Caleb Reath

3514637

AI Project

November 29, 2019

Dr. M. Flemming

Approaches:

When I began the project my first thoughts were to become “good” at the game so I can identify potential useful indicators to help when I start building the AI.

I noticed that the initial 5 turns in any game fall into two categories. Adding a card to a “New Hand” or an “Existing”

When I started programming, I first attempted to create a card relationships table. This proved to be a hassle to maintain and use since the relationship between any 2 cards is only useful for 3 of the 9 types of hands. The consequence of this is that my AI began favoring exclusively pairs and two pair hands. Averaging a score of 24.

My next approach was to calculate all the probabilities for each existing hand on the board and compare it to the probabilities of each hand with the addition of the next card. This proved to be very successful averaging on its own 45.133 points

Programming

Programming the methods to quickly calculate probabilities was extremely difficult since Combinatorics with cards becomes very complicated when the deck begins to empty.

To help decide which hands I should calculate, I built a simple function that checks if a hand type is possible given any hand. This function allows replacement and does not count how many of a card is remaining or missing entirely. It simply tells if given infinite cards, the hand type selected is either possible or impossible.

After probabilities are calculated they are very useless, pairs make up 40% of most hands where as royal and straight flushes are both < 0.004% chance of occurring. For this reason, I used the probabilities to calculate utilities in 2 dimensions. Row utilities and column utilities.

Utilities

To calculate the Expected utilities, I initially used the British scoring system as my U(0). The expected utilities generated can be combined into a 5x5 grid giving the Expected Utilities of each empty space on the game board.

My utility array is in the main class Reath_GreedyProbabilityMCPlayer.
Using the British system as my U(0) my AI with MC disabled was able to average 30 points.
$U(i+1) = \text{maxScore}_{(i=1 \text{ to } 80)}(U(i) * 40/j)$

Since I was unsure of the range the utilities could be in I used 40/j to slowly go from 40*U(0) to 0.5*U(0). This gave me very large values for "good" hands and low ones for pairs and two pairs for example.
Pair utility = 1.94033
Royal Flush Utility = 34285.7142

Iteration results:
AverageScore U(0) = 30
AverageScore U(1) = 39
AverageScore U(2) = 41
AverageScore U(3) = 43
AverageScore U(4) = 45

After iteration 4 the utilities generated had diminishing returns with a max at 45.12

Once the ExpectedUtilities are generated I do one of the following

- If MonteCarlo is disabled return the max Utility play
- If there are less than 18 utilities that are at least 1.0 pass all plays above 1.0 utility to the GreedyMonteCarloPlayer.
- If there are less than 2 plays above 1.0 play the max utility play

Speed

The AI runs with no MonteCarlo in less than 5ms. This leaves plenty of room to allow MC to take its time whenever needed. Meaning that when MC is enabled, we can use a much deeper depth limit since time spent on each space is on average 5-10 times longer.

Performance

The performance of the AI is very good, compared to my previous attempts my AI can defeat the GreedyMCPlayer more than half the time. This is possible since my AI uses a modified version of the MC algorithm. The modified MC algorithm is informed of the best immediate moves that can be made. These moves are then iterated over at depth 12 as much as possible. A bonus to sometimes skipping MC iterations using probabilities is that the AI can finish a game anywhere from half a second to the whole 30 seconds depending on game complexity (generally faster games are higher scoring)

Problems

Problems I have run into with the algorithm is the randomness of the MonteCarlo algorithm. Since they can make bad decisions the scores can vary to very much higher to very low when compared to if it was disabled from the start. However, when enabled The AI averages 2-5 points more than when it is not.

Training for the utility values had to be generated without the MonteCarlo part of the AI since it required thousands of iterations to find the maxUtility for each hand

Finding the optimal Depth Limit is a very slow process as well. Luckily since my AI can make plays based off probabilities on its own, we can easily choose times where it is more effective to use the maxUtility and skip MC all together. This dramatically speeds up the average running time of the algorithm.