Danny Lin

CS 4613

Artificial Intelligence Mini Checkers Game

**Requirements:**

1. Must have Python 2.7 installed
2. Install pip
3. Install pygame using *pip install pygame*
4. Run the command *python2 checkers.py*

**High Level Description of Design and Program**

This checkers game is a simplified version of the original 8 x 8 checkers. This version is 6 x 6. The application has a Graphical User Interface for the user to interact with the checkers board. When the user first starts the application, the user chooses a difficulty level which is first defaulted to Level 1 difficulty. The user can choose Level 1, Level 2, or Level 3. Then the user can decide if they want to go first or second. When the user selects who goes first, the game board will be loaded and the game will commence. When it is the human player’s turn, the person can click a checker piece and if there are valid moves available for that piece, the possible moves will be highlighted by the interface. Once the player makes a move, the computer will utilize the alpha beta search algorithm to determine the best move available for itself. When there is no legal move for one side, the turn will be skipped and the other player will go again. When both sides no longer have any legal moves, the game is over. Whoever has more remaining pieces after the game is over, that will determine the winner. The game will also end if one of the players runs out of pieces. There are no kings in this version and there are no multiple jumps. The alpha beta search algorithm will generate copies of the current board and utilize the algorithm to find the best next move. The Graphical User Interface is created using Pygame.

**Definition of Terminal States and Their Utility Values**

There are several terminal states.

1. Human player runs out of pieces -> Computer wins
2. Computer runs out of pieces -> Human player wins
3. Both players run out of legal moves -> Player with more pieces remaining wins, if same number of pieces, then t ie

The utility values are 1000, 1000, and 0. 1000 value indicates that the computer player wins. 1000 value indicates that the human player wins. 0 value indicates that there is a tie between the two players.

**Explain How Your Evaluation Function Works and the Heuristics You Use**

My evaluation function uses 4 different heuristics to try to find the next best move. The first heuristic that I use is the number of pieces each side has remaining. Each piece the computer has is +100 and each piece the human player has is -100. The second heuristic I used is number of pieces each side can currently capture. For each piece that the computer can capture, it is +250 and each piece that the human can capture, it is -250. The third heuristic I used is the number of pieces each side has on the opponent’s side of the board. For each piece the computer has, it would be +200 and each piece the human player has, it would be -200. The last heuristic that I used is the number of protected pieces each side has. Protected piece means that the piece cannot be jumped over. Each computer protected piece would be +150 and each human protected piece would be -150. The evaluation function would return the value to the min and max functions.

**Explain How Different Levels of Difficulty Are Implemented**

I implemented different levels of difficulty by cutting off the tree at a certain depth. In my constants file, I set the depth for level 1 to be 5, level 2 is depth of 10, and level 3 is depth of 20. These all evaluate the best move within 15 seconds. Whenever the depth is reached, there will be cut off.