James Israelson

Dr. Michael Fleming

CS4725

Dec 08, 2021

Programming Project: Poker Squares Player

For my Poker Squares player, I decided to improve upon the RandomMCPlayer that was provided. Instead of evaluating the average score achieved in a partially filled grid, my player uses a custom-trained evaluation function to assess the potential of partially filled grids. The code for the Monte Carlo simulation is essentially unchanged from the RandomMCPlayer, but all the code for the evaluation function has been written from scratch. I have given credit in the comments of my code where necessary.

Each partial hand in the grid is abstracted to a String. The encoding has the form, 1:pfsho(2), which represents the following information:

* The move number.
* The number of pairs the hand contains.
* Whether a flush is possible or achieved.
* Whether a straight is possible or achieved.
* Whether a full house is possible or achieved.
* Whether a four of a kind is possible or achieved.
* The number of cards in the hand that have no pair.

The move number is separated from the rest of the encoding by a colon. For the letter encodings, a lowercase letter denotes that the hand type is possible, while an uppercase letter denotes that the hand type has been achieved. Pairs are the only exception, where a lowercase ‘p’ denotes a single pair has been achieved, while an uppercase ‘P’ denotes that two pairs have been achieved. The number of cards in the hand that have no pair are separated from the rest of the encoding by parentheses.

The values for each encoding were calculated by JIsraelsonTrainer. The Trainer takes a number of minutes as an argument and runs simulations of games until the timer is up. The actual game simulation is the same as the RandomPlayer with a few exceptions. As the games are randomly simulated, each hand in the game is encoded and saved as the key in a HashMap. The score of the game and the number of times the hand has been encountered are stored as the value (see the private class, Heuristic). When the timer is up, each encoding and its final score are saved into a HashMap and exported to a file specified in JIsraelsonPlayer.java. After 9 hours of training, the program generated 629 unique hand encodings and their average scores.

When JIsraelsonPlayer is executed, it loads a specified file into a HashMap. As mentioned before, the actual function of the Monte Carlo search is unchanged from RandomMCPlayer. However, instead of using the average score already obtained by the partial hands in the grid, JIsraelsonPlayer encodes the partial hands into Strings, and searches the HashMap for the values of those encodings. This is how the potential value of each partial hand is determined. Hands that consistently scored higher in the random simulation by JIsraelsonTrainer are given higher scores, and thus, more likely to be picked by the getPay() method.

After some experimentation, I determined that the best max depth for the Monte Carlo search was depth 3. It was hard to decide as most depths performed similarly. I chose depth 3 as it tended to have a lower standard deviation. In 20 games on tournament seed 420L, the JIsraelsonPlayer scored a mean of 44.75, a standard deviation of 10.18, a minimum score of 25, and a maximum score of 66. This is about 10 points higher than the mean of the RandomMCPlayer, and over 5 times better than the mean of the RandomPlayer.

If I had more time, I probably would have implemented some type of card tracking into the evaluation function. My player suffers from lack of information about the specific cards that were played. Even adding the number of single cards with no pair increased the average score of my player significantly. I think the extra context would be beneficial.

I would also consider upgrading the Trainer to focus less on common hand types. Because of the nature of random simulation, rare hands are encountered less often. This leads to inaccurate average scores for what should be highly valued hands. The encoding “5:FS(5)” is a good example of this. After 9 hours of training, my function has it rated as the 21st best hand encoding. However, getting a straight flush on your 5th move is actually the best hand in the game. These issues could be solved by more training or a better simulation function.

In the end, I am quite satisfied with my Poker Squares player. I chose Monte Carlo search because the idea of creating a useful evaluation function seemed like a fun challenge. I was unsure how well it would work, but I was confident I could create a player that would outperform the RandomMCPlayer. I was surprised at how well my player performed. After a few days of experimentation, my Player consistently beat the RandomMCPlayer by an average of 5 points. After some tweaks, I was able to push that average even higher. I really enjoyed taking this course and working on this project. I have a new appreciation for the subject of AI.