CAP-4630 FALL 2017

Final Project Writeup

Michael Weiland, 5164092

My Kuhn Poker Agent

Kuhn poker is a zero-sum game, so an ideal agent would maximize the number of chips it possesses and minimize the loss from losing bets. I wrote my agent strategy in python script using this example python project, which was provided as a starting point: **https://github.com/kdub0/kuhn3p**

The agent is capable of connecting to a ACPC dealer server where other agents can compete with it as long as they are using the same protocol.

My Approach

The rules for 3 player Kuhn poker are very simple, with few usable sources of information that could be used for the minimax algorithm. To maximize my agent strategy's score, the useful information I used include the card it is dealt and a memory of the opponents' bets/bluffs.

The agent strategy I wrote is very straightforward: The frequency of the opponents' bluffs is considered when predicting the chance of winning a bet.

First, a heuristic function uses the card rank, that of which the agent is holding, to estimate a cut-and-dry chance of winning based on the single card alone. For example, an Ace means that the round is won without a doubt, since that is the best hand that could be dealt in Kuhn poker. Depending on the game state, an opponent who has placed a bet can influence the agent's prediction of winning, for better or worse. If the opponent is known to bluff very frequently, then their bet may be considered an indicator that they may lose the showdown, increasing the predicted chance of winning for our agent and vice versa.

The Method

Using Bayes' theorem, the outcomes of both opponents' bets and bluffs are stored as a probability. As there are two opponents, there are two corresponding probabilities which are stored to represent each of their tendencies. When the dealt hand has reached the end state, the results of the opponent bluffs and bets are weighed in with their probabilities that had been derived from past rounds.

When Special.py connects to the dealer, it stores a randomly generated number in the range of [0,1). The reason why I included the randomly generated number was that I was attempting to make the agent less predictable if it plays the same opponent more than once, if that opponent were using a more sophisticated learning algorithm than Special.py.

The Results

What I observed from running trials with my agent strategy, the Special.py player, was that it never faced a loss when challenged with the example players included with the python project (linked at the top of the paper). Occasionally, it did break even, but never went to a negative score when playing against the Bluffer.py player and the Chump.py player.

The variability introduced in the Special.py player's initialization had a non-negligible effect in the magnitude of winning bets. When the initial RNG was <0.5, the threshold for the agent strategy to predict a winning hand was a lot lower, so it was more aggressive in playing hands when the RNG value was lower. On the other hand, >0.5 RNG values used against the example agents made the Special.py player a lot more cautious with their bets, so scores were generally much lower, with the disparity between our score and the opponents scores being much smaller in these cases too.

When playing three separate instances of Special.py pitted against each other, the scores for the first and third Special.py player were usually negative, with the second Special.py player being the one who came out on top every time.

I have not investigated this behavior very much for this project, but I assume it is because the first player is disadvantaged by the second player predicting whether they have an advantageous hand over them. The third player may be coming out with a minimized score because it antes chips but is too cautious to play its hand against player 2, which does not appear to bluff when it makes confident bets based on its prediction from player 1.

Retrospect

Initially, I was planning on having some variability in the strategy so that it would not be so predictable (beyond the included RNG in initialization). For example, I had considered introducing a bluff to the agent strategy. I did not finish implementing it, but essentially, risky bets that could have resulted in a loss would be taken in certain situations to make the behavior less predictable.

I had hesitated to implement the bluffing ability from the start because that would only be useful when competing against relatively smarter AI than the simple example strategies included with the sample project from Github. Trying to implement "mind games" like bluffing on risky hands when playing against Bluffer.py or Chump.py would not affect their behavior in any way, so those risky bets would be taken with no benefit at all, which would compromise the optimization of Special.py agent's method of min-max'ing.

Additionally, I would have liked to record the outcome of the trials more formally, so I could possibly report them with some analytics and graphics, as opposed to just speaking about the Special.py player's performance anecdotally. Eventually, I ran down the timer and did not get around to it. However, I am still interested in seeing how the agent strategy performs against other players, maybe even a human player, in the future. I'd like to enter the annual computer poker competition next year, as a hobbyist, with a more sophisticated agent to see how it fairs.