Jakub Slowinski 16319781
Assignment 2: Minesweeper-o-matic

Small semi-formalish documentation of my thought process, code should be decently commented.

Setup: I used stack like in assignment 1 and had a very similar layout of files. This time around I used the web framework Scotty and created a small front-end made up of HTML, CSS and JavaScript which is at src/static/index.html. My lack of web development experience will be obvious. My main resides at app/Main.hs with the majority of functions in src/Lib.hs.

When doing stack build then stack exec assignment2-exe, the terminal will ask for the mine input and number of rows and cols.

You can start new random games from the website but not change these variables without shutting down scotty server.

The code should be decently commented, and some functions share similar layout such as openBox and flagBox so it will be a bit more obvious what is being tried to accomplish.

The functions in Main handling the requests will send JSON files to the website.

This JSON is then displayed in a table, and the old table deleted every time it receives a new one.

The website displays the number of flags left, which it gets from the JSON file. If you win a win banner is displayed and the board hidden.

We also obviously display our minesweeper board.

Things displayed on board:

{numberOnLeft : _} = index number, type this number in and you can flag or open this box.

X = a mine or unopened box containing a number(of adjacent mines)

[] = a box with zeroadjacent mines

(0<number<9) = opened box with number of adjacent mines

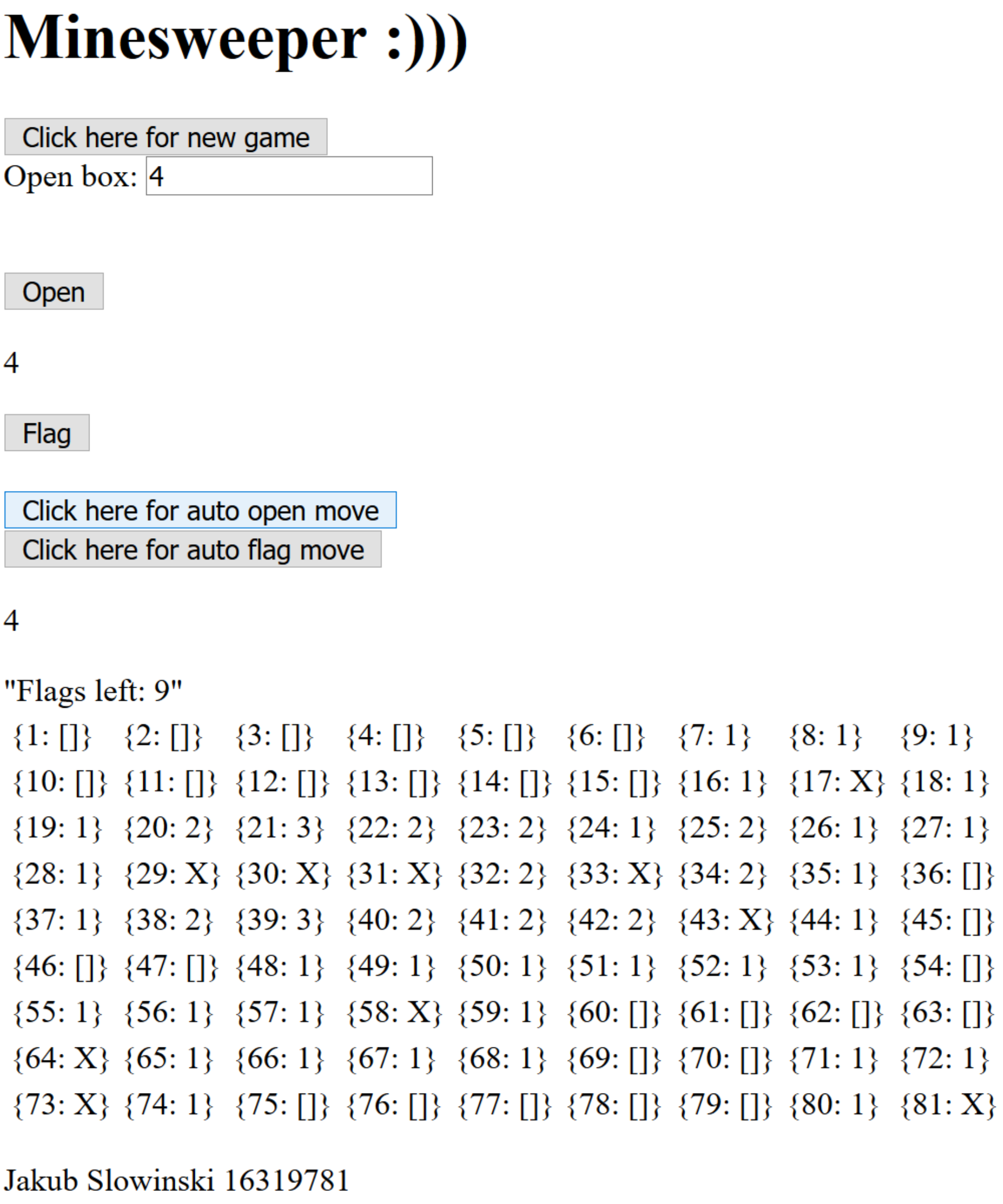
F = flagged box, can only flag unopened boxes

Scotty handles post requests: leftclick (open) and rightclick (flag) and the autoOpen and autoFlag moves.

UI: basic, but this is in html and JS so I didn’t spend much time on it.

Any number you type into the input box, you can either open or flag that indexed position. The random looking 4 below is just a confirmation of the requested index.

Final UI may have received a slight tweak.



Minesweeper :)))

Click here for new game

Open box: 4

Open

4

Flag

Click here for auto open move

Click here for auto flag move

4

```
"Flags left: 9"
{1: []}  {2: []}  {3: []}  {4: []}  {5: []}  {6: []}  {7: 1}   {8: 1}   {9: 1}
{10: []} {11: []} {12: []} {13: []} {14: []} {15: []} {16: 1}  {17: X}  {18: 1}
{19: 1}  {20: 2}  {21: 3}  {22: 2}  {23: 2}  {24: 1}  {25: 2}  {26: 1}  {27: 1}
{28: 1}  {29: X}  {30: X}  {31: X}  {32: 2}  {33: X}  {34: 2}  {35: 1}  {36: []}
{37: 1}  {38: 2}  {39: 3}  {40: 2}  {41: 2}  {42: 2}  {43: X}  {44: 1}  {45: []}
{46: []} {47: []} {48: 1}  {49: 1}  {50: 1}  {51: 1}  {52: 1}  {53: 1}  {54: []}
{55: 1}  {56: 1}  {57: 1}  {58: X}  {59: 1}  {60: []} {61: []} {62: []} {63: []}
{64: X}  {65: 1}  {66: 1}  {67: 1}  {68: 1}  {69: []} {70: []} {71: 1}  {72: 1}
{73: X}  {74: 1}  {75: []} {76: []} {77: []} {78: []} {79: []} {80: 1}  {81: X}
```

Jakub Slowinski 16319781

Minesweeper :)))

Click here for new game

Winner

Open box: 4

Open

4

Flag

Click here for auto open move

Click here for auto flag move

4

"Flags left: 7"

Jakub Slowinski 16319781

Algorithm in focus: countAdjacentBox

countAdjacentBox :: Int -> Int -> (Int, Int) -> Box -> [Box] -> Int

-- countAdjacentBox rows cols (xPos, yPos) box list

Returns the amount of adjacent boxes of type of box supplied in parameter, 0 if not adjacent to any boxes of type supplied.

It calls posChecker to check the position of the coordinates supplied and searches in all the directions that a box can be adjacent to.

This allows the program to see and reason about it, similarly like a human would.

When searching for a (No x) it will also count Mine as it is searching for all hidden boxes and this makes sure it doesn't discriminate.

Additional info:

Every new game is random by generating a new seed.

There might be 1 or 2 functions that are not in use, I left them in because they would be useful if I needed to use them when adding new functions.

Reason for not using bool anywhere?, mainly consistency after needing 1s and 0s in search function to determine the mine adjacency. If I redid it I would just have a function for bool addition, this would be safer than functions which return 0 or 1. Currently, a returning int that isn't 0 or 1(will never happen) would cause an unintended side effect. A lot of functions use 1 as true and 0 as false. A style I'm used to from languages like perl.

Autoflagging won't put a flag on the board for the sake of it. It checks for Hidden numbers adjacent to an open number and the flags adjacent. The reason flags isn't combined with hidden numbers is that if this is so it will try to unflag our correct flags at every move.

Autosolver is a good aid when there is many manual boxes open.

The autosolver could win some games not all, since minesweeper is NP complete, so if someone worked out how to solve all Minesweeper puzzles, all other NP complete problems would be also solved such as cracking public key cryptography and bank security.

What I would do different:

Possibility of a public and private board maybe? The private board will be fully known and analysed while public board is only the information opened. Only the public board will need to be passed into the html part. Currently the boxList sent is obscured in the JSON sent (eg. Both (No x) and Mine will be the symbol X).

Slight redundancy in algorithm in the mineChanceChecker which checks position and searches only in the way its certain its adjacent to. I used an algorithm in beginFixing which doesn't care about the position as it always check if its "out of bounds" with every call. The second way is quite easier to both code and reason with.

Not sold on data type especially Mine, I experimented with Mine being Hidden Mine. This would make my pattern matching less verbose in the auto solver but would mess with some calculations when making Box a number etc.

Monads for eliminating parameters that were being passed into every single function call (row, col) and (xPos, yPos)

If there was a state monad it would make the code more concise and allow for operations such as applying map in the function for opening emptys.

References:

https://jumboeats.wordpress.com/2014/11/10/haskell2048/

http://seanhess.github.io/2015/08/19/practical-haskell-json-api.html

https://stackoverflow.com/

https://en.wikibooks.org/wiki/Haskell/

http://adit.io/posts/2013-04-15-making-a-website-with-haskell.html