Retrospective Write-up

Meeting Log:

The first official meeting for project 2 took place on Wednesday the 6th of March directly after EECS 448 lecture from 9:00 to 9:50 am in a LEEP2 study alcove on floor 2 with all members present. We began the meeting by discussing the minesweeper project we received (Team Rocket’s project). Specifically, we dove into obvious things that were missing from the requirements of project 1 and minimal requirements to finish the current project. We wrote out a list that consisted of: adding adjustable rows and columns, adding a help button that would give instructions on how to play, making flags not return when revealing recursively, making the win conditions that all mines must be flagged and spaces revealed or when all non-mine spaces are revealed, implementing “cheat” mode that would reveal the board and be able to return to the game on a whim, and adding a feature that would be decided at a later date. We then discussed a few of the features we might want to add: adventure mode, RPG elements, a leveling system, and power-ups. Finally, we each agreed to open the project and look over the code. This proved more difficult than anticipated as we had no idea of how to open the .vb files included in the project. Upon leaving, each of us said our priority would be opening those files and figuring out how to edit them.

Later on the evening of the initial meeting all members took part in a video chat via Discord from 8:30 pm to about 10:30 pm. Note that this was not a constant meeting with all members, more of a flow of members coming and going as they saw fit. The goals of this meeting were to discuss working on the project from home, how to open the .vb files, and which parts each of us would work on. We ensured each team member had the same IDE (Visual Studio 2017 version 15.9.8) and we attempted to find the minimally needed installed add-ons in order to open the .vb files from visual studio. Due to the diligence of our team, we discovered from the official Microsoft website that the type of files discovered in the minesweeper project were Windows Forms. This discovery was made after noticing the files had a common “designer” file associated with them. This meant, according to Microsoft, we needed to use Visual Studio. With this discovery we added the recommended .NET development package and Visual Basic package to Visual Studio via the installer, created a new Windows Forms App, and added all of the .vb files to the project. With this, we were able to open the .vb files and edit them freely. We concluded the meeting here with much relief.

On 3/5/19 from 9:20 to 10:45 am in Spahr, a meeting of Jeff, Jon, and Thomas this time with a focus on planning what would be needed to communicate between the front end and back end of the project. The obvious elements like rows, columns, and mines were listed, however we didn’t want to alter the code heavily outside of refactoring unless our changes made sense so we decided to look at the code given to us. Promptly, we decided to move many more elements to the back end. This change on our part came after realizing that they had a random mix of elements scattered in the back end and front end that both accomplish the same task. In some cases, we noticed they had elements in a front end implementation when the structure for expansion was already in the back end. Thus, we decided to add cheat mode and guess position to the back end. The meeting then moved onto GUI interface and elements to be added. Upon finishing some white-boarded designs and playing with Visual Studio features, we concluded the meeting.

A smaller fourth meeting took place on 3/6/19 from 9:00 – 9:50 am with team members Jeff, Jon, and very briefly Benjamin. The primary topics of this meeting were finalizing communication between the front end and back end, as well as our feature. We agreed to discuss two different features: adventure mode and power-ups. Adventure mode would see difficulty increases as the player won the respective boards and power-ups would see the user gain special power-ups to use following wins on a respective board. These power-ups were to be:

1. Reveal an unrevealed cell adjacent to a mine
2. Reveal an unrevealed cell not adjacent to a mine
3. Flag a mine cell that hasn’t already been flagged

In the end, we decided on power-ups as they seemed both more challenging to implement and more fun. After getting approval for our feature, we concluded the meeting.

Later that evening, 3/6/19, from 8:40 pm to about 12:30 am Jeff and Jon held a coop coding session. For the first few hours of the meeting we used Discord chat in addition to a Visual Studio add-on called Live Share - Preview. The feature is currently unfinished hence the preview at the end of the name. With this Microsoft created feature we were able to start coding together on the same project. This was very neat and made coding as a team very functional as we would be editing the identical file. After some time we decided to work separately but remain in voice chat. Due to internet issues and the work progressing in different directions the separate approach became much easier. After fixing, adding onto, and generally making the quality much nicer for the front end of the minesweeper project in Visual Basic Jeff called the meeting due to needing rest.

The last meeting took place on 3/18/19 with all group members present, from 9:00 am to 9:50 am. During this meeting we discussed final documentation and finishing touches to the product and what would need to be ready to be demoed. It was decided that the program was solid and ready to go (it wasn’t without bugs that we didn’t notice however, more on that later) and the documentation left over would be finished before the documentation cut off.

Workload Split:

The workload split for this project was decided early and was followed through. Benjamin and Thomas would focus on the back end, Jeff and Jon would focus on the front end. The workload split was hardly debated at all but did revolve around individual members’ strengths. From influencing design of the code, UI, features to add, direction to go in, general debugging, and moral support; each member contributed and made the final product a success. Large portions of code were done in groups but with spring break upon us, the team adapted very well to dividing the work. The break was utilized to nearly complete the project with regular commits on github and regular reviews of the changes by other members.

Benjamin primarily focused his efforts on the back end of the code, the C++. This included refactoring the code, implementing board resizing, cheat mode, flags not moving on recursive reveal, as well as many quality of life changes. These changes made things much easier for the front end and made the project seemingly run much smoother overall. While that would be on going throughout the project other contributions included: bug fixing, testing, adding images, revealing the board upon a win or loss, quality of life additions, code refactoring, code clean up, and documentation.

Jeff, as previously, began by voluntarily logging all meetings and being the primary writer of the retrospective write up (so meta). Upon opening the GUI/front end elements of the project in Visual Basic, Jeff’s interests were piqued. Thus, his efforts were focused on this portion of the code. Specifically, changing around variables to work in the front end communication with the back end, implementation of the help button/GUI of the help screen for how to play, cheat mode through the front end, a reset function for starting a new game, changing the win and loss screens, and a play again button. Other duties included refactoring code, finding bugs or quirks in the game, testing the game, aid with design elements, quality of life additions, clean up, and even more documentation.

Jon got going from the outright by figuring out how to open the visual basic .vb files (this was a large effort that will be discussed further). After outlining the project needs in a fantastic readme.md, he succeeded in refactoring a 6000 line file in the project (from Team Rocket) into an under 150 line file by making clever use of a for loop. With these discoveries Jon’s interest in the front end was also piqued and he decided to focus his efforts there. Once again, Jon showed his strength with coding on the front end by refactoring much of what was in the code already in much simpler ways. In addition, he added the control box (GUI element that would take user input for rows, columns, mines, etc), changed the look of the board, put text instead of icons for the surrounding mine numbers, implemented power-ups, and implemented a way to communicate between the front end and back end. Of course, he aided in bug fixes, testing the code, and quality of life changes as well.

Thomas, at home with C++, decided to take on the back end immediately. This included bug fixes, code refactoring, and all around ease of use changes.

Challenges:

To say we had a tough go of things from the start would be an understatement. Within minutes of opening the minesweeper project from Team Rocket, we knew we had enough works for two projects. First, off when we opened the program we noticed a console application and .cpp files so we had a C++ project but also a GUI and .vb files indicated a Visual Basic application. Enter the first challenge of the project.

Upon importing the files to our IDE’s of choice every single one of us noticed a disturbing issue, either files were missing or we couldn’t open the files to see the code. After looking through our fork, the previous group did not choose to add many files to the .gitignore so it was unlikely they didn’t include the files. This meant we needed to find a way to open the files. This was a few days of frustration, lots of questions, and eventually the realization that we needed to install Visual Studio and add-ons for it. Now, we would expect this sort of thing to have been documented but alas our dig through the previous group’s documentation did not provide any insight. The only mention of the GUI interface came as what they said they used, VBA. Visual Basic for Applications, as we would discover it is called, is a type of project within Visual Studio and only Visual Studio as it seems. For a period of time, we thought we might not be able to complete the project at all but as stated we persevered. Finally we could open the GUI portion of the code and get a look at the Visual Basic within. This immediately led to the next challenge.

Upon first glance at the code, we suspected something must have gone wrong. A 6000+ line of code file doesn’t make any sense at all. However, after reading the previous group’s documentation, we soon realized that this was no mistake and in fact it was quite intentional. This was only one file though. A .designer.vb file to be exact. Further googling revealed that these are auto-generated files that are created only after the user declares elements on a Windows Form. Of course, we then opened up the Windows Form to see what exactly was going on that would require auto-generation of 6000+ lines of code. This contained not just one challenge but two! The windows form revealed that to construct their minesweeper board, they had created 100 individual buttons. Those 100 individual buttons were arranged into a 10 by 10 grid but each button had its own click events, features, attributes, and friend functions for access outside of its class. Not only did this mean we had a lot of reused code, we also had a product that was in no way extensible. This however, was not the second challenge. That was the fact that the board itself was not resizable. The user was stuck with a 10 by 10 grid, no option. This was not a minimally viable solution in any way. Overcoming these two challenges was simple enough luckily. The 100 reused buttons were placed inside a sub function and a nested for loop iterating over rows and columns provided a grid that could change size easily enough and turned 6000+ lines over two files into a single file under 150 lines of code. All that was required then was to make text boxes to ask the used for a number of rows and columns.

Mentioned above is the issue of realizing elements were missing from project 1. The non-adjustable rows and columns were discussed but other issues were present resulting in a non-minimally viable product. There were no instructions on how to play, minimal modularity, and plenty of bugs that needed squashed before we could even begin to think about requirements for project 2. Covering the how to play was simple enough after figuring out how to use Windows Forms properly. The large part was refactoring the code. The work was a simple but very tedious and not exclusive to either the C++ back-end or the Visual Basic front-end. Both sides had significant code reworks resulting in many parts of the project working without crashing and without significant slowdown. Thus, some of the bugs had been eliminated simply by refactoring the code however some others were still present. Mostly, these stemmed from the front-end and back-end communication since C++ was working too fast for Visual Basic. We fixed most of this by inserting intentional delays into the code.

Lastly, I have already mentioned multiple times that the rows and columns were not adjustable in the previous project. Indeed we did fix this issue in both the front end and back end. We made the rows and columns adjustable. This works by allowing the user to put the desired rows and columns into a text box. The value is then passed to the C++ backend. This is all well and good however, we caught an issue with the row value being passed to the backend. While patching memory leaks in the C++ code, a for loop was created that would delete an array of pointers. This garbage collection was necessary to fix issues that were inherited in the project. To keep things streamlined, we kept Team Rocket’s format of the pointers getting allocated memory and placed into an array. The problem is that the value for the upper limit of the for loop was set to m\_row\_size, which is by default 10. This means that any time the user requests more than 10 rows, the for loop goes out of bounds and thus crashes the C++ backend. This clearly causes the GUI to throw an exception and crash as well. This is a major bug that we failed to notice before the code freeze. This will haunt all of us for a long time and not being able to fix this is like having an itch we can never scratch.

Features that didn’t make the final product:

To be blunt, there isn’t much that didn’t go into the final product. Almost our entire time and effort was spent getting to a minimally viable product. That being said we did have some extra flavor additions that didn’t make it in as well as some of our own code that was refactored. This would be a good time to discuss what we cut from the previous project as well, outside of the code.

From the previous project, we changed all of the icons. Originally, they had numbers for each 1-8 representing the number of mines that would be around a space. Instead of using icons, we simple have the number display onto the cell. This solution is easier to read, seems to run better within the environment, and makes the game feel more like a traditional minesweeper. Additionally, we chose to change the color scheme of the product as a whole.

I have mentioned some minor quirks with Visual Basic previously, and we discussed adding some features specifically to help the stability of the front end. The front end does have a small issue but it would be hard to classify it as a bug. The communication errors I mentioned between the front end and the back end of the project can cause issues. Simply stated, clicking extremely quickly will tend to crash the front end due to communication being broken between the front end and the back end. This however is, in theory, possible to fix. If a queue was added to the code to hold click events from being registered within a certain time frame, the error could potentially be solved because, despite incessant click by the user, no clicks could be registered before the last click event was sent and processed by the back end. As stated, in theory this would have fixed this clicking issue but the possible bug was hard to reproduce and unless some *really* wanted to click that fast it shouldn’t be a problem. We decided against adding this queue to code due to the already seemingly fragile nature of VB, time constraints, the fact that it might not even be a bug, and the possibility it didn’t even work.

Known Bugs:

Clicking on different buttons very rapidly can send the wrong input to the C++ program since this project is really two programs that are trying to function on their own and communicate between one another.

Increasing the row size crashes the program. This issue was introduced by the code which solved the memory leaks, and this bug wasn’t discovered until after the code freeze. Inadequate testing on our part created this bug, and in the real world we would have to send out a hotfix patch to all program users, since the release deadline has already passed.

What we would have done differently:

Where do we begin? I think it was best said by Mike Tyson, "Everybody has a plan until they get punched in the mouth." Upon seeing this project, any plans we had vanished in an instant. We got hit hard by this project and honestly had a lot to do. Let’s have a discussion of things we did do and things we didn’t do, both of which most definitely should have been done differently.

To begin, little time could be devoted to additions. Perhaps, things could have been planned out differently and we could have split focus between team members further. We could have split out the back-end and front-end further by having one person focus on project 2 additions and one person focus on project 1 fixes. Judging by the major oversight in the rows bug, we should have done something differently that we didn’t do. This can be pinned down to nearly the same mistakes as in project 1. Not enough testing of the code, not enough of a serious approach to the scrum on a daily basis, not enough focus on agile development philosophy, and a lack of roles within the group. These lessons we’ve learned from class that we failed to put to use were our downfall and ultimately, assuming this really is a company, we failed to deliver the promised product as desired to the customer. So in the end, we didn’t learn from our mistakes or be proper software engineers. We did deliver a product but 95% of what the customer wanted isn’t a shippable product at all.

As things stand though, we couldn’t have done too many things differently. Development of the product led us down a fairly narrow path. We had to use the same languages we we’re given. This makes sense, in the real world this is how engineers will do their work. Honestly, it felt like programming inside a box but not a one of us would complain about that. This was a fantastic learning experience and a lesson in what makes a project high quality and what it takes to be a proper engineer. We can say we took the given product and expand it to something workable, flawed, but the experience gained and lessons learned will be invaluable.

Works Cited:

The visual basic code which creates an array of buttons is based off: <https://stackoverflow.com/questions/5497403/control-array-in-vb-net>