Implementing Artificial Intelligence Features In a No-Limit Texas Hold'em Poker Bot

By Zack McGinnis

Overview

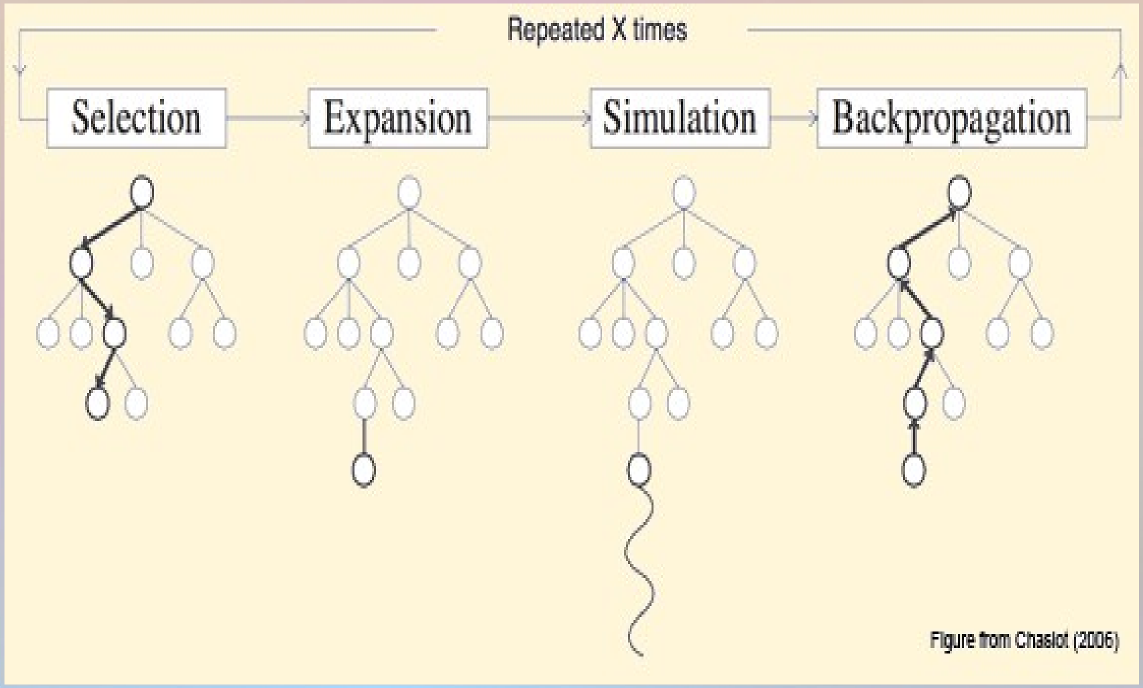
- The game of Texas Hold'em Poker
- Summary of my two added AI features:
  - SPR-Weighted UCT heuristic to the MCTS (Monte Carlo Tree Search) Bot.
  - Opponent model used to predict the probability distribution of the opponents next move.
- Results of my experiments
- Conclusions
- Demonstration?

Texas Hold'Em poker: a brief overview

- Difficult to account for stochastic elements (unseen cards, bluffing, etc)
- Players have the option to: check, bet, call, raise, or fold
- My bot will be playing against one opponent
  - No-limit betting rules (not fixed limit)
  - Our opponent will be a rule-based bot, without the capability to learn from previous games.

The MCTS (Monte Carlo Tree Search) Bot

- Modified existing MCTS Bot from open-source project
  - https://code.google.com/p/opentestbed/



- Monte Carlo Search Tree Algorithm
  - Selection: Starting at root node R, recursively select optimal child nodes (explained below) until a leaf node L is reached.
  - Expansion: If L is a not a terminal node (i.e. it does not end the game) then create one or more child nodes and select one C.
  - Simulation: Run a simulated playout from C until a result is achieved.
  - Backpropogation: Update the current move sequence with the simulation result (estimated value, and visit

Upper Confidence Bound as applied to Trees (UCT)

- Used in the first stage (selection) of MCTS
    - Acts as a heuristic to MCTS

$$\hat{V}(c_i) + C\sqrt{\frac{\ln \mathrm{T}(P)}{\mathrm{T}(c_i)}}$$

- Where:
    - $V(c_i)$ is the estimated value of the node,
    - (c_i) is the number of the times the node has been visited
    - P is the total number of times that its parent has been visited.
- C is a tunable bias parameter.

My weighted UCT implementation

- My UCT implementation will weight branches containing moves resulting in a higher SPR more heavily.
  - This will favor more aggressive actions throughout each iteration of the MCTS
  - e.g. More betting and raising, less checking and calling.

- SPR = Stack-to-Pot Ratio
  - ex. playerStack = \$100, currentPotSize = \$25, then SPR = 4.0

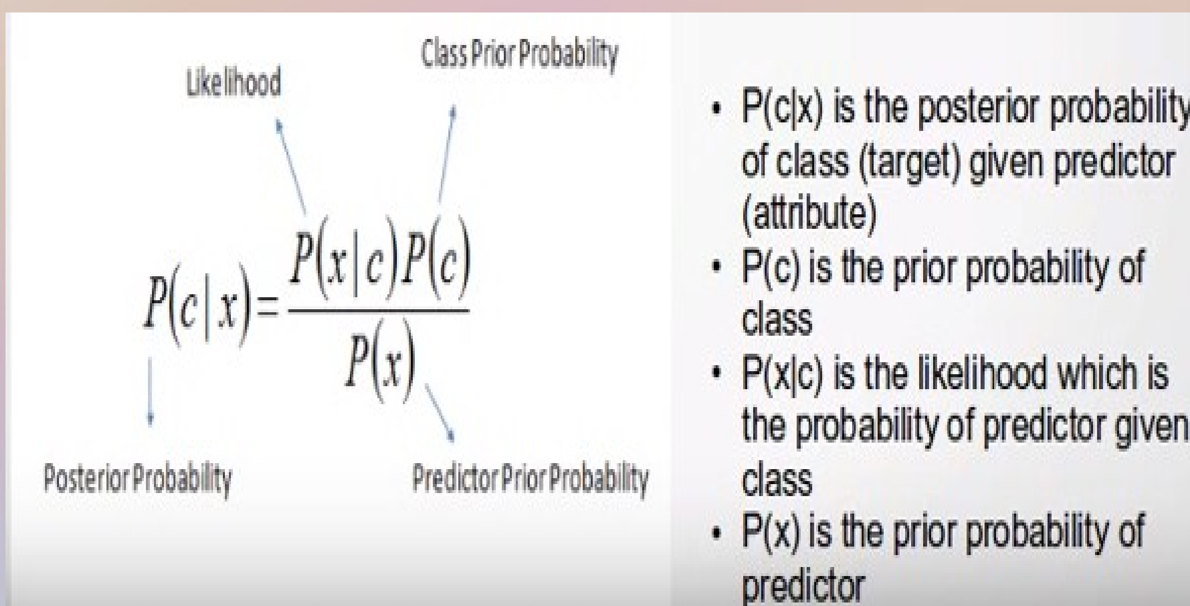
$$\hat{V}(c_i) + C\sqrt{\frac{\ln T(P)}{T(c_i)}} \quad x\,(S)$$

Where

$S = 1/SPR$

The Opponent Model

- Utilized during the "simulation" stage of MCTS



- NaiveBayes classifier
- WEKA (Waikato Environment for Knowledge Analysis) machine learning library
    - Trained on hand history data (10,000 hands) collected from a testbot vs. testbot game

The Opponent Model

- Models trained for many different gamestates:
  - 820 instances of post-flop check or bet actions
  - 2363 instances of post-flop fold, call, or raise actions
  - 10 instances of pre-flop check or bet actions
  - 6973 instances of pre-flop fold, call, or raise actions.

- Why NaiveBayes
  - Nominal actions of our opponent are classified based on attributes collected from previous hands
  - We can assign a probability of a class (action) occurring, given attribute values (approximately 32 per gamestate).
  - e.g. We can determine that the opponent will fold to a flop bet 80% of the time, so we assign 80% to our opponent node probability distribution for that particular gamestate

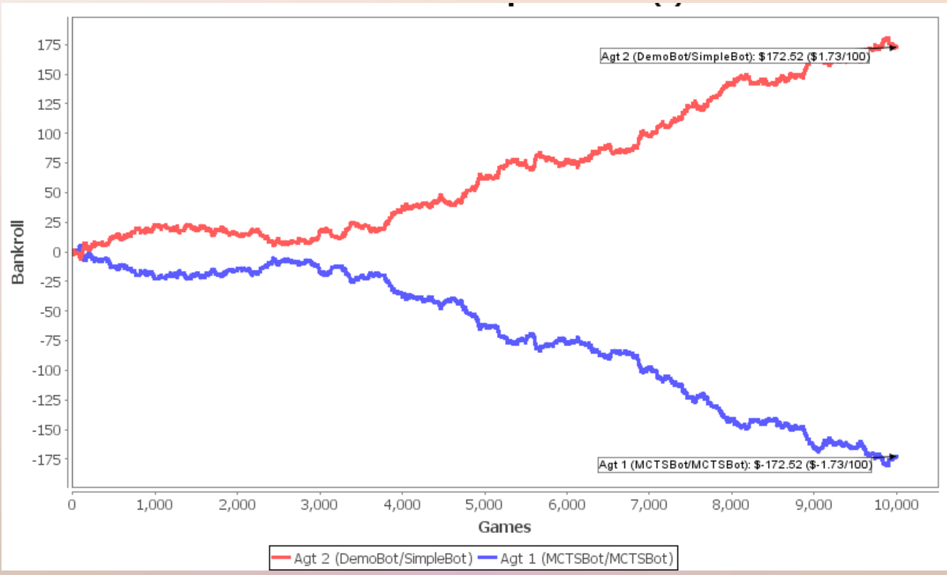
Experiment Setup

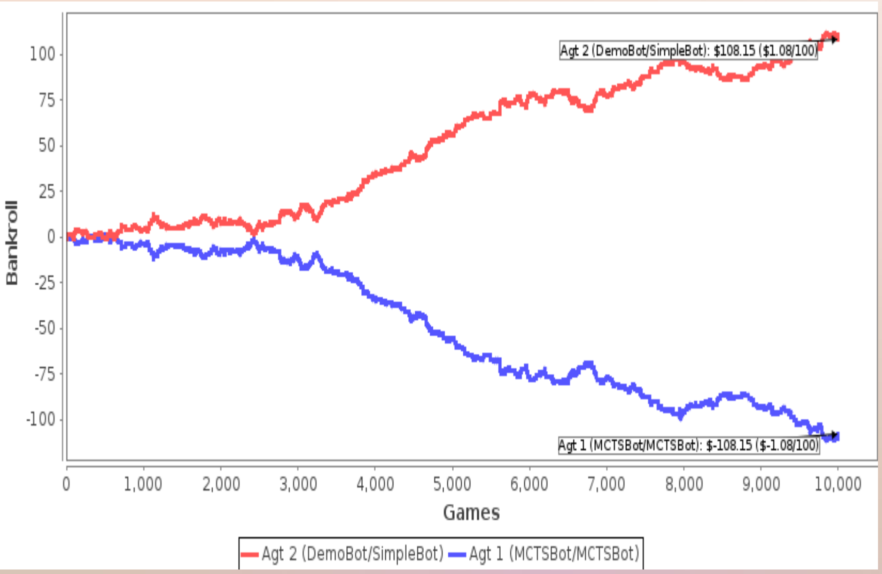
- Our opponent (testbot)
  - Rule based, (e.g. only makes decisions based on it's own two hole cards, and the exposed community cards)
  - No opponent model or ability to learn from prior games
- 10,000 games (hands) per test
- Y-axis = profit (measured in terms of bankroll ($1.00 per bankroll)
- X-axis = # of games
- Blue line = our MCTS bot
- Red line = testbot (opponent)

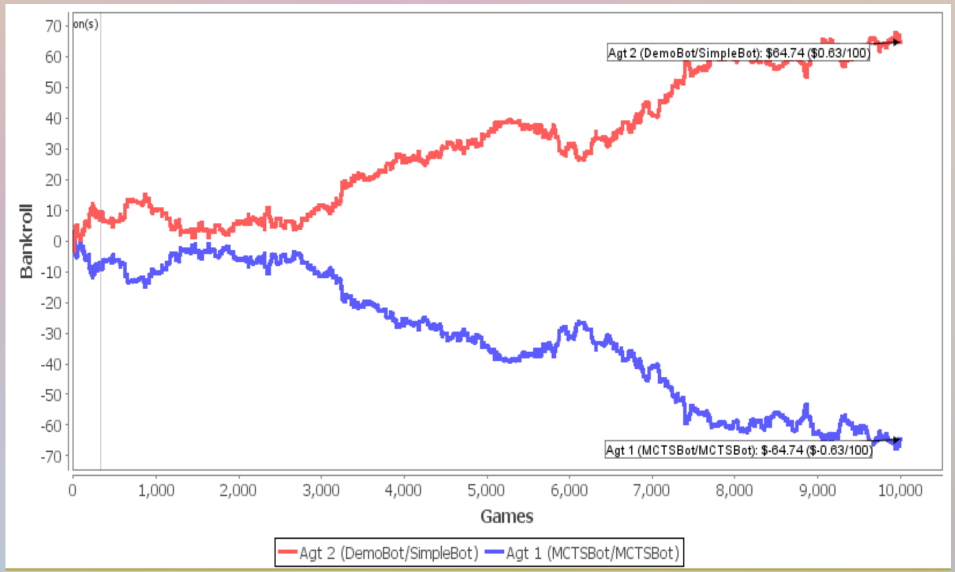
Experiment Setup

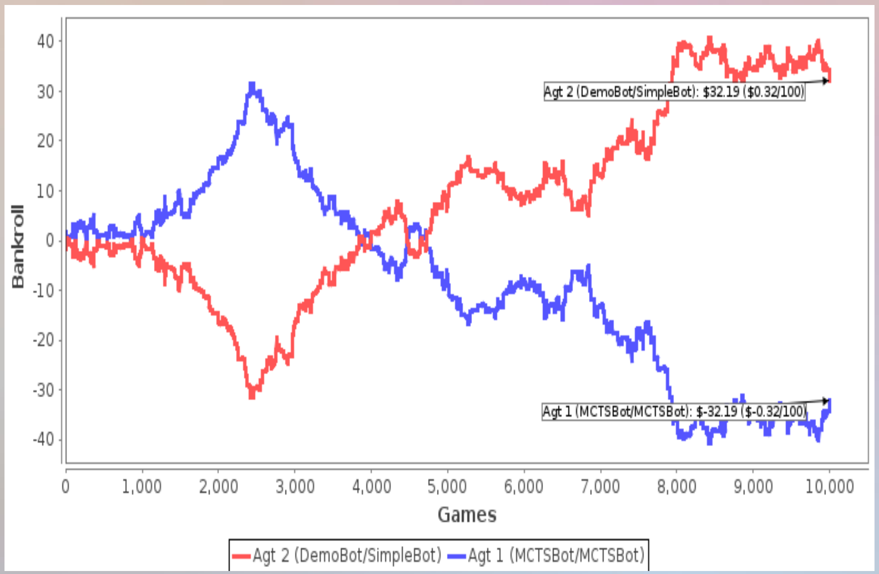
- Experiment #1
  - Standard UCT with no trained opponent model
- Experiment #2
  - Standard UCT with NaiveBayes trained opponent model
- Experiment #3
  - SPR weighted UCT with no trained opponent model
- Experiment #4
  - SPR weighted UCT with NaiveBayes trained opponent model

Results


Agt 2 (DemoBot/SimpleBot): $172.52 ($1.73/100)
Agt 1 (MCTSBot/MCTSBot): $-172.52 ($-1.73/100)
Bankroll
Games
Agt 2 (DemoBot/SimpleBot)
Agt 1 (MCTSBot/MCTSBot)


Agt 2 (DemoBot/SimpleBot): $108.15 ($1.08/100)
Agt 1 (MCTSBot/MCTSBot): $-108.15 ($-1.08/100)
Bankroll
Games
Agt 2 (DemoBot/SimpleBot)
Agt 1 (MCTSBot/MCTSBot)


Agt 2 (DemoBot/SimpleBot): $64.74 ($0.63/100)
Agt 1 (MCTSBot/MCTSBot): $-64.74 ($-0.63/100)
Bankroll
Games
Agt 2 (DemoBot/SimpleBot)
Agt 1 (MCTSBot/MCTSBot)


Agt 2 (DemoBot/SimpleBot): $32.19 ($0.32/100)
Agt 1 (MCTSBot/MCTSBot): $-32.19 ($-0.32/100)
Bankroll
Games
Agt 2 (DemoBot/SimpleBot)
Agt 1 (MCTSBot/MCTSBot)

Experiment Setup

- Experiment #1
  - Standard UCT with no trained opponent model
  - Lost at a rate of -$1.73/100 games
- Experiment #2
  - Standard UCT with NaiveBayes trained opponent model
  - Lost at a rate of -$1.08/100 games
- Experiment #3
  - SPR weighted UCT with no trained opponent model
  - Lost at a rate of -$0.63/100 games
- Experiment #4
  - SPR weighted UCT with NaiveBayes trained opponent model
  - Lost at a rate of -$0.32/100 games

Conclusions

- I'm less of a loser!
- Strange activity occuring around 2500-3000 games
    - Variance?
- Would like to create and test opponent models using different classifiers (SVM, Multi-layer perceptron, etc)

Demonstration?

- MCTS bot vs testbot
  - Visualization of game-tree and MCTS algorithm iterations