The University of Kansas

EECS 448 - Project 1
Minesweeper

Miller Bath, Alex Wittman, Adam Wallace, Teddy Kahwaji, Bo Hui Lu

February 18th, 2019

Meeting Log:

Date (x = present)	Miller Bath	Adam Wallace	Alex Wittman	Allen Lu	Teddy Kahwaji	Location	Notes/Outcome
2/7/19	x	x	x	x	x	Spahr Library	Met to determine each group member's responsibilities for the project, created a deadline for initial versions, and discussed how front-end and logic code could interact.
2/11/19	x	x	x	x	x	448 - after class	Quick update on each member's progress thus far and what they might need assistance or help on.
2/11/19	x	x				Miller's House	Worked on a function to limit the number of bombs and discussed future front-end changes/additions.
2/11/19		x	x			Spahr Library	Determined how to attach functions to event listeners in JavaScript.
2/13/19	x	x	x	x	x	Spahr Library	Separated each teammate’s code from their testing setup, changed grid storage from integers to new Square objects, combined project elements on html Page branch.
2/17/19	x	x	x	x	x	Spahr Library	Final meeting to stress test, write documentation, and ensure completion of the project.

Work Split

1. Fill-Algorithm Branch (Teddy Kahwaji)
    a. This javascript source code integrated a conventional 2D array in order to simulate the game board. Moreover, it utilized the randomized math functionalities of javascript to randomize bomb placements within the area. In essence, this source-code’s purpose was to provide a key to the minesweeper game that would provide the html with the locations of both the empty and bomb spots.
2. HTML Page Branch (Miller Bath and Alex Wittman
    a. Alex and I worked together to tackle the user interface, user input handling, and overall aesthetic of the site. This also contained windows, notifications, and the structure of how the website would hold the minesweeper board. Using HTML

and CSS, we were able to create a general frame of how elements should be centered, aligned, or placed around the page.

3. Reveal-Algorithm (Bo Hui Lu)
    a. This javascript integration was critical for gameplay functionality. For instance, the source code utilized a game board and performed recursive functional calls to reveal spots based on user interaction. Moreover, the source code utilized classical javascript object-oriented functions for each square within the game board; each square contained specific attributes including a bool for bomb and flag placement, an assignment dictating whether a square was revealed or not, and a key.
4. Win Screen (Alex Wittman)
    a. The purpose of this branch was to create end game functions to call when the user either wins by placing all flags, or loses by clicking a bomb. Using HTML and CSS, I created a popup window for both winning and losing. These windows were styled in the like of a Windows 2000 popup. They each also contain an HTML button to create a new board and start a new game. The win and lose functions called when the game is over show one of these pop ups, so the user knows the game is over.
5. build_html_table and other GridFactory.js content (Adam Wallace)
    a. create_html_table was responsible for taking a 2D Array defined in JavaScript and turning it into an HTML Table with event listeners attached, also using JavaScript. In order to understand the underlying mechanics of higher-level javascript functions, it relies on document.createElement exclusively. GridFactory.js contains many other helper functions to simplify the implementation. In order to effectively handle assigning images to a key, a Key class is created that automatically returns images when passed a given key.
6. flag (Adam Wallace)
    a. This function is responsible for placing a flag on a right-clicked square. It will also update a global value to allow for the limiting of the number of flags.

Challenges

1. Fill-Algorithm (Teddy Kahwaji)
    a. The initial issue I had was ensuring that the randomized bomb placement functions correctly. For instance, the initial algorithm allowed for instances of a bomb being placed in an already existed bomb spot, resulting in less bombs than the user requested. This was immediately corrected in a later commit with the integration of a while loop checking if a bomb had already been placed within an

array indices. Additionally, ensuring that the nuisances of my code would be functional with the reveal algorithm because it utilized my code to build upon.

2. HTML Page (Alex Wittman and Miller Bath)
    a. Alex and I encountered multiple problems when dealing with innate javascript prompts, properly and consistently displaying information on the page, and unique quirks about HTML centering. In addition, each screen has different resolution and browsers for viewing the HTML, so mitigating those differences to be consistent had some trial and error. After playing around and getting quite familiar with divs, spans, and CSS Alex and I were able to create a consistent framework to hold all of the user interaction elements. Dealing with all the classes and IDs for each HTML tag was a rough start, but after becoming familiar with the file formatting and adjustment became easier over time.
3. Win Screen (Alex Wittman)
    a. The main problem I encountered while creating the win and lose popups was styling the popups to resemble a Windows 2000 popup. I had trouble centering the popup and keeping the text left aligned. I also had trouble adding the button to create a new board to start a new game. I ended up using the same button that is in the form to create a new board.
4. Reveal and flag algorithm (BoHui Lu)
    a. Before I was actually writing the algorithm down, I had to do a lot of reading and watching Javascript related content including syntax and different thing Javascript can do. Different ways to create a object confused me a lots at the beginning. But I figured out using constructor function would be easiest for our project. Since I could use the keyword prototype to create different functions for a object, then different objects that are created can share the function. Once I learned more about Javascript, I didn't think writing the code for the algorithm was that difficult. What I basically did was I used the logic I learned from C++, then I just translated it to Javascript. Lastly I had to do some code changes to let my code interacting with the HTML my groupmate created. Instead of using constructor function, class will be more clear and easy for my groupmate to read and interact with my code. But nothing was needed to change rather than the syntax.
5. GridFactory.js (Adam Wallace)
    a. Much of the work for creating the html table was straight-forward. What was difficult was doing it in a well organized, readable way. I decided early on to break the process down into multiple functions, one for creating the HTMLElement Table and populating it, another for applying CSS styling, and yet another for appending the Table as a child node. I had issues with searching for an 'id' to append to before the DOM had loaded, and while using MS Paint to edit

the icons wasn't necessarily a 'challenge,' I'd maybe prefer to use a more capable software next time.

6. Flag (Adam Wallace)
    a. Being a simple function, there weren't many challenges due to our Square.key attribute. However, it did have to be altered to set check Square.revealed state so that you cannot place a flag on a board when game hits its end state.

Demo Features (that didn't make the cut)

1. HTML Page (Miller Bath and Alex Wittman)
    a. We wanted to try and implement a functioning Windows taskbar from Windows 2000 that kept system time, had minimize, maximize, and adjustable windows sizes for the minesweeper window. It would have been cool to almost simulate a Windows 2000 experience like most of us remember as kids playing the same handful of preloaded Windows games over and over. This would have taken an immense amount of time and did not make it into even beginning stages of production. We also wanted to add a loader for our minesweeper grid so that the user does not try to spam the system with unneeded input.
2. Win Screen (Alex Wittman)
    a. I wanted to try and add the input form into each of the popups, so the user could create a new custom board size when they win or lose. However, I settled with just having a button that would create a board that had the same parameters as the previous game.

Retrospective (What we would have done differently)

1. Function compatibility
    a. A better idea of how HTML, CSS, and the other Javascript functions pieced together to create our website and the grid itself would have made our development process much smoother. This was everyone's first time, for the most part, to make sure everything played nicely with each other. It was a tough experience but worthwhile as we all learned a lot of valuable information pertaining to how you need to develop projects and ensure you have concrete idea of how the entire application will work.Throwing things together hoping it will work out in the end is never a good idea, and thankfully we didn't go about that route. We can use the experience gained for the next group project to better understand from the beginning how the handoff from function to function will work.