# Joshua Pfefferkorn

## Programming Assignment #3

## COSC 76, Fall 2021

### Description

#### MinimaxAI

Minimax is the algorithm that served as a backbone for this project. By recursing through the game tree, it selects an action with the highest possible ulitity given expected opponent actions. Given the sheer size of the chess game tree, I implemented a depth limit to cap Minimax's search.

#### AlphaBetaAI

Using most of minimax's code, alpha-beta adds additional parameters alpha and beta in order to "prune" branches that no longer have the potential to return a better action than one already found. Furthermore, nodes can be ordered in a way that allows pruning to work more efficiently; positioning nodes with high potential on the leftmost side of the tree causes the algorithm to search those first, allowing it to prune more branches later in the tree. Note that alpha-beta with and without pruning returns the same value as minimax; pruning simply improves computation speed.

#### IterativeDeepeningAI

Iterative deepening is set up to call all of alpha-beta's functions, the one difference being that it calls alpha-beta at each depth up to the depth-limit parameter, taking the deepest move. This allows verification that alpha-beta finds (generally speaking) better-value moves at deeper depths.

### Evaluation

Runtime speeds, nodes searched, and returned moves indicate that my algorithms are working. Minimax is the slowest, searching the most nodes. Alpha-beta without move-ordering is faster, searching fewer nodes yet returning the same move. Finally, alpha-beta with move-ordering is the fasts. Print statements, pasted below in the document, evidence these trends.

### Discussion

**Vary maximum depth to get a feeling of the speed of the algorithm. Also, have the program print the number of calls it made to minimax as well as the maximum depth. Record your observations in your document.**

*Depth 1:*

- Calls made to minimax: 24545

- Maximum depth: 1

*Depth 2:*

- Calls made to minimax: 420446
- Maximum depth: 2

Depth 3 is notably slower than depths 1 and 2; as the depth increases, the algorithm must search exponentially more nodes.

**Alpha-Beta vs. Minimax**

*Alpha-Beta at depth 2:*

- Move 1:
  - Best value: -23.5
  - Nodes searched: 2788
  - Time elapsed: 0.3811788558959961
- Move 2:
  - Best value: -15.0
  - Nodes searched: 3834
  - Time elapsed: 0.4789769649505615
- Move 3:
  - Best value: -8.5
  - Nodes searched: 6708
  - Time elapsed: 0.781836986541748

*Minimax at depth 2:*

- Move 1:
  - Best value: -23.5
  - Nodes searched: 8857
  - Time elapsed: 0.9479649066925049
- Move 2:
  - Best value: -15.0
  - Nodes searched: 18989
  - Time elapsed: 2.146299123764038
- Move 3:
  - Best value: -8.5
  - Nodes searched: 32558
  - Time elapsed: 3.55916690826416

As anticipated, both minimax and alpha-beta return moves with the same value, though alpha-beta searches fewer nodes and returns a move in less time.

**Describe the evaluation function used and vary the allowed depth, and discuss in your document the results.**

My evaluation function took into account both the material value of the pieces (pawn = 10, knight/bishop = 30, rook = 50, queen = 90, king = 100) as well as the positional value of the pieces scored from -5 to 5 (e.g., knights and bishops

are less valuable on the edges of the board, etc). The positional value chart I used was found here: https://stackoverflow.com/questions/59039152/python-chess-minimax-algorithm-how-to-play-with-black-pieces-bot-has-white. Captures are weighted much more highly than positioning, but using positional values allowed the algorithm to make reasonable moves when no captures were found within the depth searched.

**Record your observations on move-reordering in your document.**

*Alpha-Beta with move-reordering at depth 2:*

- Move 1:
  - Best value: -23.5
  - Nodes searched: 2788
  - Time elapsed: 0.31617307662963867
- Move 2:
  - Best value: -15.0
  - Nodes searched: 3834
  - Time elapsed: 0.4433128833770752
- Move 3:
  - Best value: -8.5
  - Nodes searched: 6708
  - Time elapsed: 0.7813167572021484

*Alpha-Beta without move-reordering at depth 2:*

- Move 1:
  - Best value: -23.5
  - Nodes searched: 4176
  - Time elapsed: 0.4470231533050537
- Move 2:
  - Best value: -15.0
  - Nodes searched: 10240
  - Time elapsed: 1.10567307472229
- Move 3:
  - Best value: -8.5
  - Nodes searched: 12480
  - Time elapsed: 1.3961868286132812

As anticipated, alpha-beta returned the same value move with and without move-reordering, yet move-reordering enabled the algorithm to prune more nodes and thus search fewer nodes and improve computation speed.

**Verify that for some start states, best_move changes (and hopefully improves) as deeper levels are searched. Discuss the observations in your document.**

*Iterative deepening at depth 2:*

`best_move` values for move 1:

- Depth 0: -25.0
- Depth 1: -20.5
- Depth 2: -23.5
- Depth 3: -19.0

`best_move` values for move 2:

- Depth 0: -16.5
- Depth 1: -12.0
- Depth 2: -15.0
- Depth 3: -10.5

`best_move` values for move 3:

- Depth 0: -11.5
- Depth 1: -6.5
- Depth 2: -8.5
- Depth 3: -4.0

As shown, the alpha-beta algorithm generally finds moves with better values as it searches deeper levels in the tree.