AiPoker - Implementation of a Poker Agent

John Hollén, Robin Berntsson, Simon Bergstöm

Abstract— We will probably write this section the last thing we do.

Index Terms—Monte Carlo Tree Search, Texas Hold'em

1 Background

Texas Hold'em is a popular variant of poker, not just to play with friends but also online against other people for money. Poker belongs with the most difficult kind of card game to solve discretely since it is an stochastic game with imperfect information. The challenge of creating the best poker agent has attracted a lot of people and is a popular subject in AI development.

The theory behind Monte Carlo search trees will be studied within this project and tested as a part of the logic that will determine the actions of the poker agent. Smaller algorithms to improve the logic for the AI will be tested and implemented in combination of the Monte Carlo Search Tree to create the best possible poker agent.

2 Theory

2.1 Characteristics of a Poker Player

Like previously described poker is a stochastic game, that makes it very hard to control and to define what a good poker player is. A good poker player does not always need to be the one who is a winning player. But in this project a aim has been set to try to create a poker agent who wants to win and will not take too much risk when playing. But to not make the agent to predictable the agent shall randomly switch behavior between aggressive and passive playing.

3 Method

3.1 Basic Game Engine

In order to even begin implementing an AI, the core poker game had to be implemented first. For simplicity the game implemented in this study is a two player game where a human player faces a computer AI. The player who starts is chosen at random when the game is started. The player that starts then puts one dollar in to the pot. The other player puts two dollars in the pot. Then the game is turn based like ordinary Texas Hold?em, the players can bet, check, call, raise and fold. And after each betting round cards are dealt to the table and the players combine the cards on hand with the cards on the table in order to get the strongest hand possible.

To be able to test the game engine the computer was first set to do moves completely at random. When all this was done the implementation of the AI could start.

3.2 AI - techniques

3.2.1 Monte Carlo Tree Search

MCTS is a best-first search strategy. Every node in the tree contain two values, an estimate of the expected value $V(P)$ of the reward $r(P)$ and the number of times the node have been visited n_i. MCTS only starts with the root node and incrementally build the tree by repeating the following 4 steps.

Selection
Selection is made by starting from the root node and then recursively select nodes until a leaf node is reached (this does not have to be a leaf of the game tree). To determine which node is the optimal one, the upper confidence bounds (UCB1) formula is used:

$$v_i + C \cdot \sqrt{\frac{\ln N}{n_i}} \tag{1}$$

The first term vi is the estimated value of the node and is responsible for exploitation of good nodes, the second term is made up by the total number of visits of the parent node N and the total number of visits of the current node n_i. The constant C allows tuning of the exploration-exploitation trade-off.

Expansion
If the leaf node reached is not an terminating node, which means that the game is not over at the current node. Then create one or more child nodes from the node currently at. This step is illustrated in fig(1).

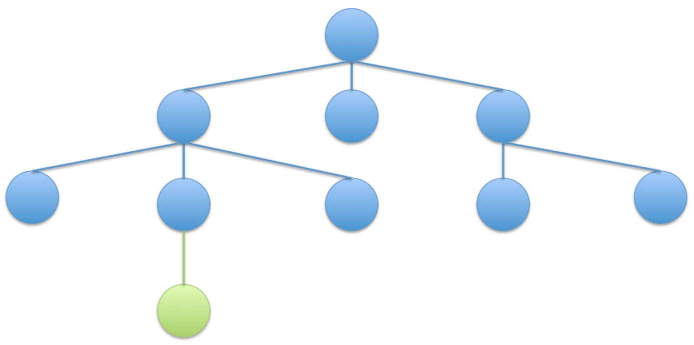

Fig. 1. Illustration of how the green node is added to the tree.

Simulation
The third step in MCTS is the simulation step. Here a game is simulated from the state in the new node until a result is achieved. The value of the reached result is recorded.

Backpropagation
The last step is to update the expected value and the selection counter for all nodes along the explored path by backpropagating the recorded result from the simulation step. This step is illustrated in fig(2). When the backpropagation has been done the whole chain starts over again starting with the selection stage. [REF!] One of the pros with MCTS is that it is anytime, that is it will give a valid solution to the problem even if it is interrupted before it ends. However the quality of the solution is an estimate and is expected to be better the more time the MCTS keeps running.

3.2.2 Hand Strength

Since Monte Carlo is a heuristic search which simulate a set of outcomes in a game to calculate what action to make the logic so far does not consider how good the cards on the agents hand is compared to

- *John Hollén, Robin Berntsson, Simon Bergström*

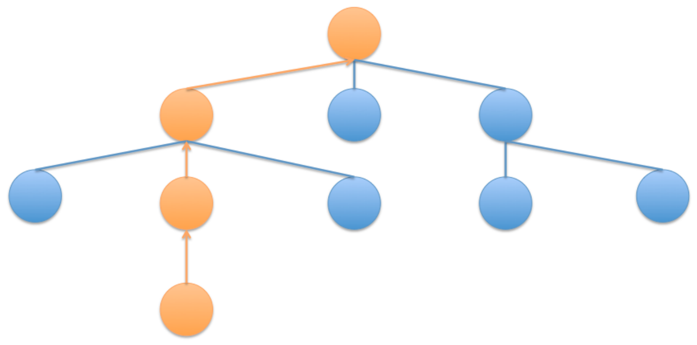

Fig. 2. Illustration of how the path to the node is updated with the score from the simulation in the whole tree.

other hands. To add this logic to the agent an algorithm could be used that would return a probability of how strong the current hand cards is with the cards on the table. This probability of the Hand Strength may, at the earliest be calculated the the flop has been shown. [REF!]

The Hand Strength is calculated by comparing the cards on the hand with all possible combination of two cards that the opponent can have and sum the time the agents cards wins compared to the opponent and divide by the number of combinations that was simulated.

3.2.3 Hand Potential

The hand potential algorithm calculates the quality of the hand as the game goes on. The algorithm is similar to the hand strength algorithm. The difference is that the hand potential algorithm considers all possible cards that have not been revealed yet. It is also heavier to calculate since it has to go through every possible combination of cards that could possibly end up on the table. It also takes into account every possible card the opponent might have on hand.

3.2.4 Effective Hand Strength

By combining the hand strength with the hand potential a probability of winning can be calculated, and this is called the effective hand strength and is calculated as following:

$$EHS = HS \cdot (1 - NPot) \cdot PPOT \tag{2}$$

3.3 Implementation

3.3.1 Main Design

The application implemented in this study was implemented in Javascript using the library Three.js for visualization. Three.js is a webGL library which provides high level functions for creating a 3D scene.

The application is implemented in an object oriented manner. The main application file is called main.js and it is here the scene is created and where all the other classes needed are initialized. The file main.js also contains all the click listeners for the buttons the human player uses to do moves. An overview over the game architecture is shown in fig(3).

3.3.2 Game

The main.js file is first painting the poker table and all buttons on the page, when you click the start button a GameState object and a AI will be created. The poker game is originated from the GameState object which only consists of a Cards object and functions for every state of the poker game Texas Hold'em. The game is created for two players where one player is an AI agent and the other player is the client controlled by a human. The Cards object consist of a full deck of cards and all methods needed for playing Texas Hold?em like shuffle the deck of cards, deliver the flop, turn card, river card and cards for the players.

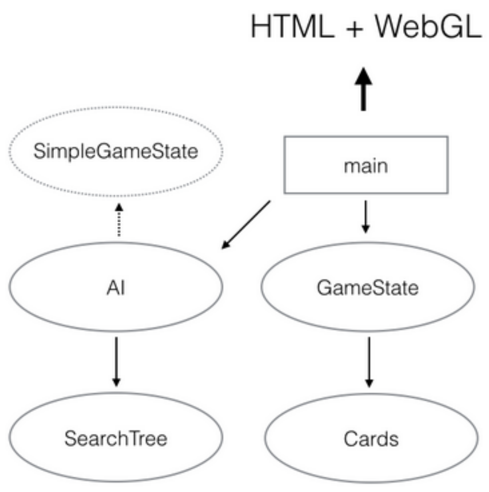

Fig. 3. Illustration of how the path to the node is updated with the score from the simulation in the whole tree.

4 Result

The result of this study is a Texas Hold?em game with an AI implemented using several different AI techniques and combining them together.

5 Conclusion

References

[1] Byron L., Wattenberg M. *Stacked graphs - Geometry & Aesthetics*. IEEE Transactions on Visualization and Computer Graphics archive Volume 14 Issue 6, Pages 1245-1252. IEEE Educational Activities Department Piscataway, NJ, USA ISSN: 1077-2626. 2008.

[2] Reynolds A. P., Richards G., Rayward-Smith V. J. *The Application of K-medoids and PAM to the Clustering of Rules*. In: Intelligent Data Engineering and Automated Learning - IDEAL 2004. Lecture Notes in Computer Science, 3177 . Springer-Verlag, pp. 173-178. ISBN 978-3-540-22881-3, 2004.