CMSC 15100 Introduction to Computer Science I

Homework 13

You should submit a single .rkt file with your code via Canvas. The homework may require more debugging than usual, so it's recommended that you start sooner rather than later. At the top of the file for this assignment include a comment,

```
;; Homework 13
;;
;; your name
;; your CNET id
;; your collaborators
```

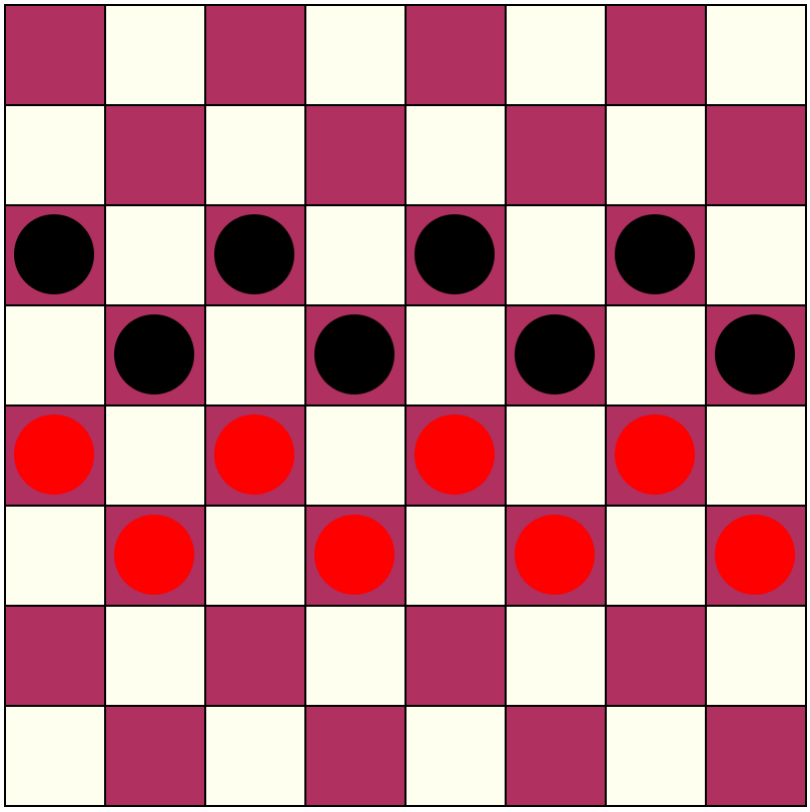
If your code doesn't compile, it may not be graded. Please double check to make sure your code runs before submitting!

This homework is optional. All problems are extra credit.

This homework is basically continuation of homework 9. You can work on it after you have finished homework 9. The solution to homework 9 will be posted on this Wednesday. If you wish you can use that code as starter code for this assignment. We will implement some of the nitpickier rules of checkers as well as some cosmetic improvements. They are listed in approximate order of difficulty of implementation. If you feel like tackling some of these, feel free to work out of order and do whichever you like – they are mostly independent.

1. (2 point extra credit) When a piece is clicked, highlight the possible locations this piece could move or jump to on the board. You can choose how to make the locations look different, for example by adding a small green circle to the square.

2. (2 point extra credit) It could be that both players have pieces on the board, but the current player has no possible moves! This could happen for example if all moves or jumps are blocked by other players:



When this happens, the rules state that the player who cannot make a move is the loser. Modify the function `game-over?` so that the game exits if the current player has no possible moves.

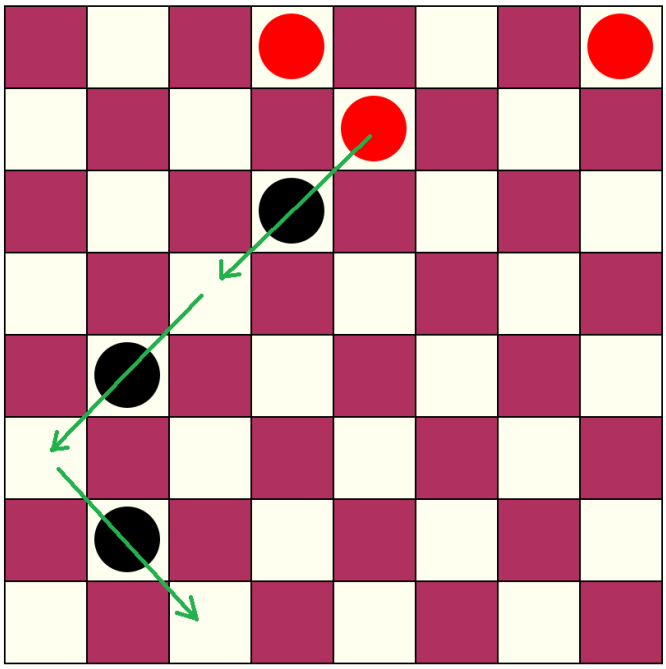
3. (1 point extra credit) When the game is over, make a fun screen that says which color won. To do this, you'll need to modify the `stop-when` clause in the `start-game` function at the end of your code. The clause should take two arguments: the first is the function to check if the game is over, and the second is a function to convert a game state to a final screen. See the universe.rkt documentation for more details, `https://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._world._%28%28lib._2htdp%2Funiverse..rkt%29._stop-when%29%29`

4. (3 points extra credit) The real rules of checkers require that if any of your pieces can make a jump over an opponent's piece, on this turn you must make a jump. Modify the `valid-move?` function to enforce this requirement.

5. (5 points extra credit) When a piece reaches the opponent's end of the board, there is a rule in checkers that says that piece is promoted to a "king". Kings have special privileges: they can move backwards (but they still must move diagonally like usual).

   Add functionality for kings. This will probably require adding a field to the `Piece` struct.

6. (7 points extra credit) After you jump over an opponent's piece, in fact you are allowed to continue jumping over opposing pieces for as long as you like. For example, the red player could follow all of the green arrows in a single move,



Implement multiple jumps as valid moves. When a player has clicked a piece, and then clicks on a desired final location for the piece, your code should search for a path of jumps that can reach that square. If you find such a path, you should move the piece along the path and remove all opposing pieces that are jumped along the way. If there is no such path, the board should not change, like the player clicked an invalid move.

It may be that there are two ways to reach a given square by jumps with a given piece. If the player opts for that square, you can remove pieces along any valid path.

This rule interacts a little strangely with the rule about required jumps. The way it works is this: if any of your pieces can jump another piece, your next move must be a jump, and furthermore you must continue to make jumps with that piece while it can jump over other pieces. Therefore, the only valid moves are now *maximal* jump sequences. You will get most of the credit if you allow pieces to make any jump or multiple jump; you will get full credit if you require them to be maximal jump sequences.