GAME THEORY - CHECKERS SCORING DESIGN

Checkers is played by two opponents. Each player has 12 checkers (of their color) on the black squares of the first three rows on the opposite ends of the board. The object of the game is to capture all of the opponent's checkers, or block them so they cannot be moved. The checkers are moved diagonally and each player moves alternately one of his checkers. In order to "capture" an opponent’s checker, your checker must jump over the opponent checker when there is a vacant square behind it. Single men may move diagonally forward. When a checker has reached the last row of his opponent's side, it becomes a "King" and can now move diagonally forward or backward. The "King" is "crowned" by placing another checker on top of it. When there is a "jump" available, the player must jump. You are allowed to jump as many of the opponent's men (with a single checker of yours) on the same move if there are vacant squares diagonally behind each. Rows are numbered from top to bottom and left to right (from 1 to 8)

0	1	2	3	4	5	6	7	8
1		W		W		W		W
2	W		W		W		W	
3		W		W		W		W
4	_		_		_		_	
5		_		_		_		_
6	B		B		B		B	
7		B		B		B		B
8	B		B		B		B	

To learn how or practice playing checkers, you can find free apps to play checkers against another person.

You should design various heuristics (at least 3) to "score the board" for a game theory program to play and win checkers. Basically, when shown a checkers game in progress (not yet an end game) and given a player color (black or white), you need ideas for how you can determine a relative score for the board for that player color that somehow predicts their chance of winning the game (positive score for good chance of winning and negative score for good chance of losing).

It may help to first answer these questions.

1. Do you think all pieces should have the same value towards winning?
2. Do you think all locations on the board should have the same value towards winning?
3. Are you going to change your strategy as the game progresses?
4. Are you going to change your strategy depending on who goes first?
5. The min-max algorithm only chooses a new move when the board score exceeds the current best and does not choose a new move if the board score ties the current best. Is your strategy going to include something so your computer does not always play the same move given the same board?

Then code various “score the board” methods and try them out with game theory code I provide below. Your ultimate goal is to pick one of your “score the board” methods to make the best checkers playing computer program to defeat your fellow CS100 students’ programs.

GAME THEORY - CHECKERS SCORING CODE

Code and test multiple heuristics to "score the board" that you designed. Submit the code for your best playing heuristic.

1. Download https://moss.cs.iit.edu/cs100/assign/CS100-CheckersFinal.zip Un-compress the files into a folder called CS100-CheckersFinal.
2. Open Eclipse. Click “File”… “Import”… from the Eclipse main menu. Expand “General”, select “Existing Projects into Workspace”, and click “Next”. In the “Select Root Directory” box, browse to your CS100-CheckersFinal folder, select the folder and click “Finish”. You should now see the new project in Package Explorer.
3. Expand the package, and “src”, and “submissions”. TestName.java is a shell that you can copy multiple times and code your different heuristics in. You should name each file (and class name in the file) with a descriptive word for the heuristic so when you compete each one against the heuristics we provide, you can keep track of the results and choose your best one for submission. The one you submit in lab next week must be named with first initial, last name of someone on your team (e.g. Bob Smith -> BSmith)
4. You need to do the following coding.
   a. In the constructor, update the name (this will be displayed when competing) and the section.
   b. The evaluateBoard method is where your code your board scoring heuristic. It must return an integer, positive values if good for you, negative values if bad for you. The relative size of the integer you return should reflect how good or bad the board is.
   c. Four boolean methods are available for you to call. They all take a single argument, a position (char) on the board (a 2D array of characters). See the board layout below and a sample method call in TestName.java.
      - ownChecker (char tile): own regular checker pieces
      - ownKing (char tile): own king checker pieces
      - oppChecker (char tile): opponent's regular checker pieces
      - oppKing (char tile): opponent's king checker pieces
   d. Compiling - The classpath file should be set up to have all the libraries and documentation included, so compiling should be automatically set up when opening the project in Eclipse.
   e. To run, a new run configuration must be added.
      1. At the top, click the small down arrow to the right of the green run button and choose “Run Configurations”
      2. Near the top left, click the “New launch configuration” icon. Name the configuration “CheckersPlay”
      3. In the “Main” tab, enter “checkers.Play” in the “Main class” text box
      4. In the “Arguments” tab, enter in the program arguments in the “Program arguments” tab. The only argument that is required is “-f <file> File submission testing, only include class name”. For example, “-f BSmith”. These arguments can be changed any time afterwards by the same steps as before.
      5. Click either 'Run' or 'Apply' and 'Close'
      6. Now to run, the green run button can just be used
   f. When running, you are given the option to compete the code you wrote (and chose with the –f argument) against 6 different heuristics we have provided. 1 and 2 are just return a random number. 3, 4, 5, 6 are progressively more complex heuristics, but they are not necessarily progressively more difficult to defeat. Four games total are played, with each opponent allowed to go first twice. Games are limited to 150 moves total, then a tie is called.

```
// the board, rows are numbered from top to bottom and left to right (from 1 to 8).
char [][] position
```

0	1	2	3	4	5	6	7	8
1		W		W		W		W
2	W		W		W		W	
3		W		W		W		W
4	_		_		_		_	
5		_		_		_		_
6	B		B		B		B	
7		B		B		B		B
8	B		B		B		B	

You should write your evaluateBoard method assuming the Evaluator's side to always be row 1 of the board, regardless of the color. When we swap colors we swap sides too

Arguments

Usage: java -cp <classpath> checkers.Play [-hV] [-d <num>] -f <file>

-h	Print this help message.
-V	Optional print out board after each move.
-d <num>	Optional number of depth to search for possible moves. Must be even. Default is 6
-f <file>	File submission testing, only include class name