CS 182 Final Project Status Update

Jacob Lurye, Walter Martin, Ryan Wallace

November 23, 2016

1 Progress

Originally, we intended to write the entire back-end framework for losing chess from scratch. After experimenting with this approach, it was decided that the time involved in creating a bug-free, efficient framework could be better spent on actual algorithmic problems. Instead, we have chosen to build our framework around the open source python-chess package. Since the dynamics of losing chess are identical to that of standard chess supported by python-chess, our framework inherits the majority of functionality from python-chess. However, since the rules for winning and making moves are different, we wrapped the python-chess board class with our own class which "overrides" the methods to find legal moves and to decide the status (win, lose, or draw) of a game.

In addition, we have built a class which, given two agents, plays a game by manipulating the state of the previously mentioned board class. On top of this, we have implemented a baseline alpha-beta pruning minimax search algorithm. Since exploring to full depth with this algorithm is computationally infeasible, we have also created a set of three evaluation functions that give a rough utility of a given game state (based on amount of material, number of pawns, and a combination of the these factors with whether pieces are in attacking positions).

When we face two alpha-beta minimax AIs against each other, we observe games that typically last 30-60 moves (15-30 per color). Moves seem to be made with decent foresight.

Finally, we have downloaded the records of losing chess games available online, and used the parser provided in the python-chess package to construct a list of representations of the games that can be used to encode the features for training an evaluation function.

2 Current Issues

Currently, our alpha-beta pruning minimax implementation searches 1-ply deep very fast (entire games are sometimes played in roughly a second; others in closer to 1 minute). However, searching to depth 2 takes on the order of 5 minutes per turn, unacceptably slow. The majority of the time is spent finding legal moves, and generating the successor game state for a move.

3 Next Steps

Our next steps are to investigate ways to speed up search (e.g., introducing parallelization), as well as to investigate TensorFlow and devise a transformation from our game state representation to features for training. Moreover, we will implement a way for humans to play against the AIs.