CMSC 15100 Introduction to Computer Science I

Homework 7

Your answers should be saved in a .rkt file and submitted via Canvas before the start of the next lecture. At the top of each file for this assignment and all future assignments include a comment,

```
;; Homework 7
;;
;; your name
;; your CNET id
```

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

Problems

1. (25 points) On this homework and on future homework assignments, we'll continue building our checkers game that we started in Lab 2. By the end of the course, you will have a playable game of checkers! The goal of this particular assignment is to make the pieces move.

   This problem builds on what was done in Lab 2, so if you made it through Lab 2 (clicking to add a piece), go ahead and use that code. If you didn't make it through the end of Lab 2, the file hw7-checkers.rkt is available on Canvas, which has most of the solutions to the lab. You can create a new directory for HW7 inside your CS151 directory and save the file there – submit this file with your homework handin.

   If you haven't done so already, in order to get familiar with the checkers datatypes, work through the "Playing Checkers" section of Lab 2 until you finish Task 8. This will also serve as a reminder to download cs151-universe.rkt and cs151-bigbang.rkt as these are necessary to make hw7-checkers.rkt compile. In the starter code, the `starting-board` variable is left for you to define as in Task 8. You should now a third argument to the `pieces-in-row` function, the color of the pieces.

   The tasks after Task 8 have been implemented for you in the handout code. Once you have `starting-board` set up and can click to add pieces to your board, proceed to the main tasks for this assignment.

   You don't need to write any tests for this homework. Test your code by running `start-game` and clicking around.

   (a) When you're playing a checkers game and you click on the board, it's necessary to remember the piece you clicked. Then, the next time you click the board, the piece will be moved to the new location. Add a field `clicked-piece` to the `Checkers` struct to keep track of a piece that was clicked on the last. The type of this field should be `(U 'none Piece)`. You don't have to do anything with this field for this part, just change the uses of `Checkers` so that the code compiles with the new field.

   (b) Now there are two functions to write: one that takes a grid location and moves a piece into the `clicked-piece` field, and a second that takes a grid location and moves a piece out of the `clicked-piece` field. Write a function `click-piece` that solves the first

task. The function should do two things to the `Checkers` input: first, it should set the `clicked-piece` field to the given piece; second, it should remove the given piece from the `pieces` list. The type of the function should be

```
(: click-piece : Piece Checkers -> Checkers)
```

You may need to write a helper function or two for this task.

(c) Write a function `place-piece` that takes a grid location and a game state and places the `clicked-piece` in that grid location. The `clicked-piece` field should be set back to `'none`, but the piece should be added to `pieces` with the new location. The type of the function should be

```
(: place-piece : Loc Checkers -> Checkers)
```

If the `clicked-piece` field is `'none` in the `Checkers` input, your code should raise an error.

(d) One more helper function is needed: a function `get-piece-or-none` which takes in a grid `Loc` and a `Checkers` and returns either a `Piece` if the game has a piece at that location, or `'none` if no piece exists. The type of this function is

```
(: get-piece-or-none : Loc Checkers -> (U 'none Piece))
```

(e) Using your functions from parts (b), (c), and (d) as helper functions, update the definition of `click-board` so that one of the two functions `click-piece` or `place-piece` is called when the mouse is clicked. Which one is clicked depends on whether `clicked-piece` is `'none` or not. Specifically, the behavior is:

- If `clicked-piece` is not `'none`, your code should call `place-piece`.
- If `clicked-piece` is `none` and `get-piece-or-none` returns a piece at the clicked location, your code should call `click-piece`.
- If `clicked-piece` is `none` and `get-piece-or-none` returns `'none` at the clicked location, your code should return the input `Checkers` board without modification. In English, your code should not modify board if you click on an empty space.

If you want to know more about the `click-board` function and how `Mouse-Event`s work, check out the Lab 2 handout or ask on Piazza.

(f) (1 point extra credit) After you update the `click-board` function, and if everything's working correctly, you'll notice that when you click a piece, it disappears from the board completely. Then, when you click the board again, it magically reappears. This deserves to look better! The reason this happens is the `draw-board` function wasn't updated to accomodate the new `clicked-piece` field. Change the function `draw-board` so that the piece doesn't disappear when clicked.

You should also make the clicked piece look visually different from other pieces on the board, for example by making the square it's in a different color.