Mike Dwyer
FreshPlanet Code Test
Minesweeper

Embedded SWF: https://dl.dropboxusercontent.com/u/30240892/index.html

Time Breakdown:
15 minutes - Playing minesweeper to remember the rules
30 minutes - Download and install various SDKs (Flex, FlashDevelop, Starling)
45 minutes - Setup Sample Project to Run in Flash Develop
90 minutes - Playing around with Starling to learn it
360 minutes - Programming the game
15 minutes - Setup Github repository and test repo pulled from repository
30 minutes - This took a lot longer than I expected. I was having an interesting issue with the render mode that I think was FlashDevelop/Starling specific. I eventually just overrode it in the html.
20 minutes - Creating this documentation

Notes:
Use *ctrl+click* on a grid space to flag it. Flagged spaces cannot be revealed until unflagged.
Flagged spaces will have an “F” on them after being flagged
Mines will have an “M” on them after being clicked
Spaces with mines adjacent to them will have a boxed number after being clicked
Spaces with no mines will be empty and will flood fill after being clicked.

Interesting Decisions:
I chose to try to work with Starling for this test (as recommended), although I had never used it before. Overall I found it really fun to use and the learning curve was very quick. My starting point was using the Starling demo to set up a FlashDevelop project. I created a package under *src* called “*dwyer”* that I used for all my personal code. When I was done with the core gameplay, I removed all unreferenced demo code, but I left in the demo assets because I had used them to make the game look slightly cleaner.

I decided to represent the mine grid data as a 2 dimensional array of grid nodes for debug readability, rather than a 1 dimensional array with modified index access. If grid sizes were much larger, or we were really optimizing runtime process I would change this decision.

There are two obvious places to calculate mine adjacency numbers (when the scene is loaded and when spaces are clicked). Either of these could work, given that the mine placement is fixed throughout the difficulty session. I opted to calculate on click with the reasoning that reducing the level load time is a better choice with the given grid sizes. This makes the reveal moment less efficient if a flood is triggered, and if I noticed a slowdown on target platforms during this period I’d switch to calculating the grid adjacency numbers at initialization.

Further Work:
The interface is very rough and hard to read. It's also missing a flag counter and timer from the actual game. If this was going to be used as anything other than a judge of my coding architecture, I would improve on the usability of the game play.

I decided not to spend much time on setting up an ideal grid, as it was not something requested, but it was difficult to resist because it's an interesting design problem. I'm not sure what actual minesweeper does under the hood, but I'd imagine they try to make sure there are always empty spaces of certain sizes and probably calculate the mine placement based on your first click. They also probably give you a reasonable chance of getting a flood fill on your first click. Since I didn't do anything in this area, you can still click a mine as your first move.

I placed all the game data into a package called "designData". This is stuff I'd move to a server that would feed information to the client in a higher level production game. That way designers could tweak these numbers without users having to download any new files.