B351 AI Project: Texas Hold 'em

Steven Myers and Samuel Eleftheri

Indiana University, Bloomington, IN, USA

May 5, 2017

Our final project was to build a working Texas Hold'em Poker emulator, and an AI agent to play the game efficiently. Our approach was to accurately emulate poker as it is played professionally with realistic betting systems. We built an agent that can effectively determine the best move given its cards and river by evaluating the strength of its hand by comparing its relative strength to all other possibile hands. The agent is able to rationally make moves in a poker game and take steps to win, even against a human player.

1 Introduction

Creating an agent for a game needs to be a well thought-out procedure. "Agent-based AI is about producing autonomous characters that take in information from the game data, determine what actions to take based on the information, and carry out those actions." (Hawthorn, Weber, and Scholten, 2001). This reference can make the process seem simple, but can be (and will be) incredibly complex. Simpler games will contain a initial state, a set of possible actions, the repercussions of the actions, and the goal state. Understanding this information is crucial to creating an intelligent agent for a game. We must be prepared for any unknown information if we are dealing with an unknown environment. After dealing with the information, we need to recognize what best action would reach the goal the quickest.

We can use chess as an example. In early development of intelligent agents, Chess was often used as a game to anaylze the performance of agents. Chess was a popular choice since the environment is known. Players cannot decieve each other since all the information is on the board. Since chess has a large amount of possible actions, early agents needed a effective search algorithm to find the best move. The agent must be able to simulate a numbers of possible moves ahead in order to react to the other player's actions.

Implementing agents for games with unknown environments presents a new challenge. Predicting which action to perform with insufficient information requires a more observant agent. Poker is a good model to use when having a game with unknown data. "...the Poker's gamestate is hidden because each player can only see its cards or the community cards and, therefore, it's much more difficult to analyze. Poker is also stochastic game i.e. it admits the element of chance." (Teófilo, 2)

2 Playing Texas Hold 'em as an AI Problem

Since Texas Hold'em has unknown information, it is important to develop an attentive agent that learns from previous rounds and opponent's actions. An agent has to consider the significance of its information and understand how strong of an impact it can make. Some of our information includes:

1. Known Information:
    - Player's Cards
    - Current River Cards
    - Deck
    - Pot Size
    - Opponent's Money
2. Unknown Information:
    - Opponent's Cards
    - Future River Cards
    - Bluffing Probability

With the known information of our player's cards and the current river cards, an agent can figure out how valuable our hand is. This is an essential procedure as it will greatly influence what action we execute. A higher value hand can result in the action of betting, where as a lower value hand can result in folding, especially

when challenged by another player. Agents can then enhance the choice making algorithm by including unknown information. Agents can perform certain calculations to estimate the chances that certain cards can be drawn, or the probability of opponents having a certain card.

Other players' actions will greatly influence an agent's decision algorithm. Having an agent that can understand an opponent's actions can be the difference between winning or losing money.

3 Code

Classes Our Texas Hold'em python file consists of two major classes: *Table* and *Player*. The Table class handles distributing cards to the players, keeping track of player actions, collecting bets/ante's from every player, and finally determining who wins a round of poker. The Player class represents a poker player and has two cards. Its methods are invoked by the Table class during poker rounds to determine what action a player would like to take based on its current hand. The Player class includes basic decision making to determine poker hands are strictly better than other hands.

Main Poker Loop A majority of the code written for this project is dedicating to poker hand detection and the main poker game loop. The code was written in such a way to accurately represent poker rules as they are in the real world. Poker rounds are split up into open and closed pots. An initial round of betting goes around and each player is able to make a bet. This is called *opening* the pot. If the pot was opened, then turns are taken again to raise the current bet or call the current bet. This continues until every player calls or folds. The poker game loop is written in the Table class, under the *play* method. Below is a bit of pseudocode that outlines the main game loop, particularly the OPEN/ClOSED pot.

```
while river cards <= 5:
 # Open the pot
 bet = infinity
 for player in players:
  action = Player.act()
  if action == BET:
   biggest_bet = min(bet, action.amount)
  elif action == FOLD:
   remove(player)
 # Close the pot
 if bet != infinity:
  while not all players fold or call:
   raise = infinity
   for player in players:
    action = Player.act()
    if action == RAISE:
     raise = min(raise, action.amount)
    elif action == FOLD:
     remove(player)
   bet = raise
```

Card Represenations and Determining Poker Hands Cards are represented as the integers $0-51$. A card value can be calculated by taking the card ID and modding by 13. Similarly, the suite of a card can be found by dividing by 13. Determining poker hands was relatively straight forward. I wrote a function called *poker_hands(cards)* that accepts a list of card integer values. I constructed a frequency count of the four suites and the 13 different possible face values by calculating the face and suite values for each input card. From there, the frequency of a face values corresponds directly to a N-of-a-kind, where N is a 2, 3, or 4. Similarly, if any suite count was greater than 4, then the cards formed a flush. In order to determine a straight, I iterated through the frequency table of face counts 5 cards at a time. A snippet of the code to generate frequency tables is below.

```
face_count = [[] for x in range(0, 13)]
suite_count = [[] for x in range(0, 4)]
for c in cards:
  value, suite = (c % 13), (c // 13)
  face_count[value].append(c)
  suite_count[suite].append(c)
```

Ranking Poker Hands and the Best Hand Once I determined what poker hands were held given a list of cards, I was able to iterate through all of the possible poker hands and determing the *best* poker hand. I was able to do this because the different tiers of poker hands are strictly better or worse than another. So, a two-of-a-kind is always worse than a three-of-a-kind. If two poker hands were of the same tier, then simply summing the face values of all the cards that make up that hand can serve as a secondary ranking system to determine a better poker hand when they are the same tier. This took a list of poker hands and converted them into tuples of first and secondary rankings. After sorting, the best hand can be taken from the front of a sorted list.

Player Actions The Player serves as an AI agent. Its *act* method is used to return an action, which is one of FOLD, BET, RAISE, CALL, or CHECK. Proper actions are determined by a boolean value called OPEN passed to the act method. This lets the agent know whether or not the pot is open or closed and if raising or betting is appropriate. The player's act method then identifies its best hand so far with its privately held hands and the public river cards. If the best hand held by the player is better than some threshold, like a high card less than 7, then the player will not fold. Moreover, if the best hand held by the player is better than some percentile ranking of all hands, the player will bet chips.

4 Results

The results of the project was not impressive, as much of the game's information was not used appropriately. The agent lacked the core ability to identify and apply data into its decision making algorithm. It's intelligence for which action to performed was horridly implemented, due to my (Samuel Eleftheri) lack of preparations. Things that I would have done differently would be to start working on other information into the agent's decision making ability. I became too focus on trying to implement Bayes Theorem for future card probability, that it wasted time.

The things that went right was establishing an environment to operate our agent's performance. No problems were detected with recognizing hand ranking, performing the set of available actions (CALL, FOLD, CHECK, RAISE), comparing players' hands, and bringing multiple players into the poker game. The Player's chips and cards were efficiently recorded and used without error. The Table class ran the poker game correctly, with the betting system and kicking of players implemented.

BIBLIOGRAPHY

Ian Millington, John Funge. *Artificial Intelligence for Games 2nd Edition.* CRC Press Taylor & Francis Group, 2009. Printed

Luís Filipe Teófilo, Luís Paulo Reis, Henrique Lopes Cardoso (2013). "Estimating the Odds for Texas Hold'em Poker Agents". In: *Intelligent Agent Technologies (IAT)* Volume 02, pp. 369-374.
URL: http://dl.acm.org/citation.cfm?id=2569335