CPSC 481

C. Ryu

**Assignment #1**

**Oct. 8 2014**

a)

Members:

Brian Brick

Brian Mitzel

Chris Doan

Reynaldo Galvan

Ricardo Lucero

E-mail address:

[bmitzel@csu.fullerton.edu](mailto:bmitzel@csu.fullerton.edu)

[brianbrick@csu.fullerton.edu](mailto:brianbrick@csu.fullerton.edu)

[cdoan@csu.fullerton.edu](mailto:cdoan@csu.fullerton.edu)

[dorkii10@csu.fullerton.edu](mailto:dorkii10@csu.fullerton.edu)

[luceror27@gmail.com](mailto:luceror27@gmail.com)

Contribution %:

20

20

20

20

20

b)

Our program was written in Python and you will run the play.py script through the Python interpreter on your machine.

Command line instruction:

> python play.py

**\*\*\*Note: You need to have the Python 3.4.1 interpreter installed for accurate results\*\*\***

c)

Mini-Max algorithm:

Using the mini-max algorithm, we looked at possible moves to be taken and attributed a high score value to moves which put the most(or least) squares under attack and traced through as many as 4 moves ahead to determine the best course of action. Illegal moves according to the rules of chess were disallowed. Also, moves which put the rook at risk of being taken were given an extremely high magnitude negative value to avoid losing the ability to achieve checkmate. Likewise, stalemate is also given a high magnitude negative value because it is not an optimal outcome. During player X’s turn (who has the advantage), the tree branch with the highest heuristic value is determined and chosen for making the next move. During player Y’s turn, the lowest possible heuristic value is chosen for its next move. For alpha-beta pruning, we stop searching for new moves when the value for player X (alpha) is maximized, or the value for player Y (beta) is minimized. Because of the way mini-max works, the black heuristic “assumes” that player X will never move into a less-than-optimal situation for itself, such as where the rook could be taken, so a limitation exists because such moves are eliminated from consideration because heuristic value is too high during minimization.

White Heuristic(X):

To achieve checkmate, player X must restrict the black king as much as possible by pushing it against an edge or into a corner. Our heuristic function can calculate how many of the possible squares are currently un-maneuverable by the black king; the higher the number, the better for player x. If player y can no longer make legal moves and is under attack, this is checkmate.

Black Heuristic(Y):

The black player should try to avoid checkmate for as long as possible, as well as taking the white rook if it doesn’t compromise itself. This requires avoiding the White king and the board edges. We have prioritized moving away from the nearest edge including moving diagonally away from 2 equidistant near edges.

d)

Our program did not reference any particular source program from another individual, group, or classmate not included in our group.

e)

Through this assignment, we learned about the effectiveness and some limitations of the mini-max algorithm such as the occasional disadvantage of ignoring certain possibilities because one player must assume that the other will play the same way. We learned that when thinking programmatically and trying to impart our knowledge of chess into a heuristic for winning and avoiding loss, it can be relatively trivial but potentially very time or resource intensive if you don’t think ahead.