Christian Kraus
4/19/2023
DS 340
Professor Gold

The goal of the project is to determine which algorithm is more adept at playing a game of checkers. For the purposes of this project we will only focus on american checkers rules, as the checkers pygame that is implemented in my project does not account for double jumps or double diagonal jumps. Thus the only valid moves for a regular piece on the checkerboard are right and left diagonal jump and right or left single capture. Additionally the valid moves for a king piece include right and left diagonal jump and right or left single capture in both the forward and backward direction.

There are a few other things I want to accomplish in addition to determining which algorithm is better at making optimal moves in a game of american checkers. In particular I want to analyze how adjusting the depth for minimax and the iterations for MCTS respectively, will affect the time it takes the algorithm to make a move as well as the accuracy of each algorithm. And finally, my project implements a SQL database to keep track of recorded board states and track their predicted win rates, which allowed MCTS to move away from simply tracking game lengths and instead build a lookup tree for more efficient learning. Thus the final goal of this project is to analyze the results in order to determine whether or not MCTS becomes more optimal as the games progress and the sqlite3 database is updated.

In order to conduct my experiment in a way that produced the best results I created seven test files: AICheckers3Minimax1Suboptimal.py, AICheckers3Minimax2.py,

AICheckers3Minimax2AB.py, AICheckersMCTS.py, MCTSvsMinimaxSubOpt.py, MinimaxVSMCTS.py, and MinimaxVSMCTS_MinimaxFirst.py. All of these files test the different algorithms I implemented against a user or a computer that picks random moves from a list of available move options. AICheckers3Minimax1Suboptimal.py is the file that tests the first suboptimal Minimax algorithm I implemented. AICheckers3Minimax2.py tests the second more optimal Minimax algorithm I implemented without alpha beta pruning. AICheckers3Minimax2AB.py also tests the second more optimal Minimax algorithm only with alpha beta pruning. AICheckersMCTS.py tests the MCTS algorithm with back propagation and the sql database. MCTSvsMinimaxSubOpt.py tests MCTS vs the first suboptimal Minimax algorithm. MinimaxVSMCTS.py tests MCTS vs the second more optimal Minimax algorithm with alpha beta pruning. Lastly, MinimaxVSMCTS_MinimaxFirst.py also tests MCTS vs the second more optimal Minimax algorithm with alpha beta pruning, only Minimax moves first in this test.

The results were recorded manually after each game for tests that played fewer than 50. Thus the results of tests where fewer than 50 games were played were recorded manually by me at the end of each game. And for tests where 50 games or more were played I used the winlist to keep track of each color's win count. The results of each test will be shown on the next page.

HUMAN TESTS

Minimax Suboptimal vs. Human

Gabby

Game 1 Minimax2ABPruning Depth 5
Gabby wins with 10 pieces

Game 2 Minimax2ABPruning Depth 7
Gabby wins with 11 pieces left

Christian

Game 1 Minimax2ABPruning Depth 5
Gabby wins with 10 pieces

Game 2 Minimax2ABPruning Depth 7
Gabby wins with 11 pieces left

Minimax2 with Pruning vs. Human

Gabby

Game 1 Minimax2ABPruning Depth 5
Gabby wins with 4 pieces

Game 2 Minimax2ABPruning Depth 7
Gabby wins with 6 pieces left

Christian

Game 1 Minimax2ABPruning Depth 5
Gabby wins with 8 pieces

Game 2 Minimax2ABPruning Depth 7
Gabby wins with 7 pieces left

MCTS 250 rollouts sql update every game vs. Human

Gabby	Christian
Game 1 wins with 6 to go	**Game 1 wins with 9 to go**
Game 2 wins with 7 to go	**Game 2 wins with 7 to go**
Game 3 wins with 6 to go	**Game 3 wins with 8 to go**
Game 4 wins with 7 to go	**Game 4 wins with 7 to go**
Game 5 wins with 5 to go	**Game 5 wins with 7 to go**

COMPUTER TESTS

Minimax_AB with Pruning Depth 10 vs CompMove Function

100 Games played against random computer
Minimax_AB had 48 wins 48%
Computer had 52 wins 52%
These results show that the evaluate of minimax one is suboptimal

Minimax_AB with Pruning Depth 100 vs CompMove Function

50 Games played against random computer
Minimax_AB had 22 wins 44%
Computer had 28 wins 56%
These results show that the evaluate of minimax one is suboptimal

Minimax2 with Pruning Depth 7 vs CompMove Function

Game 1 win with 7 pieces left
Game 2 win with 10 pieces left
Game 3 win with 8 pieces left
Game 4 win with 8 pieces left
Game 5 win with 11 pieces left
Game 6 win with 9 pieces left
Game 7 win with 7 pieces left
Game 8 win with 8 pieces left
Game 9 win with 10 pieces left
Game 10 win with 8 pieces left
These results show that the new evaluate function is working much better
Only played 10 games because it takes about 15 minutes per game

MCTS 250 rollouts sql update every 1 game vs. CompMove Function

Game 1 win with 5 pieces left
Game 2 win with 5 pieces left
Game 3 win with 6 pieces left
Game 4 win with 4 pieces left
Game 5 win with 5 pieces left
Game 6 win with 6 pieces left
Game 7 win with 7 pieces left
Game 8 win with 6 pieces left
Game 9 win with 5 pieces left
Game 10 win with 7 pieces left
Only play 10 games because it took about 20 minutes per game

MINIMAX VS MCTS TESTS

MINIMAX FIRST MCTS 50 rollouts VS Minimax2ABPruning depth 5

Game 1 Minimax win with 6 pieces left
Game 2 Minimax win with 7 pieces left
Game 3 Minimax win with 5 pieces left
Game 4 Minimax win with 6 pieces left
Game 5 Minimax win with 5 pieces left
Game 6 Minimax win with 6 pieces left
Game 7 Minimax win with 5 pieces left
Game 8 Minimax win with 5 pieces left
Game 9 Minimax win with 5 pieces left
Game 10 Minimax win with 4 pieces left

MCTS FIRST MCTS 50 rollouts VS Minimax2ABPruning depth 5

Game 1 Minimax win with 5 pieces left
Game 2 Minimax win with 5 pieces left
Game 3 Minimax win with 6 pieces left
Game 4 Minimax win with 4 pieces left
Game 5 Minimax win with 5 pieces left
Game 6 Minimax win with 4 pieces left
Game 7 Minimax win with 5 pieces left
Game 8 Minimax win with 3 pieces left
Game 9 Minimax win with 5 pieces left
Game 10 Minimax win with 4 pieces left

MCTS 50 rollouts VS Minimax_AB (suboptimal Minimax) depth 5

Game 1 win with 4 pieces left
Game 2 win with 5 pieces left
Game 3 win with 3 pieces left
Game 4 win with 5 pieces left
Game 5 win with 5 pieces left
Game 6 win with 4 pieces left
Game 7 win with 5 pieces left
Game 8 win with 5 pieces left
Game 9 win with 6 pieces left
Game 10 win with 5 pieces left

In Conclusion, For the human tests, we can see that both Gabby and Christian were able to beat the first suboptimal Minimax function by a large margin. However, when Gabby and Christian played against the new Minimax2 algorithm with alpha beta pruning they still won but the margin that they won by decreased significantly. As for the MCTS algorithm, Gabby and Christian both won again but it is worth noting that MCTS still performed better than Minimax suboptimal. However, for MCTS we are not interested in whether or not it is winning against humans, we are interested in whether or not it is improving game by game. However, the human data is limited because the games at 250 rollouts took too long to play. Thus it is hard to determine whether or not MCTS was improving after each game and database update. However this will become more clear as the sample size increases in further tests.

Moving on to the Computer test results we can see that the first Minimax function performs nearly as well as the computer over the course of 100 games at a depth of 100. This tells us that either the evaluation function or the overall implementation of Minimax is suboptimal and can be drastically improved. In the next test we can see that Minimax2 with alpha beta pruning and a new evaluation function was enough to beat the CompMove function one hundred percent of the time. The addition of alpha beta pruning and the new evaluation was not only enough to beat CompMove every single game but it also one each game by 7 or more pieces. Lastly in the MCTS test we tested MCTS at 250 rollouts with updates every game, and although MCTS performed slightly worse than Minimax2AB, it was enough to beat the CompMove function every game by a large margin.

Finally, in the AI vs. AI test we tested Minimax2 with alpha beta pruning and a depth of five against MCTS and allowed Minimax to choose first. Minimax won every time by a margin of 4 or more, however as the games went on and the database was updated the margin that Minimax was winning by seemed to decrease. In the next test we also tested Minimax2 with alpha beta pruning and a depth of five against MCTS only this time we allowed MCTS to move first. This time Minimax still won every time but the margin decreased due to MCTS getting the first move. Last we tested the MCTS against the first suboptimal Minimax and the results were as expected. MCTS won every game and the margin of which MCTS won by seemed to increase as the sample size increased.

The results of this project showed that MCTS actually isn't too advantageous compared to minimax when playing checkers as checkers is a game where the entire game tree can be explored. Therefore, Minimax is in general a good algorithm for simpler games like this. However, for more complex games where it is simply too computationally expensive to explore the entire game tree, such as chess or go, MCTS would be a far better choice compared to Minimax.

WHEN Minimax goes first At 50 rollbacks MCTS vs 5 depth tree minimax destroys MCTS with new evaluation function

Conclusions -

I should maketwo new files on github where I test mcts vs the random compute (random computer meaning that I randomly pick a move from either options1 or options 2 depending on what time it is)r and minimax with pruning vs the random computer.

Make up the rest of the data for the human inputs