Minesweeper Project Design Update

1. User decides on a difficulty
    a. Beginner is a 9x9, 10 mines
    b. Intermediate is a 16x16, 40 mines
    c. Expert is a 30x16, 99 mines
2. User can input custom values (max is 35x35)

Vectors in Parallel

<table>
<tr><th>2D array of Button objects</th><th>2D array of ints (“A”)</th><th>2D array of ints (“B”)</th></tr>
<tr>
<td>
● Nested for loop
● Pointers of Button objects
● Made using nested for loop
● The position of each button is:
○ x is index*width
○ y is index*height
○ For squares, width = height
</td>
<td>
● Depending on difficulty, number of mines are randomly placed in the scope of dimensions of 2D array
● Mines will be assigned as the integer 9
● 1-8 will be clues
○ Count the adjacent 9s
○ Done by looping through the array (checking the location of [i+/-1][j+/-1]
○ Special cases for the ones on the edge
</td>
<td>
● Status of the tile
● Possible values:
<table>
<tr><td>0</td><td>open</td></tr>
<tr><td>1</td><td>closed</td></tr>
<tr><td>2</td><td>?</td></tr>
<tr><td>3</td><td>flag</td></tr>
</table>
● Initialize all values to 1 to indicate closed
● User action changes status of tile
</td>
</tr>
</table>

- *Function* using *FLTK’s Fl::event_button()* that tells which mouse button was pressed (returns *1* for *left click* and *3* for *right click)*
- *Function* taking in an int parameter that deals with the return value of *event_button()*
    - Action depends on values in 2D array “B”
    - Use of switch cases
    - For the event *right click:*
        - If the tile status is:
            - Open (0), nothing occurs
            - Closed (1), changes “B” value to flag (3)
            - Flagged (3), changes “B” value to question mark (2)
            - Question mark (4), changes “B” value to open (0)
            - This affects the button display, not the button’s state (up/down box)
            - Generate callback to change the images (flag/question mark)
    - For the event *left click:*
        - If the tile status is:
            - Flagged (3), nothing occurs
            - Open (0), nothing occurs
            - All else, turn “B” value to 0 and generate callback for that pointer of a Button
                - Sets value of Button to 1 (leaves it in a down box state)
    - For all cases that result in *opening* a Button, call the function that checks for mines (see below)
- *Functions* that return the indexes of the Button pressed as private helper functions

- *Function* that gets called everytime a tile is open via *left click*
    - Check 2D array “A” at the indexes [i][j]
        - If the value is:
            - 9, call the function that deals with losing (see below)
            - 0, open and recurse the adjacent tiles
                - This continues as long as the tile value of “A” = 0
            - 1, displays the clue value (value of “A” at those indexes)
                - This is the *base case* if there is recursion in the event that a tile with 0 as its “A” value was clicked upon
- *Function* that deals when a mine is left-clicked upon/losing
    - Redraws the board, perhaps implemented as a special instance in the display board function
    - Has its own set of images
    - Display the image of a mine on all indexes where 9 is its value in 2D array “A”
        - In cases that those indexes have 3 as their value in 2D array “B” (flagged), display the image of a mine X’d out
        - In the case of the indexes of the mine that triggered losing (the one that ultimately called this function), display a red mine
    - Nothing to wrongly flagged buttons
- *Function* that checks a winner
    - If the player opens all buttons that are not mined
        - This causes all the mines to automatically be flagged
        - Check if the number of closed is equal to the number of mines
            - This is only possible if the player has survived to this point of the game, no need to check location of the closed/mines
    - Even if the player successfully flags all the mines with no extra flags, must open the rest
    - Ends the game and allows the option to restart
- *Function* that counts how many mines vs. flags are on the board (mine number minus flag number)
    - Decreases by 1 with every flag placement
    - Increases by 1 with every question mark placement (when a flag is converted to a question mark, the only way to remove a flag)
- *Function* that displays the counter of mines
    - Displays as a widget; gets called in the function that displays the board (see below)
    - Generates callback
- *Function* that creates and displays a timer
    - Uses *Fl_Timer*
    - Generates callback
- *Function* that tells whether the game has been lost, playing, or won
    - Returns -1 for a loss, 0 for playing, 1 for winning
- *Function* that displays the game status
    - Uses *Fl_JPEG_Image* for the three different cases
    - Generates callback
- *Function* that displays the board
    - Loops through each Button pointer of the 2D array and generates the callbacks of each instance
        - Image associated with each button is decided via the values of vector “B” at the same indexes
    - This gets called/redrawn after every button press
    - Calls a function that checks winning if the player has not lost yet
    - Displays images for closed, flags, question marks, and opened

        - For *opened* cases, a grey box for “A”-valued 0 and numbers for 1-8 “A” values (clues)
    - A button in order to replay/restart the game
    - A widget that displays the mine count (calls the function that calls the callback)
    - A timer (calls the function that displays the timer)
    - A widget/image indicating the game status