AI Project Report

Poker AI

Nathan Rose, Johnathan "JP" Prideaux

Version: 1.0

April, 2022

Revision History

Date	Version	Description	Author
4/19/2022	0.1	Created basic formatting	JP
4/19/2022	0.2	Introduction	JP
4/20/2022	0.3	Related Works	JP
4/20/2022	0.4	Source Code Description	JP
4/20/2022	0.5	Initial Software Design	JP
4/20/2022	0.6	Initial Resources Used	JP
4/20/2022	0.7	Initial Data/Problem Analysis	JP
11/29/2021	0.10	Formatting Pass 1	Nathan Rose
12/01/2021	1.0	Final Formatting	Nathan Rose

Contents

1 Introduction

This project has set out to create an AI that is able to play Texas Hold'em Poker. The goal of the AI is to make as much money as possible while playing. For this project we built a Texas Hold'em Poker game in Python and trained several AI's to play the game. We rewarded them based on how much money they earned. The more the better!

Texas Hold'em Poker can be played by 2 to 9 player each with a bank of starting money. The initial setup before any play occurs is as follows: First a "dealer" button is placed in front of a player (can be one of the players physically next to the dealer), a "big blind" button is placed in front of the player to the left of the "dealer" button player, then to the "big blind" players left is the "small blind" player which receives a "small blind" button. These buttons are just tokens showing the previously mentioned words on them. They rotate clockwise after each hand is completed. Once these buttons are placed the big blind and small blind players pay their blinds and place the chips in front of them. Blinds are a set amount of money of x and 2x value. For example small blind of $5 and a big blind of $10. Next the dealer deals each player two cards, one at a time starting with the player with the "big blind" button in front of them.

Once the cards are dealt the first main round begins. This round consists of players initial decisions. Each player has 5 main choices when making a decision. One they can fold their hand and get rid of their cards. They are no longer in the hand. Two they can check if there is no bet they are required to match. Three they can bet an amount of money that is larger than the minimum bet and smaller or equal to what they have in their bank, if no other player has bet before them. Four, if another player has bet before them they can call the bet and match the bet with money from their bank. Five, they can raise the bet and add more money to the previous players bet. Once all the players have made their decisions and all decisions have been resolved and we reach the last player than needs to act the round is over.

Starting the second round is the dealer who burns a card then deals three of the five community cards. This is referred to as the "flop". Once the three cards are shown another round of decisions are made by the players. Once all the decisions are resolved the round is over.

Starting the third round is the dealer who burns a card then deals one more community card. This is referred to as the "turn". Once the new community card is shown another round of decisions is made by the players. Once all the decisions are resolved the round is over.

Starting the fourth and final round is the dealer who burns a card then deals one more community card. This is referred to as the "river". Once the new community card is shown a final round of decisions is made by the players. Once all the decisions are resolved a winner is determined. The winner is determined by the player with the best 5 card hand. The hand order is shown in INSERT HAND VALUE FIGURE. If two players have the same level of hand the one with the highest card value wins. For example if two players have straights, one player has 3,4,5,6,7 and the second player has 4,5,6,7,8 the second player would win.

Some notes, play can end sooner than the fourth round if all but one player folds. If a player wants to call or raise using all their chips its called going "all in". If a player who is all in loses and has no more money in their bank they are out of the game.

The AI in this case will have all the information about the current state of the game and make one of the previously discussed decisions. After a number of hands the AI will be judged and then the best performers are chosen for the next generation. This process will repeat until a set stopping criteria.

2 Related Work

Ever since computers have been around have we been pitting them against humans in all manners of games. Some famous examples are chess AI and GO AI that have defeated the worlds best humans at these games[1][2][3]. Poker is just another game waiting to be conquered by computers. However it presents a different type of game compared to chess and GO. The later games are perfect information games meaning that both (all) players can see the complete state of the game/board. Poker however is an imperfect information game meaning that one player can not have/see the complete state of the game/board[1][4][2][5][3]. This forces the AI to have to guess about what action to take since they do not have all the information. This has been the trickiest part of making an unbeatable poker AI. This task also applies to many real world applications that also deal with imperfect

information.
In order to solve this problem many different approaches have been taken. "Libratus: The Superhuman AI for No-Limit Poker" has taken a three module approach to the problem[1]. Libratus consists of a pre-computing module, a nested sub-game solver module and a self improvement module. The first module deals with the general strategy of the AI and makes categories of all the possible situations and actions that could be made from that point. It has two kinds of abstractions that it groups these situations into: action abstraction and card abstractions[1]. The card abstractions deal with the possible card combos at each stage of play and lumps groups of card possibilities into buckets that millions of card combos. This is only employed on the last two rounds of play[1]. The use of abstractions gives the AI a general game plan and one that will be refined in the second module. The abstracted game that is created in this module is solved with a distributed version of an improvement over Monte Carlo Counterfactual Regret Min-imization (MCCFR)[1].
The second module examines the game at a finer detailed level to make the best decision it can. Since an imperfect information game cannot be solved on its own the AI second module looks at sub games similar to the one that it is trying to solve and uses that information to make its decision[1]. The third and final module is the self improvement model. It uses the previous data and the outcome of the most recent hand to fine tune its performance. It learns what sub games it should have used and how to arrive at that conclusion in the future[1].
Another approach is taken by "Poki" which is a poker AI that specifically focuses on opponent modeling[4]. Poki's goal is to keep track of how opponents are playing, how it expects them to play and to react to these factors. Poki uses an artificial neural net to keep track of and predict its opponents actions. Poki used feed-forward and back propagation algorithms to tune the model. This model had much better results at predicting opponents actions than previous attempts[4].
Another approach uses "Rhode Island Hold'em Poker" to accomplish the goal of training a poker playing AI. Rhode Island Hold'em Poker is a simplified version of Texas Hold'em Poker than contains less possible states the game can be in making the imperfect information problem easier to solve[2]. The goal for this approach is to train and develop models on a less complex version of the imperfect information problem so that these models can become really good at this version quicker and more efficently before being applied to the more complex problem[2].
From looking into all these different approaches we decided to combine a few and apply a neural net to a slightly simplified version of Texas Hold'em Poker. Where our focus was on the decision making of the AI to make the most money possible.

3 Approach

3.1 Data/Problem Analysis

The data in this problem is poker hands, games and the results of them. We did not use any historical data for this solution. All data was used for the AI learning while it played the game. We valued AI that could win the most money. NATHAN MAYBE SOMETHING HERE ABOUT THE VALUE FUNCTION??? IT COULD GO LATER

3.2 Resources Used

Our programming language of choice is the same as the one from this class, Python 3. We used several libraries and they can be found in the below table Table 1. The most important of these is the Neat library. The Neat library is the one that we use for the AI. The neat library is a pure python implimentation of NEAT. IE theere there is no dependency on Tensorflow or anything(sadly). However it is a very solid library for NEAT, allowing alot of customizaiton from mutation rates and strengths, to starting network topology, activation functions and more!

Table 1: Python Libraries Used

Library name
numpy
itertools
tensorflow.keras
neat
logging
pytest
functools
graphvis
warnings
matplotlib.pyplot
os
copy

3.3 Software Design

Our software design was a bottom up object oriented design. There is two halves to solution. One half is the poker game itself and the second half is the AI that can play the poker game. The first half, the poker game was built bottom up objected oriented starting with the basic building blocks of the poker game. First a card was implemented then a deck. From there the logic of the game was implemented in pieces where each piece added more complexity. Pieces include, players, hands, community cards, board, hand values, rounds and games etc.
Once the game was complete the next step was adding the Neat integration to create the AI player. For the AI the first step was to get a basic set up of the AI, and to do this we used a NEATPlayer which is a player that holds a neat algorithm for playing. This player is used in order to interact the the Board(which as a reminder manages the game).
In main.py and config-feedforward the configuration can be seen for the Neural networks, first we used the ReLU activation to encourage big bets, and we dont care about negative bets(and we wanted a system that can handle non-linear system as poker is 100% not linear). The mutation rates are all set to 0.2, as this was a decent number that allowed for some mutation but still allowing the system to maintain its lead when it develops one. THe system is also designed to start with 30 hidden nodes, these are dense nodes that allow the system to have some compexity starting out to give it a chance to do something, but this model is quickly dropped as can be seen in some of the images. Most of the other values were left as is from the example as they were appropriate for this problem as well.
At the top of the config file you can see the population size as 5000, which is a good critical mass of players to make sure that learning actually occurs. The threshold is just some stupid high number to make sure that the main.py goes through all of the generations, it is irrelvient.
The number of generations is set in the main.py, and it typically ranges between 100-1000 for most testing. Ideally this would be much higher as NEAT can take while to get off of the ground however these tests are being run on my laptop during pre-finals time.... so the available computer time is limited, though a few longer overnight tests were run(shown below): TODO: dont forget to add these, or remove this comment

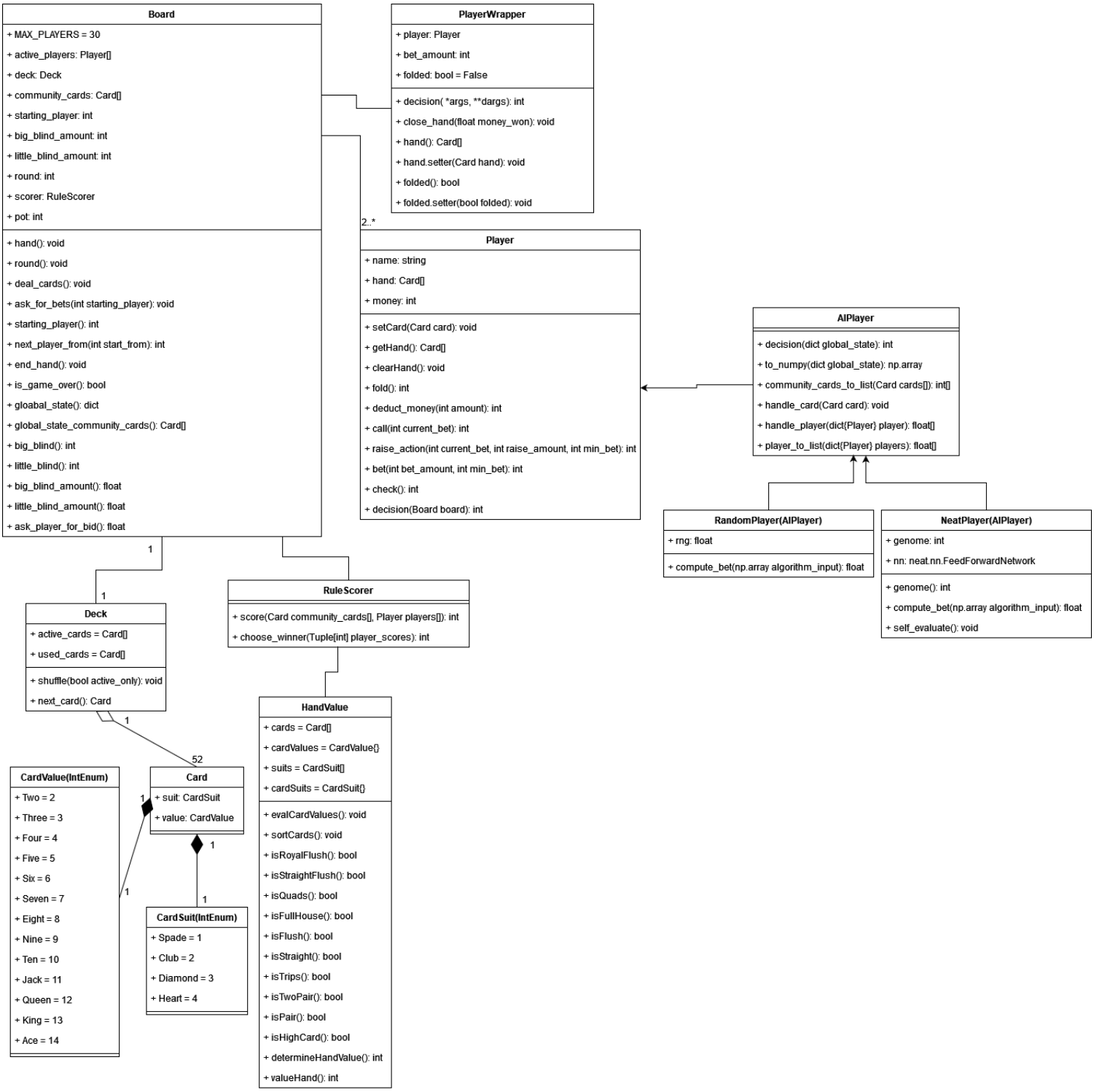

Figure 1: Class Diagram for our Poker AI

3.4 Source Code Description

Table 2: Source Code Files

File Name	Description
main.py	The driver of the program
visualize.py	Creates the graphs and visuals
AIPlayer.py	Implements AI Player class both random and Neat player
Board.py	Poker board (table) implementation. All the control logic for how the game is played is handled here
Card.py	A playing card implementation
Deck.py	Deck of Cards implementation
HandValue.py	Evaluator for hands value. Uses the official poker ranking for determining hand value
Player.py Mock_player.py	Poker Player functionality. Fold, check, bet, raise, call, cards in hand and money
test_AIPlayer.py	Test cases for the AIPlayer Class
test_Board.py	Test cases for the Board Class
test_Card.py	Test cases for the Card Class
test_Deck.py	Test cases for the Deck Class
test_HandValue.py	Test cases for the HandValue Class

4 Evaluation and Results

4.1 Results

As can be seen from the following graphs TODO: add images

4.2 Results Discussion

5 Conclusion

6 References

7 *

References

[1] N. Brown, T. Sandholm, and S. Machine, "Libratus: The superhuman ai for no-limit poker.," in *IJCAI*, 2017, pp. 5226–5228.

[2] A. Gilpin and T. Sandholm, "Optimal rhode island hold'em poker," in *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, vol. 20, 2005, p. 1684.

[3] N. Brown and T. Sandholm, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019. DOI: 10.1126/science.aay2400. eprint: https://www.science.org/doi/pdf/10.1126/science.aay2400. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aay2400.

[4] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron, "Improved opponent modeling in poker," in *International Conference on Artificial Intelligence, ICAI'00*, 2000, pp. 1467–1473.

[5] A. Gilpin and T. Sandholm, "A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation," in *AAAI*, 2006, pp. 1007–1013.

8 Appendix: Personal Contribution and Lessons Learned

8.1 JP

My personal contributions were to the brainstorming and deciding on the topic for the project, helping design the implementation of Texas Hold'em Poker in Python 3, creating and formatting the presentation, creating and the initial formatting of the final report. Writing the introduction, related works, a portion of the approach section, created the class diagram, creating the file description table, compiling the libraries used table. For the implementation I specifically wrote the hand value and player classes for the Texas Hold'em Poker Game as well as the tests for these class.

For lessons learned. I learned that the power and capabilities of a poker playing AI is really interesting and hard to get a complete one working. I discovered this while reading all the previous research and seeing how far they have come but also how far they have to go. The idea of an imperfection information problem is nothing new to humans and I believe we are uniquely suited for that kind of problem but computers are not. They are designed for complete information problems. When dealing with incomplete information problems with computers is where the human inspired neural networks come into play and they have had the most success in these problems. Early computers were just not powerful enough. So knowing where they came from to see Nathan and I be able to create pretty decent Texas Hold'em AI players is pretty neat (pun intended). Although we used a slightly simplified version it is still really rewarding to see the outcome.

One thing I did not expect was that during our training and testing of the models there was original one AI just completely destroyed the random AI. I would have expected it to take a while to learn a decently complex game. From this we put several AI players against each other instead of having the random AI and it improved all the AI players. It was really interesting to see the growth of the best AI model over generations of its development. Some times it was really clear who the best was and sometimes it was back and forth or sometimes the AI would regress in skill, which I did not think it would do.

8.2 Nathan