Minimax Tree for Corners Game

In this homework, you are going to implement Minimax Tree Search on a simple board game.

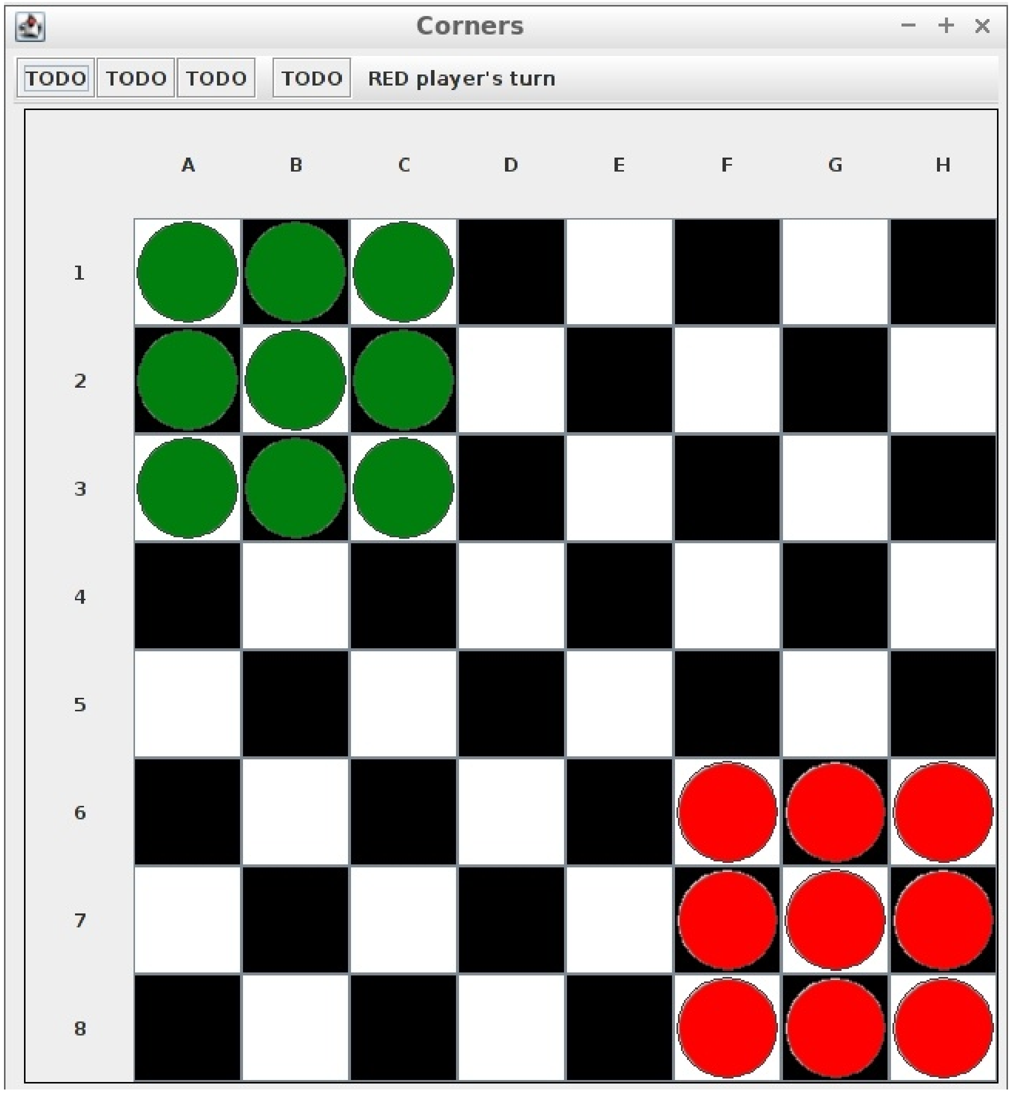

Figure 1: Initial state of the corners game.

Below are the rules of the game.

1. Red player makes first move.
2. Red player tries to gather its pieces in the starting position of green pieces (3x3 location at the top left), and green player tries opposite.

3. Red pieces can only go in up and left directions, similarly, green pieces can only go down and right directions (there is an exception to this rule, explained in rule 6). Pieces can either go 1 square in their allowed directions, or they can jump over other pieces if the square behind jumped piece is empty. Jumping can continue recursively.

4. There is no capturing of pieces in this game.

5. Similar to chess, if a position occurs three times in the same game, game result is draw.

6. If player has no available moves with any of its pieces in forward direction, it can perform a backward move with one of its pieces in 3x3 goal location. There is no jumping in backward direction, piece can only move to adjacent empty square.

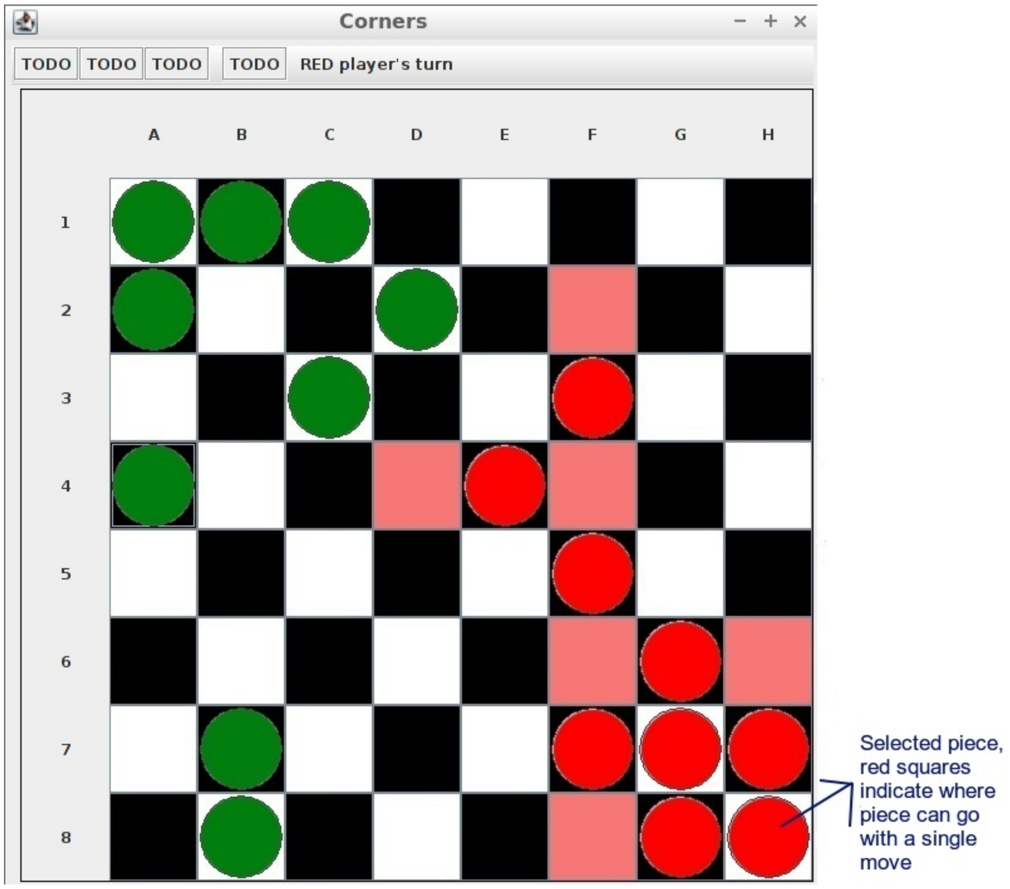
Corners
TODO
TODO
TODO
TODO
RED player's turn
A
B
C
D
E
F
G
H
1
2
3
4
5
6
7
8
Selected piece, red squares indicate where piece can go with a single move

Figure 2: An example image for rule 3, showing available moves for the red piece at the bottom right.

You will work on an existing Java implementation. If you want to work with another language you can always implement whole game from scratch and complete homework on it. You can get assignment code from github repository.

Problem 1 - Move Calculation for a Piece (10 points)

In the first part, you will implement the calculation of possible moves for a piece. In particular, you will implement *getNextMovesForward* function in *GameState.java*. Implementation of a similar function *getNextMovesBackward* is already provided in the same file, however it does not contain jumping moves (as explained in the rules).
After you successfully implemented *getNextMovesForward* function, you can start playing the game against yourself or the example random agent. You can change players from *initializePlayers* function in *Game.java*.
Aim of this part is to get you familiar with the code itself before working on the actual topic of the homework. Here is a list of files which are **completely irrelevant** for your homework (they mostly deal with GUI stuff):

- BoardCreator.java
- HumanPlayer.java
- PieceGUI.java
- PieceListener.java
- RoundButton.java
- SquareListener.java
- App.java

Problem 2 - Finding Two Heuristic Functions for Estimating State Utility (10 points)

In the second part, you need to come up with two different heuristic functions for evaluating utilities of states and implement them in the given code. These heuristics are expected to assign utilities for internal states without evaluating the exact MINIMAX values. You need to report the analysis of these heuristics.
Important note for implementation: Remember that in your implementation, you will assign some numerical values to utilities of winning, losing and drawing (terminal) states. The outputs of these heuristic functions should be correlated with the actual chances of winning. Heuristic function you design should always give values between winning and losing utilities (exclusive) for other states.

Problem 3 - Minimax Implementation (60 points)

In this main part of the homework, you will implement an agent that selects its move by performing minimax tree search. You will implement *getMove* function in *ComputerPlayer-Minimax.java*. You can look at *getMove* function in *ComputerPlayerRandom.java* for a very basic example. Your minimax agent should work for either of the following cases:

- Full minimax search, search until encountering a terminal state before computing utilities.
- Search until **depth 6**, and use heuristics to evaluate utitilies.

Report (20 points)

Write a report about what have you done in the assignment, give detailed explanations on the minimax algorithm implementation, the heuristics you design and a comparative analysis of them based on the computation time and performance, whether one dominates the other or not.

Submission

Submit your homework files through Ninova. Please zip and upload all your files using filename GAM507E_HW_2_STUDENTID.zip. Include whole **src** folder in your project directory in the submitted file and also your report as a pdf file. Your homework should be able to compile with Java version 8.