B351 AI Project: Texas Hold 'em

Yining Wang (Lynn) and Xing Wei (Joe)

Indiana University, Bloomington, IN, USA, Earth

May 1, 2017

Royal Flush	10♦,J♦,Q♦,K♦,A♦
Four of a kind	7♥,7♦,7♠,7♣,3♥
Full house	4♥,4♦,5♣,5♦,5♥
	(Above are some example of cards)

This project is about the strategy card game Texas Hold'em, a very classic and popular game around the world. There is going to be n players participating in this game, and each player is a different computer AI programmed by other student groups. The AI with the most in-depth decision making ability and as well winning the game will be the final champion of this project.

1 Introduction

In video games, artificial intelligence is used to generate intelligent behaviors primarily in non-player characters (NPCs), often simulating human-like intelligence. The techniques used typically draw upon existing methods from the field of artificial intelligence (AI). However, the term game AI is often used to refer to a broad set of algorithms that also include techniques from control theory, robotics, computer graphics and computer science in general. Video game AI has come a long way in the sense that it has revolutionized the way humans interact with all forms of technology. This has become more versatile in the way we use all technological devices for more than their intended purpose because the AI allows the technology to operate in multiple ways, developing their own personalities and carrying out complex instructions of the user.

Texas hold 'em or Texas holdem is the most popular of community card poker games and is the most prevalent poker variant that is played in casinos in the United States. Its no limit form is used in the main event of the World Series of Poker (WSP). It also happens to be the main game in the World Poker Tour (WPT) that hosts international poker tournaments around the globe. Here is a brief glimpse into the history of the game.

While there is no clear evidence as to where the game emerged from, it is sure that the game originated in Texas. The legend says that the earliest version of the game was played by Robstown, Texas in the early 1900s. The game is said to have first come to Dallas, Texas, in 1925 and the word 'Poker' of which it is a popular version, comes from the German word 'pochen', which means, "to knock".

The game is then said to have been introduced to Las Vegas by a few Texan gamblers and card players. This included the names of Crandell Addington, Doyle Brunson and Amarillo Slim. Soon later, Crandell Addington also wrote about the history of no-limit Texas hold'em for Doyle Brunson's Super System 2. The game gained popularity in 1970, and in the same year the Horseshoe Hotel and Casino was opened in Las Vegas by Benny Binion. Not just the popularity, but also the manner in which the game can be played has evolved too. With modern technological innovations, today it is possible to play Texas Holdem directly on the Internet or even on a cell phone.

The game gained popularity as the main event in poker tournaments, which started with Binion accepting a small invitational Poker tournament from Riverside Casino's Tom Morehead. Binion focused on rapidly increasing antes and blinds and other methods so that a winner could be produced in a short span of time. In 1972, 8 players took part in the World Series of Poker tournament; and thirty years later, the figure shot up to 800. Popularity statistics show that over the past five years, the popularity of televised tournaments has grown, in turn meaning that Texas Holdem has grown in popularity too, becoming a part of mainstream television.

2 Playing Texas Hold 'em as an AI Problem

When designing any agent-based system, it is important to determine how sophisticated the agents' reasoning will be. Reactive agents simply retrieve preset behaviors similar to reflexes without maintaining any internal state. On the other hand, deliberative agents behave more like they are thinking, by searching through a space of behaviors, maintaining internal state, and predicting the effects of actions. Although the line between reactive and deliberative agents can be somewhat blurry, an agent with no internal state is certainly reactive, and one which bases its actions on the predicted actions of other agents is deliberative. Here we describe one system at each extreme as well as two others that mix reactive and deliberative reasoning.

The agent's goals are only implicitly represented by the rules, and it is hard to ensure the desired behavior. Each and every situation must be considered in advance. For example, a situation in which a helicopter is to follow another helicopter can be realized by corresponding rules.

But if the programmer fails to foresee all possible events, he may forget an additional rule designed to stop the pursuit if the leading helicopter crashes. Reactive system in more complex environments often contain hundreds of rules, which makes it very costly to encode these systems and keep track of their behavior.

The nice thing about reactive agents is their ability to react very fast. But their reactive nature deprives them of the possibility of longer-term reasoning. The agent is doomed if a mere sequence of actions can cause a desired effect and one of the actions is different from what would normally be executed in the corresponding situation.

Deliberative agents constitute a fundamentally different approach. The goals and a world model containing information about the application requirements and consequences of actions are represented explicitly. An internal refinement-based planning system uses the world model's information to build a plan that achieves the agent's goals. Planning systems are often identified with the agents themselves.

The basic planning problem is given by an initial world description, a partial description of the goal world, and a set of actions/operators that map a partial world description to another partial world description. A solution is a sequence of actions leading from the initial world description to the goal world description and is called a plan. The problem can be enriched by including further aspects, like temporal or uncertainty issues, or by requiring the optimization of certain properties1.

Deliberative agents have no problem attaining longer-term goals. Also,the encoding of all the special rules can be dispensed with because the planning system can establish goal-directed action plans on its own. When an agent is called to execute its next action, it applies an internal planning system.

3 Code

Minimax is a decision rule used in game theory for minimizing the possible loss for a worst case scenario. It is usually being considered for 2-player games such as tic-tac-toe and chess. However, there are two main tree search algorithms that are widely being used. One of them is AlphaBeta search algorithm. This is also called the Alpha-Beta pruning algorithm, for it is used the simple tree search to minimize opponent's possible moves in a game like chess. It seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games. It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision. Below is the pseudocode for Alpha-Beta Pruning:

```
01 function alphabeta(node, depth, α, β, maximizingPlayer)
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node
04     if maximizingPlayer
05         v := -∞
06         for each child of node
07             v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
08             α := max(α, v)
09             if β ≤ α
10                 break (* β cut-off *)
11         return v
12     else
13         v := +∞
14         for each child of node
15             v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
16             β := min(β, v)
17             if β ≤ α
18                 break (* α cut-off *)
19         return v
```

Figure 1: *Pseudocode for Alpha-Beta Pruning Algorithm*

On the other hand, for more than two players, a new type of algorithm callled Monte Carlo Tree Search is more suitable. It is a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. Their essential idea is using randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three distinct problem classes: optimization, numerical integration, and generating draws from a probability distribution. In computer science, Monte Carlo tree search (MCTS) is a heuristic search algorithm for some kinds of decision processes, most notably those employed in game play. Two leading examples of Monte Carlo tree search

are the computer game Total War: Rome II's implementation in their high level campaign AI and recent computer Go programs, but it also has been used in other board games, as well as real-time video games and non-deterministic games such as poker. Below is the formula for Monte Carlo Tree Search:

$$\text{UCB1} = \overline{X}_j + C\sqrt{\frac{2\ln n}{n_j}}$$

- X_j is estimated reward of choice j
- n is number of times parent has been tried
- n_j is number of times choice j has been tried

Figure 2: *Formula for Monte Carlo Tree Search*

The focus of Monte Carlo tree search is on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space. The application of Monte Carlo tree search in games is based on many playouts. In each playout, the game is played out to the very end by selecting moves at random. The final game result of each playout is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future playouts.

Each round of Monte Carlo tree search consists of four steps:

1. Selection: start from root R and select successive child nodes down to a leaf node L. The section below says more about a way of choosing child nodes that lets the game tree expand towards most promising moves, which is the essence of Monte Carlo tree search.
2. Expansion: unless L ends the game with a win/loss for either player, either create one or more child nodes or choose node C from them.
3. Simulation: play a random playout from node C. This step is sometimes also called playout or rollout.
4. Backpropagation: use the result of the playout to update information in the nodes on the path from C to R.

In our case, it seems the only choice for us is to use the monte carlo tree search with the given possibilities for each different hand results. Since for the game Texas Hold'em, the most ovbious problem is that there are more than two players being involved in the game, and the decision making also need to include the bitting for each players and the chances of forfeiting. Computations show how the frequencies for 5-card poker hands were determined. To understand these derivations, we should be familiar with the basic properties of the binomial coefficients and their interpretation as the number of ways of choosing elements from a given set. below is a table for the rank table and probabilities for Texas Hold'em:

Table 1: *Rank Table and Probability*

Name		
Hand	Example	Probability
Royal Flush	10♥,J♥,Q♥,K♥,A♥	$P(1/649740)$
Straight Flush	3♣,4♣,5♣,6♣,7♣	$P(3/216580)$
Four-of-a-kind	5♥,5♣,5♦,5♠,Q♥	$P(1/4165)$
Full House	2♥,2♦,Q♠,Q♣,Q♥	$P(6/4165)$
Flush	3♦,6♦,7♦,9♦,J♦	$P(1277/649740)$
Straight	3♥,4♦,5♥,6♠,7♣	$P(5/1274)$
Three-of-a-kind	A♥,A♣,A♠,4♦,2♣	$P(88/4165)$
Two Pairs	6♦,8♣,8♥,K♠,K♣	$P(198/4165)$
Pair	10♠,10♥,3♥,K♠,8♦	$P(352/833)$
High Card	3♥,5♥,8♦,Q♠,A♣	$P(1277/2548)$

As a result, our code is going to be much more complex than any other simple games like tic-tac-toe. Here is our final code for achieving the functionality of the Texas Hold'em poker game:

$$\text{UCB1} = \overline{X}_j + C\sqrt{\frac{2\ln n}{n_j}}$$

- X_j is estimated reward of choice j
- n is number of times parent has been tried
- n_j is number of times choice j has been tried

Figure 3: *Actual code for Texas Hold'em poker game*

4 Results

Discuss the result of your project – what went right and what went wrong. For the former, what would you do to make it even better? For the latter, what would you do differently?