Introduction

Our approach was to use a particle filter to track all the possible board states and determine the most probable state out of the particles. Monte Carlo Tree Search would be used to determine the best move given the most probable state. We also had the idea of using minimax search and some evaluation heuristics to interface a hybrid setup with MCTS. Using minimax with alpha beta pruning over MCTS was another option. Sensing the squares would be based on the area of the board that was most likely to contain the opponent's previous move.

Methods Used

The unique aspects of the minimax search with alpha/beta pruning was the evaluation function and choosing the ordering of which possible states to first search over over others (for more effective pruning). The latter was calculated by prioritizing moves that lead to better evaluations over others. The evaluation function consists of 4 heuristics: material, piece placement, king safety, opening. A positive score was given for having a higher material count, having a piece in certain squares over others, having more pieces nearby the enemy king, and playing certain moves over others in the first few moves. Ideas like attacking king zone and piece-square tables came from chessprogramming.org.

A CNN was played around with to make evaluation, in which it trained on board positions given stockfish evaluations. Board positions were converted as inputs by setting a 8x8 matrix for each piece type (-1=black pieces, +1=for white pieces, 0=empty squares) along with some extra feature matrices for castling and en passant. The paper "Predicting Moves in Chess using Convolutional Neural Networks" was used as inspiration.

To track the potential chess board states without knowing the opponent's state, we used a particle filter with 2,500 particles. Each particle consisted of a potential board state and its associated probability. We updated the particle filter whenever we gained new information about the board, whether through sensing an area of squares, our piece being captured, or the result of our move. Whenever this occurred, we checked to see how closely a particle's board matched the observed information, and then we set the particle's probability based on how it matched the information. We then sampled new particles from our particle filter, giving the particles with higher probability a larger chance of being resampled and continuing to be tracked. On our turns, we selected the particle with the highest probability and its associated board state to pass into our minimax function. In this way, we made thousands of predictions about the opponent's moves, evaluated how likely they are based on our observations, and kept track of the most likely board states.

Results

The engine is able to select somewhat reasonable squares to sense and then make moves that follow general chess principles. Even without setting an opening book, the engine was able to reliably beat the hidden agent and the random agent. It captured pieces back when they were captured and also took pieces when given the chance.

Discussion

The neural network for evaluation performed terribly compared to handmade heuristics because of overfitting, and so the neural net was scrapped. More parameter tuning and optimization and perhaps additional data could have helped solve this. Given limited time and hardware limitations, this was not pursued further. Heuristics worked decently because they follow human intuition.

MCTS with minimax search for rollout also performed worse versus solely using minimax search. This could be that a different policy for MCTS was in order to make more reasonable moves, or that our hybrid MCTS minimax search approach was not well planned. The particle filter was able to keep track of a general sense of the possible board positions in order for the engine to sense squares and make moves that were reasonable. Resampling the particles every turn helped ensure relevant board states were being tracked.

For improvements, instead of determining the best move for the single most probable state, perhaps finding the maximum likelihood move based on some probability distribution of board states would result in a more consistent/effective agent.

Peer Review

Jacob engineered the entirety of the particle filter file.

Jesse implemented the minimax search and the evaluation functions.

Hasan created the MCTS algorithm, interfacing it with minimax search and also integrating all the components together in the my_agent.py.