seat] (and (> ?seat 1) (< ?seat 5))]
  then [action poxpert fold])

(defrule action-fold2 (:backward :certainty 0.85 :importance 249)
  if  [and [hand-score poxpert ?score] (< ?score .40)
          [cost-of-calling poxpert ?coc] (not (eq ?coc 'cheap))]
  then [action poxpert fold])


(defrule action-fold3 (:backward :certainty 0.75 :importance 248)
  if  [and [hand-score poxpert ?score] (< ?score .50)
      [cost-of-calling poxpert expensive]]
      then [action poxpert fold])

(defrule action-fold4 (:backward :certainty .90 :importance 247)
  if [strategy poxpert prevent-loss]
  then [action poxpert fold])

;; rules for calling (or checking when amount to call is zero)

(defrule action-call1 (:backward :certainty 0.90 :importance 239)
  if [and [hand-score poxpert ?score] (and (> ?score .3) (< ?score .7))
     [num-cards-on-the-table table 0]]
  then [action poxpert call])

(defrule action-call2 (:backward :certainty .85 :importance 238)
  if [and [hand-score poxpert ?score] (and (> ?score .5) (< ?score .7))
     [position-on-the-table poxpert bad]]
     then [action poxpert call])

(defrule action-call3 (:backward :certainty .9 :importance 237)
  if [strategy poxpert slow-playing]
  then [action poxpert call])

(defrule action-call4 (:backward :certainty .7 :importance 236)
  if [strategy poxpert normal-play]
  then [action poxpert call])

(defrule action-call5 (:backward :certainty .75 :importance 235)
  if [strategy poxpert waiting-for-draws]
  then [action poxpert call])

;; rules for raising
(defrule action-raise1 (:backward :certainty .90 :importance 225)
  if [and [hand-score poxpert ?score](> ?score .7) (< ?score .9)
     [amount-to-call poxpert ?coc] [big-blind table ?bb]
     (> ?coc (* 2 ?bb))]
  then [action poxpert raise])

(defrule action-raise2 (:backward :certainty .55 :importance 224)
  if [strategy poxpert bluffing]
  then [action poxpert raise])

(defrule action-raise3 (:backward :certainty .85 :importance 223)
  if [strategy poxpert pocket-pair]
  then [action poxpert raise])

;; rules for determining the strategies based on hand's score
;; position , table's current stage (pre-flop, flop, turn or river)

;; slow-playing can be very beneficial before the river and when
;; the player is almost certain to win the pot (hand score is very high)
;; in this case player does not raise to scare anyone out of the pot
;; instead, she would only call, but on the river she would jam the pot.

(defrule slow-player (:backward :certainty .80 :importance 195)
  if [and [hand-score poxpert ?score]  (> ?score .85)
     [num-cards-on-the-table table ?num-cards] (< ?num-cards 5)]
  then [strategy poxpert slow-playing])

;; certain hand-scores together with the current table situation
;; would be good for drawing, in which case player calls the bets.


(defrule waiting-for-draws (:backward :certainty .75 :importance 193)
  if [and [hand-score poxpert ?score] (> ?score .3) (< ?score .6)
     [num-cards-on-the-table table ?num-cards] (eq ?num-cards '3)]
     then [strategy poxpert waiting-for-draw])
;; if the program has a poor to average hand but the cost of calling is low
;; the program may choose the bluffing strategy
(defrule bluffer (:backward :certainty .65 :importance 197)
  if [and [hand-score poxpert ?score] (and (> ?score .1) (< ?score .4))
     [cost-of-calling poxpert cheap]]
  then [strategy poxpert bluffing])

(defrule normal-hands (:backward :certainty .75 :importance 192)
  if [and [hand-score poxpert ?score] (and (> ?score .2) (< ?score .7))]
  then [strategy poxpert normal-play])

(defrule presenting-pocket-pair (:backward :Certainty .85 :importance 191)
  if [and [hand-score poxpert ?score] (> ?score .7)]
  then [strategy poxpert pocket-pair])

;; when hand-score is low
(defrule prevent-losses (:backward :certainty .90 :importance 196)
  if [and [hand-score poxpert ?score] (< ?score .2)]
  then [strategy poxpert prevent-loss])
;; these rules determine that how much the program has to call
;; if it's the first person to call, so after the big blinds on pre-flop
;; and after the button on the other turns.

(defrule how-much-to-call1 (:backward :certainty 1.0 :importance 190)
  if [and [num-cards-on-the-table table 0] [seat poxpert 3] [big-blind table ?bb]]
  then [amount-to-call poxpert ?bb])

(defrule how-much-to-call2 (:backward :certainty 1.0 :importance 189)
  if [and [num-cards-on-the-table table ?tr] (> ?tr 1) [seat poxpert 2]]
  then [amount-to-call poxpert 0])
;; these rules determine if the player has an edge
;; based on her position on the table

(defrule position-value1 (:backward :certainty 1.0 :importance 185)
  if [and [seat poxpert ?seat] (> ?seat 1) (< ?seat 4)]
  then [position-on-the-table poxpert bad])

(defrule position-value2 (:backward :certainty 1.0 :importance 184)
  if [and [seat poxpert ?seat] (eq ?seat 1) (> ?seat 4)]
  then [position-on-the-table poxpert good])
;; these rules determine how much the player should raise if the
;; selected action was raising, the outcome of this rules can be
;; looked up in the database.

(defrule amount-of-raising1 (:forward :certainty 1.0 :importance 180)
  if [and [action poxpert raise][strategy poxpert bluffing]]
  then [amount-of-raise poxpert 5BB])

(defrule amount-of-raising2 (:forward :certainty 1.0 :importance 179)
  if [and [action poxpert raise][strategy poxpert slow-playing]]
  then [amount-of-raise poxpert All-in])

(defrule amount-of-raising3 (:forward :certainty 1.0 :importance 178)
  if [and [action poxpert raise][strategy poxpert normal-play]]
  then [amount-of-raise poxpert 2BB])
;; for instance, if the program has chosen a bluffing strategy
;; it should raise at least 5 times the big blinds to scare the others
;; away and win the pot
(defrule amount-of-raising4 (:forward :certainty 1.0 :importance 177)
  if [and [action poxpert raise][strategy poxpert bluffing]]
  then [amount-of-raise poxpert 5BB])

(defrule amount-of-raising5 (:forward :certainty 1.0 :importance 176)
  if [and [action poxpert raise] [strategy poxpert pocket-pair]]
  then [amount-of-raise poxpert 10BB])

;; these rules determine if the amount of calling is relatively high
;; or low, there are three important factors, pot size (total amount of bets
;; on the table), program's chip count and the amount of big blinds (minimum bets)
(defrule cost-of-calling1 (:backward :certainty 1.0 :importance 170)
  if [and [amount-to-call poxpert ?cost] [pot-size table ?pot] [big-blind table
?bb]
     [chip-count poxpert ?money] (> ?cost (max (* 2 ?pot) (* 10 ?bb) (* .3
?money)))]
  then [cost-of-calling poxpert expensive])

(defrule cost-of-calling2 (:backward :certainty 1.0 :importance 170)
  if [and [amount-to-call poxpert ?cost] [pot-size table ?pot] [big-blind table
?bb]
     [chip-count poxpert ?money] (and (>= ?cost (max (* .5 ?pot) (* 5 ?bb) (* .1
?money)))
                                 (<= ?cost (max (* 2 ?pot)  (* 10 ?bb) (* .3
?money))))]
  then [cost-of-calling poxpert moderate])

(defrule cost-of-calling3 (:backward :certainty 1.0 :importance 170)
  if [and [amount-to-call poxpert ?cost] [pot-size table ?pot] [big-blind table
?bb]
     [chip-count poxpert ?money] (<= ?cost (max  (* .5 ?pot) (* 5 ?bb) (* .1
?money)))]
     then [cost-of-calling poxpert cheap])
```