Final Project - Interactive Sudoku

Due: July 23, 11:59 PM (CST)

> **Note 1:** The write-up was updated on July 14, 0:29 AM (CST). I moved the `get-sublist` function to Task 0, as it may be helpful for Task 1. I also add a new Image-Color `transparent` to the code we provided, which may be useful.

> **Note 2:** July 23 is the last day of this course, so it is not possible for me to grant any extension on this assignment except for extremely unusual cases. And you are also going to have your last lab assignment due on the same day. Although I will make sure the last lab assignment is very light-weight, please do plan ahead.

For your final project, we are going to create a 9*9 Sudoku that can accept users' input.

You may have implemented some of the features in your HW2, and you are welcome to reuse and modify your functions written in HW2.

Setup

Please create a new rkt file and put the following code in it. Save the file as `project/project.rkt`.

```
#lang typed/racket

(require "../include/cs151-core.rkt")
(require "../include/cs151-image.rkt")
(require "../include/cs151-universe.rkt")

(require typed/test-engine/racket-tests)

(define-type (Optional A) (U 'None (Some A)))

(define-struct (Some A)
  ([value : A]))

(define-type Sudoku (Listof Integer))

(define-struct SudokuWorld
  ([cell-size : Integer]
   [font-size : Byte]
   [grid-color : Image-Color]
   [bg-color : Image-Color]
   [clue-color : Image-Color]
   [user-color : Image-Color]
   [to-change-color : Image-Color]
   [puzzle : Sudoku]
   [user-answer : Sudoku]
   [waiting-input : (Optional Integer)]))

 (define-struct Click
```

```
  ([x : Integer]
   [y : Integer]))

;; a sample sudoku puzzle
;; you can use this for testing
(: sudoku1 : Sudoku)
(define sudoku1
  (list 0 0 6 0 0 8 5 0 0
        0 0 0 0 7 0 6 1 3
        0 0 0 0 0 0 0 0 9
        0 0 0 0 9 0 0 0 1
        0 0 1 0 0 0 8 0 0
        4 0 0 5 3 0 0 0 0
        1 0 7 0 5 3 0 0 0
        0 5 0 0 6 4 0 0 0
        3 0 0 1 0 0 0 6 0))

;; a sample SudokuWorld
(: sw1 : SudokuWorld)
(define sw1 (SudokuWorld 36
                         24
                         'black
                         'white
                         'black
                         'blue
                         'gray
                         sudoku1
                         (make-list 81 0)
                         'None))

; you may need to set some image's color to transparent
; and here is its definition
(: transparent : Image-Color)
(define transparent (color 0 0 0 0))

; ======= Task 0: general helper functions

(: get-sublist : All (A) (Listof A) Integer Integer -> (Listof A))

; ======= Task 1: draw a grid based on the given Sudoku, sizes, and colors

(: draw-grid : Sudoku Integer Byte Image-Color Image-Color Image-Color -> Image)

; ======= Task 2: more general helper functions

;; You are welcome to define more helper functions for all tasks.

(: editable? : SudokuWorld Integer -> Boolean)

(: clicked-within : Click SudokuWorld -> (Optional Integer))

(: update-cell-by-index : Sudoku Integer Integer -> Sudoku)

; ======= Task 3: universe
```

```
(: draw : SudokuWorld -> Image)
; draw a 9*9 grid, the index of each cell is from 0-80, starting from top-left
; corner. The numbering of index goes from left to right first, then from top
; to down.
; user's inputs and clicking effect should be correctly rendered as well
(define (draw world)
  (error "draw: todo"))


(: react-to-mouse : SudokuWorld Integer Integer Mouse-Event -> SudokuWorld)
; handle user input
(define (react-to-mouse world cx cy e)
  (error "react-to-mouse: todo"))


(: react-to-keyboard : SudokuWorld String -> SudokuWorld)
; depending on the pressed key, put a new number into the grid or not
(define (react-to-keyboard world key)
  (error "react-to-keyboard: todo"))


(: run : Integer Byte
   Image-Color Image-Color Image-Color Image-Color Image-Color
   Sudoku -> SudokuWorld)
; create a new world
(define (run cell-size font-size
             grid-c bg-c clue-c user-c to-change-c
             sudoku)
  (error "run: todo"))


(test)
```

Sudoku is a puzzle game where players place numbers into a 9*9 grid. Each Sudoku puzzle will have some clues --- some initial numbers that are placed in the grid.

Here is a real Sudoku puzzle example:

In this project, we will ask you to draw a Sudoku puzzle first. Next, you will use `universe` to create an interactive Sudoku puzzle. The interactive Sudoku puzzle will accept mouse events and keyboard events.

For all tasks, the validity of the Sudoku here will not be checked.

Let's go through each task one by one.

Task 0

Before we start, let's first define and implement the following function. It may be useful for later tasks.

```
(: get-sublist : All (A) (Listof A) Integer Integer -> (Listof A))
```

The function returns a sublist of the given list.

The arguments of the function are as follows (in order):

- an input list
- the starting index, indicates where the sublist starts in the original list. The index here is inclusive.
- the ending index, indicates where the sublist ends in the original list. The index here is exclusive.

For example, let's we have a list `(list 1 2 3 4 5)`.

```
(get-sublist (list 1 2 3 4 5) 2 4) --> (list 3 4)
(get-sublist (list 1 2 3 4 5) 0 3) --> (list 1 3)
```

You need to make sure the starting index and the ending index both are between `0` and `(length input-list)` inclusive. Otherwise, an error should be thrown out.

You also need to make sure that the starting index is equal to or less than the ending index. Otherwise, an error should be thrown out. When these two indexes are equal, an empty list should be returned.

Task 1

Let's use the above Sudoku puzzle as an example.

A Sudoku puzzle is essentially a 9*9 grid, which means it has 81 cells.

Let's assume each cell has an index from 0 to 80, counting from left to right first, then from top to bottom. Then, we can use a list of Integers to represent a Sudoku puzzle. Each cell `i` is represented by the `i`th item in the list (`i` starts from 0).

If cell `i` contains a number `a`, then the `i`th item in the list should be `a` as well. If cell `i` doesn't contain a number, then the `i`th item in the list should be `0`.

Based on the above method, we can convert the above image into a list of Integers, like the one we defined below (i.e., `sudoku1`).

```
(define-type Sudoku (Listof Integer))

(: sudoku1 : Sudoku)
(define sudoku1
  (list 0 0 6 0 0 8 5 0 0
        0 0 0 0 7 0 6 1 3
        0 0 0 0 0 0 0 0 9
        0 0 0 0 9 0 0 0 1
        0 0 1 0 0 0 8 0 0
        4 0 0 5 3 0 0 0 0
        1 0 7 0 5 3 0 0 0
        0 5 0 0 6 4 0 0 0
        3 0 0 1 0 0 0 6 0))
```

For clarity, we rename `(Listof Integer)` as Sudoku. You can safely assume a Sudoku will always be a list of 81 integers, and the integers are between 0-9 inclusive.

Now, let us define the following function.

```
(: draw-grid : Sudoku Integer Byte Image-Color Image-Color Image-Color -> Image)
```

The arguments this function has are as follows (in the same order):

- a Sudoku puzzle represented by a list of Integers
- the side length of each cell
- the font size (although it says it is a `Byte`, but you can treat it as an `Integer`)

- the color of the lines in the grid
- the color of the background
- the color of the numbers

For example, if we call `(draw-grid sudoku1 36 24 'black 'white 'black)` should output the following image.

You don't have to write `check-expect` for this task. Eyeball testing is enough. You also don't need to worry about cases where the number inside the cell will be bigger than the cell itself.

Task 2

Let's first go through what we would like to achieve eventually.

What we eventually would love to achieve is as follows:

1. Draw a grid with numbers based on the given Sudoku puzzle. These initial numbers in the puzzle are called clues.
2. A player needs to click on a cell to put a number down there. Once the cell is clicked, the cell's background color will change, indicating that it is waiting for input. However, those initial numbers in the puzzle should not be clickable (i.e., if one clicks on it, nothing would happen). Note, if a cell contains the player's own input, then the cell is clickable and they should be able to put a new number there.
3. Once a player clicked on a clickable cell, then they can use a keyboard to input the new number they would like to put there. The new number can only be numbers between 0-9 inclusive. If a player inputs `0`, it means they would like to remove the original number from that cell. Any other inputs will not have an impact, which means if the player press `A` after clicking on the cell, the cell will keep waiting for valid input.

You don't have to worry about the validity of the input.

Now, we will go through the `SudokuWorld` we defined earlier.

```
(define-struct SudokuWorld
  ([cell-size : Integer]
   [font-size : Byte]
   [grid-color : Image-Color]
   [bg-color : Image-Color]
   [clue-color : Image-Color]
   [user-color : Image-Color]
   [to-change-color : Image-Color]
   [puzzle : Sudoku]
   [user-answer : Sudoku]
   [waiting-input : (Optional Integer)]]))
```

- `cell-size`: the side length of each cell of the grid
- `font-size`: the font size of the numbers in the grid
- `grid-color`: the color of lines in the grid
- `bg-color`: the background color of the grid
- `clue-color`: the color of the initial numbers of the given Sudoku puzzle
- `user-color`: the default color of the numbers that are input by the user
- `to-change-color`: the background color of a cell once a player clicks on it, indicating that it is waiting for input. The background color of the cell will change back to `bg-color` once an input is received.
- `puzzle`: the Sudoku puzzle. It should be a list of 81 integers.
- `user-answer`: a list of 81 integers that records the player's inputs. It should be all-zero when a SudokuWorld is initiated.
- `waiting-input`: an optional integer. If it is `'None`, it means we are not waiting for input. If it is a `(Some Integer)`, the integer represents the index of the cell that is waiting for input.

Before we move on to the universe part of this project. There are several helper functions we would like you to implement first.

You are welcome to define more helper functions for any tasks given. Actually, you will and should define more helper functions to complete this project.

```
(: editable? : SudokuWorld Integer -> Boolean)
```

The function takes in a `SudokuWorld` and an integer. The function will decide whether the given cell (the input integer is the index of this cell) can be edited or not.

If the cell contains a clue (numbers from the Sudoku puzzle instead of user input), then that cell cannot be edited. Otherwise, it is editable.

```
(: clicked-within : Click SudokuWorld -> (Optional Integer))
```

The purpose of this function is similar to what you have in HW2, although the implementation would be different.

It tries to decide which cell is clicked. The output is an optional Integer.

If the clicked cell is not editable, then the function should return `'None`.

If the clicked cell is editable, then the function should return `(Some the-index-of-the-cell)`.

```
(: update-cell-by-index : Sudoku Integer Integer -> Sudoku)
```

The function here tries to update a Sudoku based on an index and a new value.

The arguments are as follows (in order):

- an input Sudoku (i.e., a list of 81 Integers)
- a cell index that indicates which cell we would like to update
- the new integer that would be put into the given cell. You can safely assume this integer is valid (i.e., between 0-9 inclusive)

All helper functions, as long as the output is not an image or a universe, should be well-tested.

Task 3

Now, let's move on to the universe part.

```
(: draw : SudokuWorld -> Image)
```

The `draw` function should correctly render a `SudokuWorld`, which contains both clues from the Sudoku puzzle and the player's inputs.

Here are some examples:
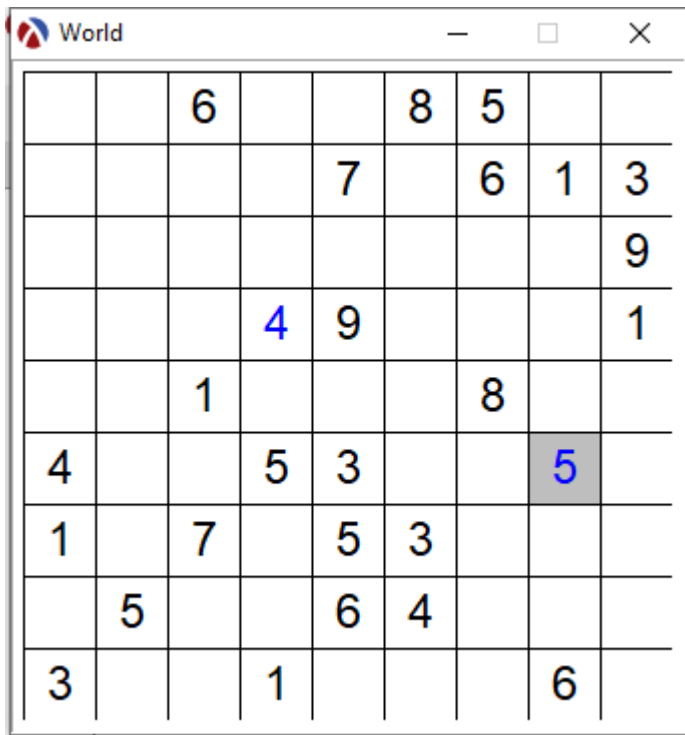
When there are no user inputs:

When clicking on an editable and blank cell:
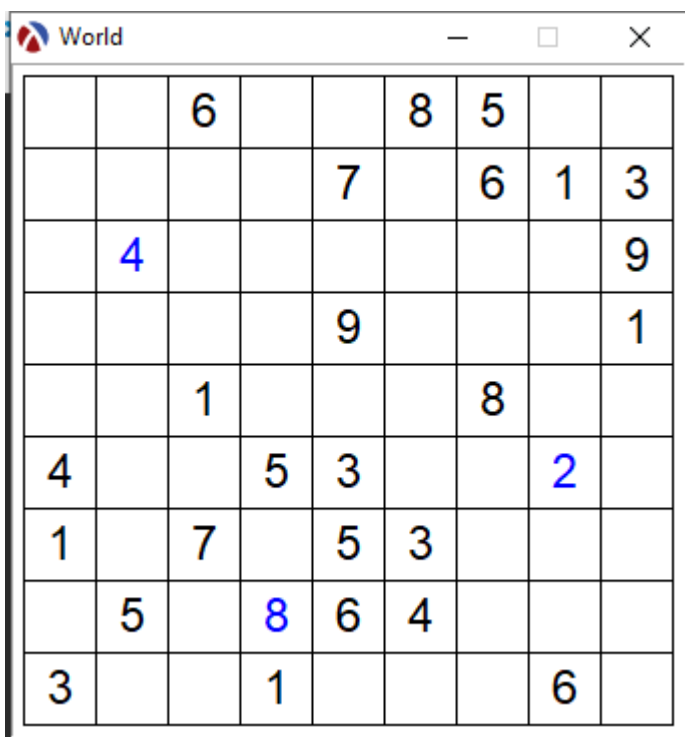
When clicking on an editable and non-blank cell:

When there are user inputs (the blue ones):


```
(: react-to-mouse : SudokuWorld Integer Integer Mouse-Event -> SudokuWorld)
```

If the player clicks on a cell that doesn't contain a clue, then the cell should turn into `to-change-color` and waiting for input.

If the player clicks on a cell that contains a clue, then nothing should happen.

```
(: react-to-keyboard : SudokuWorld String -> SudokuWorld)
```

If SudokuWorld is not waiting for an input, then no matter which key you press, the SudokuWorld should not change.

If SudokuWorld is waiting for an input, then

- press a number between 1-9, the number should show up on the cell that is waiting for an input
- press number 0, the original value in the waiting cell should be removed.
- press any other keys, nothing should happen. The cell will keep waiting for its input.

```
(: run : Integer Byte
   Image-Color Image-Color Image-Color Image-Color Image-Color
   Sudoku -> SudokuWorld)
```

Creating an initial universe based on the given configuration.

The arguments correspond to the fields of the SudokuWorld (in order):

- cell-size: the side length of each cell of the grid
- font-size: the font size of the numbers in the grid
- grid-color: the color of lines in the grid
- bg-color: the background color of the grid
- clue-color: the color of the initial numbers of the given Sudoku puzzle
- user-color: the default color of the numbers that are input by the user
- to-change-color: the background color of a cell once a player clicks on it, indicating that it is waiting for input. The background color of the cell will change back to bg-color once an input is received.
- puzzle: the Sudoku puzzle. It should be a list of 81 integers.

user-answer should be all-zero when the universe is initially created.

waiting-input should be 'None when the universe is initially created.

For all cases, the validity of the Sudoku is not considered.

That being said, you are welcome to perform the validity check for each user input you are given as an optional exercise.