CS1006 Project 3 – Checkers/Draughts

Introduction

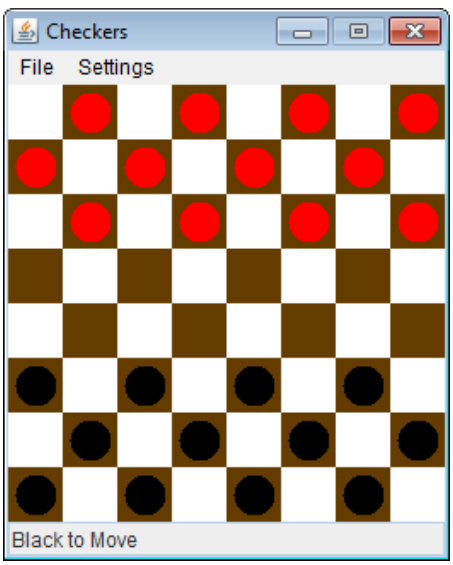
This project is about creating an implementation of Checkers (UK: Draughts), where one human player can play against the computer or two human players can play each other.

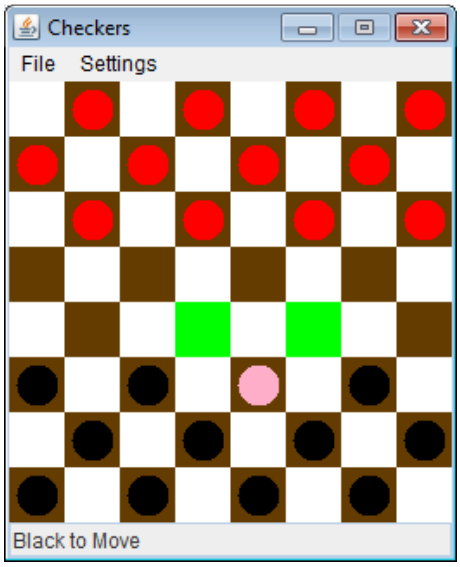
History

Checkers-like games have been found in Iraq and Egypt and are thought to have been played back in 3000 and 1400 B.C. respectively (Rayment). Alquerque, the Egyptian variant, is said to have been played all over the western world for centuries (Rayment). Both of these games, however, were played on different boards and with different number of pieces. Around 1100 a Frenchman got the idea of playing the game on a chessboard, which meant expanding the number of pieces to twelve a side. It was therefore called "Ferses" or "Fierges". Later, around 1535, the rule which states that players must capture an opponent's piece if possible was added (Neznanich). The French version now became known as "Jeu Force", which was later exported to the United Kingdom where it became known as Draughts.

Checkers: The Rules

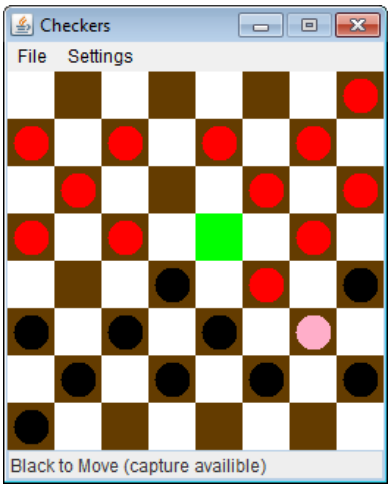
Checkers is a game with two players; black and red. The player with the black pieces starts, and the board is set up as seen below to the left. Each player starts with twelve pieces which can move diagonally toward the other side of the board, for instance, the pink[1]-colored piece in the picture below can move to either of the green squares. Only one piece can occupy a single cell at any one time.



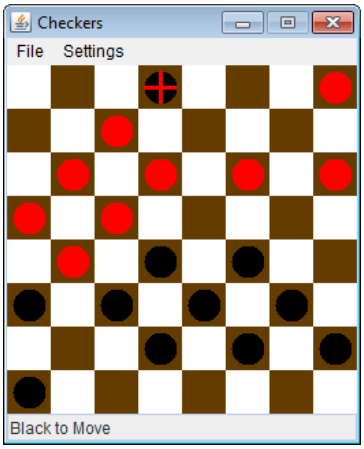


A piece can capture an enemy piece by jumping over it. Pieces can only capture diagonally in the direction of its movement, and the destination cell has to be empty (see left picture).

[1] Note: The pink color is not part of actual gameplay, but is added for visual effect.



The pink piece can capture the red piece (i.e. remove it from the board) by jumping to the green square. A piece becomes a "king" if it reaches the opposite side of the board (see right). Kings can move and capture diagonally in all directions. If a capture is available it has to be taken, and if several are available, the player is free to choose. Multiple captures can be made in a single move, if after a capture, the piece used to capture can capture another piece.



Winning & Losing

The game ends when one player cannot make a move, i.e. if they have no more pieces or none of the pieces have legal moves.

Draw

"A game is declared a draw when neither player can force a win." (Rogers) A draw can be declared any time both players agree to it.

Numbered Board

In order to summarize a game in terms of moves, a cell number notation is sometimes used.

Key Features of Implementation

- Allows two human players to play each other in accordance with the rules:
  - Displays the game in progress.
  - Keeps track of whose turn it is to play.
  - Only accepts valid moves.
  - Requires a capture to be made, if available
  - Option to play with multiple captures.
  - Support kings.
  - Recognizes when the game is over.
  - If a human player presses the left mouse button on a piece the program shows the possible positions they can move to with that piece, if any.
- Allows a human player to play against the computer
- Allows two computers players to play each other.
- Can save/load games.
- Can record/replay games.
- Three move openings from the official list.

Design

Explanation of Classes

Class	Purpose
checkers.Board	Deals with the graphical representation of the board.
checkers.BoardLogger	Logs the moves of a game.
checkers.CheckersApplet	Presents the board to the user, lets the user move pieces on the board and controls the AI opponent.
checkers.CheckersFrame	Controls the main window and menus.
checkers.Constants	Defines constants which are used elsewhere.
checkers.FileChooser	Lets the user open or save a file of a given file format.
checkers.GamePosition	Represents the state a game is in, including all of the pieces, the player whose move it is, whether the current player has a capture available, and the last piece used to capture with.
checkers.ThreeMoveOpening	Used to apply three move openings to GamePositions.
checkers.wrapper.Cell	Wrapper class used to store the row and column of a cell.
checkers.wrapper.Circle	Used to determine if a mouse click is on a Checkers piece. It extends Point because a circle has a position.
checkers.wrapper.GameStyle	Describes the four different types of games: PVP * (Player-vs-Player), PVC (Player-vs-Computer, i.e. player starts), CVP * (Computer-vs-Player, i.e. computer starts), CVC (Computer-vs-Computer).
checkers.wrapper.Move	Represents moves (i.e. the piece which is to be moved and the destination cell).
checkers.wrapper.Piece	Stores information about a piece on the board. This includes the player two owns it, it's location on the board and whether or not it is a king.
checkers.wrapper.Player	Describes the players of the game: The one starting at the bottom of the board, namely BELOW, and the other starting at the top of the board, namely ABOVE. These players may also be called "Black" and "Red".
checkers.wrapper.Point	Used to represents points on a surface.

Circle and Point classes

The main inspiration for the Circle and Point classes was to be able to pick up pieces from the board. This was done by finding the centre of the circle and then determining whether the distance from the centre of a piece to a clicked point is less than the radius of the circle. This is all done in the Point.distanceTo(Point) and Circle.contains(int, int) methods, and these were inspired by code found on the Computer Science Department at Princeton University's website (Sedgewick and Wayne).

Menu vs JMenu

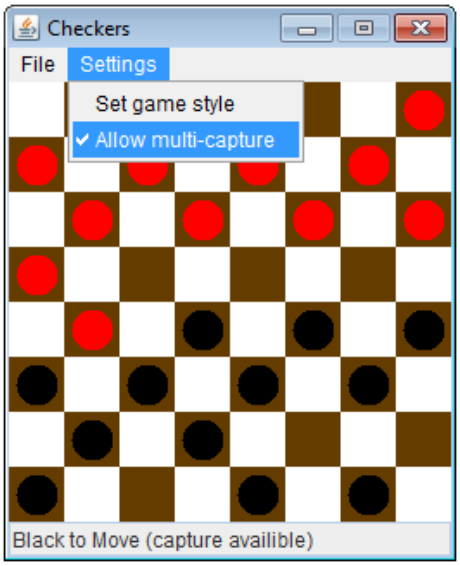
The menus were initially implemented as instances of javax.swing.JMenu, but this caused them to appear under the board applet. Therefore they were replaced by java.awt.Menu objects, which are so called “heavy-weight” components. A heavyweight component is “one that is associated with its own native screen resource” (Mixing heavy and light components), as opposed to lightweight components ‘which “borrow” the screen resource of an ancestor’ (Mixing heavy and light components).

Flickering

Initially, the applet would flicker as it updated. This was because it takes time to draw it and the image on the screen updates during drawing. Hence, this was avoided by using a back-buffer (McGuffin) onto which the image is first drawn and then drawing that buffer onto the visible Graphics context.

Allow multiple captures

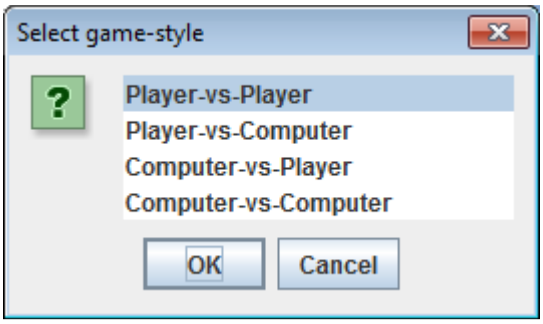
The user can choose whether or not to allow multiple captures through the Settings menu:



This Java called used to create this menu-item is CheckBoxMenuItem.

Selecting GameStyle

Setting GameStyle is done through the Settings menu.



The user has the option between Player-vs-Player, Player-vs-Computer (player first), Computer-vs-Player (computer first) and Computer-vs-Computer.

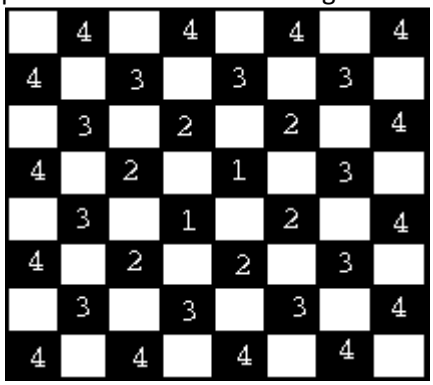
This popup window is made using JOptionPane.showMessageDialog with a JList as its message.

The AI

The AI, artificial intelligence, works by getting all the available moves, and choosing a random one based on the best resultant GamePosition (evaluation is discussed bellow). A recursive minimax algorithm was attempted, but not implemented successfully. This attempt can be seen in the MiniMaxSource folder.

GamePosition evaluation

The AI opponent evaluates GamePositions based on a number of factors. These include the pieces are kings, whether the pieces have valid captures, and the pieces position on the board. The positions on the board are given the following values (Pinto):

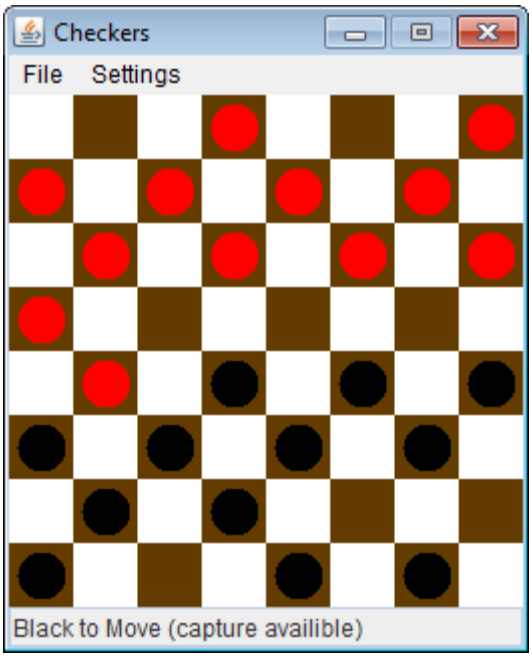


Other factors which could be used to determine the value of a position include how far away each piece is from being a king.

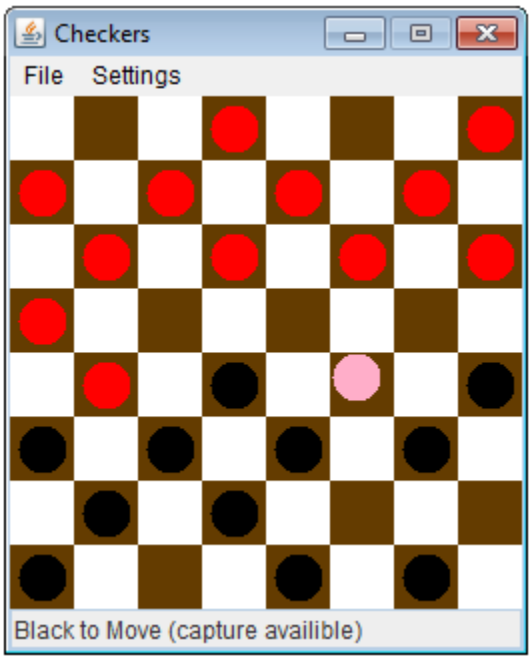
Testing

Captures

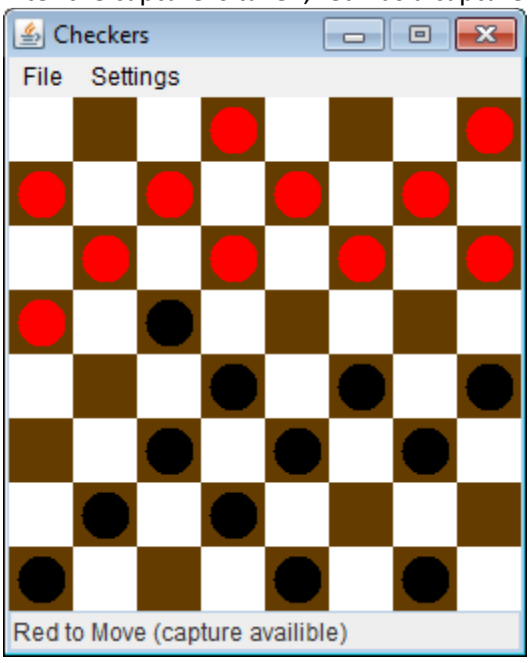
If a capture is available, the status bar not only says whose turn it is, but also says that a capture is available.



Additionally, it is not possible to move pieces when a capture is available. One of the available capture(s) has to be taken (in this case there is only one). Therefore if a different piece is selected, no squared are highlighted in green:

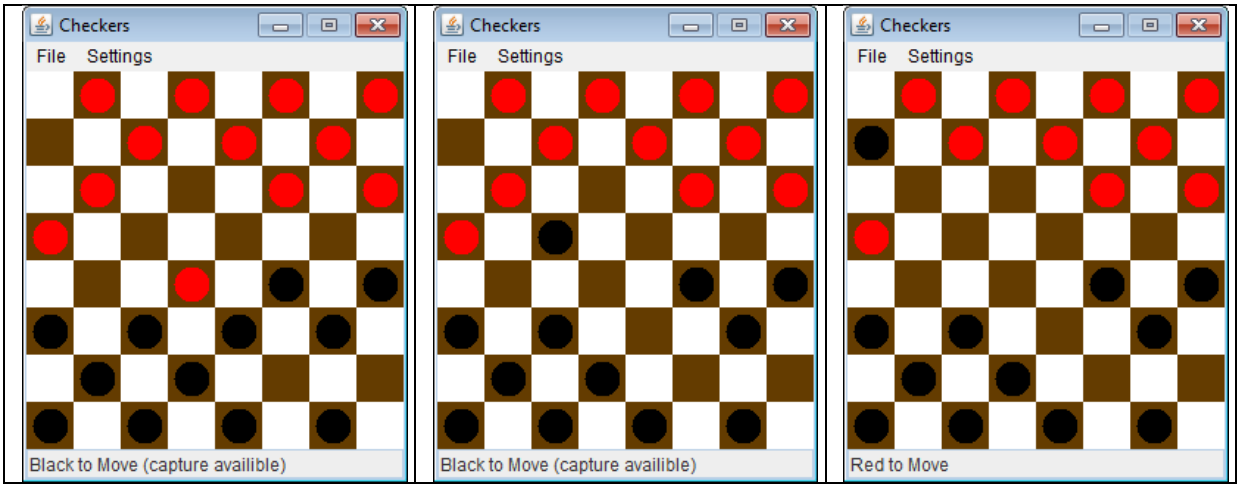


After the capture is taken, red has a capture available:



Double captures

Double captures can, as mentioned earlier, be allowed or disallowed. If they are allowed, the user has to make each capture separately, but in the same turn:



Replay

The user can choose to Replay a previously saved game through the File menu. Once a game is being replayed, there is a 3000 interval between each move. The user might find this confusing, and in the future this could be improved by adding next- and previous- move buttons, and not automatically moving on to the next move.

Limitations

There are several limitations with the current implementation. The AI has a very limited depth (looking only one move ahead, the user cannot set up the board as they wish, and the AI cannot recognize draws.

Conclusion

In conclusion, I would like to say that I think this project has been reasonably successful apart from the fact that I was unable to get the mini-max algorithm working. It has given me significantly further insight in graphical user interfaces in Java and taught me to play checkers.

In the future, a better AI would be desirable. This would most likely be implemented using a mini-max algorithm with alpha-beta pruning. Additionally, it would be beneficial to be able to set up the board as the user wishes, some recognition of a draw (by the AI) and ability to suggest the best move possible for the current play. A settings file where the settings (game-style and multiple capture options) are automatically saved could also be useful.

Bibliography

McGuffin, Michael J. Applet Tutorial: Backbuffers. University of Toronto. 22 April 2010 <http://www.dgp.toronto.edu/~mjmcguff/learn/java/07-backbuffer/>.

Mixing heavy and light components. Oracle Corporation. 22 April 2010 <http://java.sun.com/products/jfc/tsc/articles/mixing/>.

Neznanich, Modar. Game10. 9 May 2010 <http://www.modaruniversity.org/Game10.htm>.

Pinto, Paulo. Minimax explained. 28 July 2002. 23 April 2010 <http://ai-depot.com/articles/minimax-explained/>.

Rayment, W.J. History of Checkers. 9 May 2010 <http://www.indepthinfo.com/checkers/history.shtml>.

Rogers, Timothy J. Rules - Checkers. 6 May 2010 <http://www.darkfish.com/checkers/rules.html>.

Sedgewick, Robert and Kevin Wayne. Case Study: Purple America. Princeton University. 9 April 2010 <http://www.cs.princeton.edu/introcs/35purple/>.