Com S 572 Fall 2020

Project: A Checkers-Playing Agent

Summary:

Win rate for average human player is 50%.

It will normally take 3-6 seconds for the agent to process and respond.

Used alpha-beta pruning algorithm with evaluation function to improve agent's move.

Used Java Jpanel as GUI to display checker board and status.

Main function should be able to do the following:

a. Take as input a move from the user.

b. Update the board with the user's move.

c. Output the agent's move from the alpha-beta search.

d. Update the board with the agent's move.

e. Repeat the steps until the end of the game.

Note:

- When using alpha-beta pruning, we have to be careful when after several depth search, there may no possiable moves for the agent and returned a nullpointerexception. This usually happens when agent is going to lose and it cannot survive before reaching search at depth n. this can be eliminated by checking new legal moves is null or not after appluing an move to the state.
- During the alpha-beta search phase, we create new instance of CheckersData for boards. So we do not make any modification to the board pass by original board. We only make and apply further moves to our newly created instance so it will not affect boards in other nodes when trackback and go through another path. This approach maybe an drawback that requires longer processing time.
- Min-node: compare alpha, change beta, if current value <= alpha, prune remaining branches.
- Max-node: compare beta, change alpha, if current value >= beta, prune remaining branches.

Compare the effect of increasing search depth:

In my implementation:

When depth is less than 6, the agent will respond immediately (less than 1 second).

When depth is 6, which is our case, the agent will respond 3-6 seconds.

When depth is 7, the agent will respond around 8-15 seconds.

The time increase greatly as we increase depth value.

Evaluation function:

In my implementation, two combination of evaluation functions are used:

1. Scores calculated from overall number of checkers' difference: if black checker becomes negative, it means that black lose more pawns than red side, with more weight on King Checker.
2. Scores calculated from the trends to becoming king. If value is positive, then the resulting move black pawns can turn into King more easily than red pawns.

Screen shots:

```
        selectedRow = move.toRow;  // Since only one piece can be moved, select i
        selectedCol = move.toCol;
        repaint();
        return;
    }


/* The current player's turn                    o the other player.
   Get that player's legal m                    as no legal moves,
   then the game ends. */
if (currentPlayer == Checkers
    currentPlayer = CheckersD
    legalMoves = board.getLeg                   ;
    if (legalMoves == null) {
        gameOver("BLACK has no moves.  RED wins.");
    } else {
        message.setText("BLACK:  Now AI's turn.");
    }
```

As we can see the agent (black) tries to maximize number of kings. And a case where agent win: