Retrospective Write-up

Meeting Log:

The first official meeting of the PySweeper team took place on Wednesday the 6th of February directly after EECS 448 lecture from 9:00 to 9:50 am in a LEEP2 study alcove on floor 2. The group members in attendance were Benjamin, Jeff, and Jon. We began the meeting by discussing minesweeper and the approach we should take to complete the project. We agreed that using python with pygame would be our approach for several reasons: our lack of experience with the language would make the project a great learning experience, the pygame module is famous for making games simple to code, and the language is widely used enough we could easily find sources for help should we run into issues. Moving forward we laid out a rough plan for what our python implementation would include. This process involved white boarding ideas until we can up with what we considered the simplest and most effective approach. Having familiarity with the game minesweeper, a board, cells on that board, and GUI were decided to be classes in addition to the main file that would control the game. The game logic was then outlined, followed by some of the GUI elements such as images for mines and flags. Breakdown for group members work from here was to be as followed: Benjamin works on game logic needed to implement minesweeper (general logic, not a python implementation yet), Jon would focus on python and pygame implementations including a starting a skeleton for the project, Jeff would work on documentation including logging meetings and begin work on the GUI including design elements, Thomas would begin work on other GUI elements such as the how to play instructions and showing the user they won or lost. Duties were notability left loose at this point; As a project evolves, a team members’ duty must always adapt to those ebbs and flows. Of course, each of us now needed to learn python so we agreed to implement some code in python before our next meeting as practice, Thomas was contacted via GroupMe chat. The next meeting was decided to be Thursday 2/7/19 and Friday 2/8/19. Due to the snow day, the Thursday meeting would not occur.

In the evening of the initial meeting Jeff, Jon, and Thomas started a video chat via Discord at 10:00 pm and conclude the meeting at 11:00 pm. The goals of this meeting were to discuss working on the project from home. We ensured each team member had the latest version of python (3.7.2), pygame (1.9.4), and, though to a lesser extent, git (2.20.1) on their system of choice. Previously, Jon pushed a text file with each members’ name spelled incorrectly. To ensure each of the team members had their systems working, each member was to pull and make a new commit that would correct their name during the meeting time. Each member was successful, and we were ready to get the project started in earnest.

Following a snow day and cancelled team meeting, PySweeper was eager to dig into implementing minesweeper with python. Friday the 8th of February from 9:00 am – 3:30 pm at the Fishbowl in Eaton Hall members Benjamin, Jeff, and Jon began the implementation for pysweeper.py based on our previous meeting and skeleton for the program. From 10:00 to 11:00 am Jeff was in class, from 11:00 am to 12:00 pm Benjamin was in class, and Jon worked on the project straight through from 9:00 am – 3:30 pm. To begin implementation, most of the coding, design choices, and testing were done together. This process began with deciding on a design for the board that would be displayed on the screen to the user. Once the board was displaying properly with a margin and the cell coordinates on the board were accurate, work on implementation was divided with Benjamin working on the cell and board classes, Jon and Jeff working on the GUI class and GUI elements. Benjamin completed most of the game logic for both the cell class and board class quickly. Once the mouseClick, drawboard, and uiElement definitions were defined in the GUI, Benjamin continued working on the board class by implementing revealCell, countNearbyMines, and flagCell. At least, we could see flags being placed and numbers being revealed around mines. This back and forth process of GUI to board went on until the board and cell classes were completed. Meanwhile, the GUI class and main pysweeper,py quickly evolved as their files grew larger, our design choices became more demanding, and our knowledge of how best to implement further elements because limited. Decisions on how best to continue were needed. We decided to add all the required parts first, get minesweeper working, debug, add on, and clean the code as needed. With all of our focus on the GUI now, we now had to come up with fields for the user to change the size of the board and number of mines. Previously, we had created space for a toolbar and placed text on the screen for the fields but creating text boxes to take user input and display it on the screen was proving difficult. We ended up finding, and heavily modifying, a class implementation from stack exchange (see works cited) for an input box that we aptly named the inputbox class. Upon getting the text fields working, we were able to implement custom sizing of the board and amount of mines according the project specifications. Some documentation and comments were added to the code, as well research into pygame functions to add images to the game for mines and flags. Near the end of the long session, the retrospective write up officially began, although the meetings had been logged from the start.

A semi-fourth meeting over chat took place on GroupMe from 10:00 to 10:30 am on Saturday the 9th of February with all members in attendance. Note that the GroupMe chat and Discord chats were on going throughout the project but the discussions were logged as a meeting if noteworthy progress or choices that impacted the design of the project were made. During this GroupMe chat session, we discussed if flags placed on none mines coming back to the player when cells are revealed is a bug or not. Eventually, it was decided to leave the program in its current state at least temporarily. We also added items to work on in a README.md file located on our github. Among these items were: adding images, refactoring code, changing where code executes, and what was left to work on to fulfill the requirements of the project.

On Monday February 11 from 9:00 – 9:50 am in a LEEP2 alcove on the 2nd floor, a meeting to finalize dividing up the remaining parts of the project was held. Thus, the results are simply what each team member was tasked with. Benjamin would implement any bug fixes, code refactoring, and comments on the board and cell classes. Jeff would continue work on documentation, pydoc (Comments that can be output to HTML file, work in command line), help as needed with implementation of the Game Over/Win prompts, update variables, find bugs, and aid in refactoring code as needed. Jon would primarily focus on reorienting, cleaning, refactoring, and otherwise working with the code as a whole. Thomas would implement the Game Over/Win prompts, play again/quit buttons, how to play instructions, and aid with documentation. All members would contribute with their thoughts and ideas as the project moved into the later stages. Members would each other if they had time and had done their tasks.

On Tuesday from 9:20 – 10:00 am in LEEP2 on the ground floor, Jeff and Thomas met after a class to work on the project together. During this meeting we went over how to comment our code properly and commented some of the code. At first this process was commenting for someone who may not have seen code in their lives but upon realizing that this is a software engineering class that approach was abandoned for higher level but still basic and clear commenting to explain why we took certain approaches. Ideas were shot around about how to best design the Game Over/Win screen and play again/quit buttons. Eventually, a design was chosen and we began working on a splash screen to display when a Game Over was reached. At first a Game Over state was either a win or a loss, simply a game has ended screen with no options. Implementation on clickable buttons for play again/quit were started but the meeting had to come to an end as we both had class.

On Tuesday the 13th of February during a discord video chat from 4:00 to 5:30 pm members Jon and Thomas had a meeting to work on some code for the project. The play again/quit buttons were finished and all input buttons were organize into an inputbutton subclass. Lastly, a minesweeper caption for the game window was added, previously this said pygame by default.

On February 14th in the EECS448 Lab classroom during regular lab hours from 12:30 – 2:15 pm with all members present, time was spent adding the last few features and finishing touches. During this time we focused on adding the how to play instructions. After much deliberation, it was decided an input button saying help would be the simplest solution that when clicked would display in text how to play and win the game. We also did some pydoc formatting but the large portion was bug checking and trying to fix the game. Some bug fixes and general quality of life improvments were implemented at this time but it should be noted that Jeff’s Linux environment still had a game breaking bug while the other members did not. After resizing the board, the user would not be allowed to provide input any longer. However after much searching on why, we decided to not pursue the matter further as no other members were able to replicate the issue. At the end of the meeting, the minesweeper code was complete as per the project guidelines. The only portions left to complete were the documentation portions and refactoring of the code for ease of use.

A deep dive into beautifying the code took place late on the evening of February 14th. The main goal here was to refactor code, make things look better, and see if there were unused items that could be eliminated. In the end, much of the code was easier to follow and some elements that we thought we needed were cut as they were not needed at all.

On February 15th in the Fishbowl at Eaton Hall from 8:15 – 9:50 am with snow pouring down outside, Benjamin and Jeff advanced the pydoc portion of the comments. There were some hurdles with this process, more on that later, but overall the final product turned out satisfactory and professional. This was the last meeting and was one of the final portions of the project to complete. The pydoc wasn’t quite finished the meeting concluded due to Jeff having class at 10:00 am. The pydoc portion was later finished that afternoon, as not part of a meeting. This would conclude the required portion of the code and documentation, save for the write up you are reading at this moment, the project could be considered finish. Any further advancements would be additions, not requirements.

Workload Split:

The workload split started out simple and for the most part that didn’t change. However, as the project evolved and, more importantly, as we learned more about each other as a group as the workload split evolved accordingly. This will be touched on more on later. In the end, the final workload split was based on available time, scheduling, and individual members’ strengths. Before going into specific portions that each team member agreed to work on, I want to note that every member had a hand in nearly every portion of the project. From influencing design of the code, UI, features to add, or direction to go in; everyone had a hand in every process. Many parts were done as a group, indeed most of the coding was done together rather than individually in an effort to make the project a more cohesive final product. Of course, parts were done by specific members but for the most part everyone used their strengths to make this project the best it could be.

Benjamin started the project dialed in on the game logic for minesweeper: what the code would be required to do and how to implement that into python. Soon this evolved into aid with implementing other classes to the code. Specifically, these portions of the code included, Board class, cell class, the GUI class, InputBox class, and the main file pysweeper.py. While that would be on going throughout the project other contributions included: bug fixing primarily within the game logic and input field logic, aid with design elements, aid with GUI elements, aid with text elements, quality of life additions, code refactoring, code clean up, and documentation. A sizable portion of the pydoc, HTML file generation for comments, was contributed by Benjamin.

Jeff began by voluntarily logging all meetings and, by virtue, being the primary writer of the retrospective write up. Initially, the GUI class was enormous and asking one person to take on the entire class would have been unfair. Thus, it was agreed that Jeff would aid Jon in implementing said GUI class. This took the specific form of drawing boxes, mouse events, and displaying text. With documentation and the GUI ongoing as the project moved forward, other duties became adding in and implementing images, aiding with splash screens, aiding with the main file pysweeper.py, finding bugs or quirks in the game, testing the game, aid with design elements, quality of life additions, code refactoring, clean up, and even more documentation. Outside of the listed duties, a smaller duty included contacting instructor Paul Kline or TA Oqi as needed.

Jon jumped right into the code from day 1. After outlining the code for the project with a skeleton, he implemented several functions into the pysweeper.py main file such as drawing the cells on the board and working on the main game loop, how the game continues to run until the user ends the game. Recognizing that the GUI would be a significant portion of the code, he began working on how to best implement the needed features. This included drawing objects onto the screen, mouse events, input boxes, UI elements, and input buttons. Once again while this was ongoing, it became very clear Jon was at home with the code. Thus, duties evolved into fixing bugs, refactoring code (a whole lot of this), organizing the code, many implementations for quality of life, aiding with the help input buttons, aid with how to play instructions, aid with some of the documentation, and testing the code. Outside of simply aiding in the project, Jon has a lot of experience with git, bash, visual studio code, and features of many of the interfaces that were used. Being able to ask him any question about these interfaces and get a very precise answer was helpful.

Thomas started by working primarily on design elements of the project. This included coming up with a quit, restart, and help button, as well as a game over, winner, and a how to play splash screen. Other duties included: aiding with the input button class, finding bugs, refactoring code, cleaning up code, and testing the code. A large portion of the ideas for things to add to the project came from the mind of Thomas, many of these were creative elements that otherwise would not have been implemented. Without these contributions to style, the game wouldn’t have the feel of mine sweeper.

Challenges:

The Pysweeper project was a fun and rewarding experience to everyone involved, however the project wasn’t without its hurdles. Most clear and present, is the fact that we are all used to working alone on our own code. As with any team work in any facet of life, working with others requires adjustments. At the forefront, was splitting the workload evenly and fairly. On top of this project being smaller in scope than expected making dividing the workload among four people difficult, we are used to being able to jump around to whatever portion of the code we wish and changing things on the fly. When the workload must be divide four ways, that isn’t so simple to do without stepping into another members’ work. Additionally, scheduling played a direct role. Portions of the code needed to be implemented for another member to move forward. To remedy the division of the workload and scheduling, we simply agreed to primarily set aside time to work together and pound out as much as we could in portions of time. From here we had a meeting to divvy out work but left open that as time available, we would aid each other and have open communication of who was working on what. Thus the work division was loose but this worked well for a group of hard working mature students. Tied directly too this is humbling ourselves and making compromises. For the most part, the group was very fair about this. If another team mate requested something then it would be discussed, if the work became uneven it would be brought up, and most members were open with their thoughts. That didn’t always mean we agreed on everything with implementation, design elements, and in what direction we should go but that was okay because we all agreed to focus on the requirements for the project first and help each other out. So, while the problem was there for portions of the project, in the end, focus and a common goal kept things flowing properly. In addition to these issues, snow days and meeting as a full group in general was difficult. Some of these problems can’t be helped, however again open communication was key. Keeping everyone updated was mainly the best for overcoming this issue. As stated earlier this group overcame the loss of time with their maturity and focus.

Aside from general working as a group issues that likely everyone faces during certain periods, the actual code became an issue or rather, the lack of meticulous planning before beginning to code caused the code to become an issue. Now don’t get that wrong, each member of the team understood exactly how important planning out the project would be which is why we spent the entire first meeting planning out the project and further discussing the plan in a chat. We had a nice outline and even a skeleton to build our code off of. Where we slipped up is when the program began to evolve, we did not adjust the plan or try to come up with a new one as meticulously as we did the first. Our agile development approach resulted in large chunks of repeated and inefficient code. This consequently left us investing significant amounts of time in to refactoring code, cleaning up the code, and all around trying to make the code readable. There was no real overcoming this issue. Unfortunately, we had to simply invest the time needed to get the project in a state that was presentable. All this refactoring and changing of code, that did already work, led to further challenges since we would commonly need to solve new problems. At times we even introduced bugs that would require more time investment to patch up. Often times though, we simply didn’t notice a problem until we were further along. One issue that came up a few times was involved with the algorithm used to reveal a cell. Since the algorithm is recursive and gets called 8 times for each cell it reaches, a very large board that contains a lot of cells will result in very large amounts of recursion. Hence, we ended up with a too much recursion error. This specific bug was squashed by limiting the size of the board, therefore the cells on the board, to 30 by 30. After some research, it was determined that a 32 by 32 board with only 1 mine would cause the error but any smaller would not. A 30 by 30 board was large enough to be larger than any board a normal minesweeper would allow, thus we decided to limit the user to 30 by 30 board max size.

The last main section of issues is a simple one and involves the pygame module itself. Simply put, pygame can be confusing. The module’s functions aren’t always exactly clear and resources for how to use functions for specific purposes are very limited thus most of the time to implement a feature using a pygame function we were left to our own devices. This resulted in few issues with actual implementation, indeed pygame was quite friendly in that regard, but it did result in a whole lot of trial and error. Often elements on the screen were out of position and adjustments had to be made by adjusting the element’s position in pixels. In some instances where we draw elements on top of others, we had no idea that the user could still interact with the element below the newly draw surface. Mostly, it was difficult to understand when we needed to call certain functions from the pygame module and when they were unneeded. After learning more about pygame and python in general, we were able to decipher a lot of elements that were never being used and thus could be cut from the game, but it is likely some unneeded elements are remaining in the code. A smaller issue was with the pydoc module built into python 3. The module would not allow us to include pygame in the program. Try as we might we couldn’t solve the issue, so we simply worked around it by taking out the import pygame portion of the code then generating HTML files before adding it back in.

Features that didn’t make the final product:

At a surface standpoint, minesweeper is a rather simple project to churn out. Most people with programming experience can comfortably recreate its main features using the language of their choice. Again, this may seem minimal but that’s the joy of minesweeper. For some the game is simple, but for others the game has a level of competitive depth that other games don’t measure up to. Keeping this in mind as PySweeper took shape, there wasn’t too much we left out. We wanted a minesweeper experience, not just a game. However, there is one unique feature to our game and key features of minesweeper that were indeed left out.

Unique to our project, as we progressed we made the choice to move in different directions. One of these design choices was in the form of a semi-bug that we kept in. Originally, in the toolbar displayed at the top of our minesweeper game had a white box drawn around it keeping everything nice and neat. As is the natural order of things, the project evolved to encompass more elements inside the toolbar. While there wasn’t a whole lot of clutter inside this box, it was upon seeing the toolbar with no box that it was decided the game looked cleaner. This was especially obvious on larger size boards where the toolbar was much smaller than the board itself. Thus, we decided to leave the toolbar without this box. How this “bug” happened in the first place was due to us refactoring the code to make it simpler. We moved everything to be drawn in the tool bar from being a uiElement method within the GUI class to being a subclass of the GUI class. When moving the code to this class, we failed to notice that we had not drawn the box. Upon realizing the box was missing, instead of fixing it, we eliminated the old code that was calling the box to be drawn.

Features of other minesweeper games that were notably, and disappointingly, left out were the use of the middle mouse button and difficulty options preset for the user. We decided to leave out the middle mouse button, used to show the user which cells the active cell is adjacent to, for several reasons. For one, it wasn’t required but merely a quality of life addition. Secondly, this would have required time we should be focusing on making our currently project a more quality product. Mostly however, we understood that the project should be easily extensible. An easily extensible product leaves room for improvement and a middle mouse feature is a clear way to extend this project. For the same reasons we did not add difficulties for the user. This was discussed and would have seen our project have easy, medium, and hard buttons that would set the rows, columns, and mines for the user according to what other versions of minesweeper have implemented. Having the user define the rows, columns, and mines is sensible although not traditionally in minesweeper to our knowledge.

Somethings that we chose not to implement were “fixes” for ambiguous issues that either we weren’t sure needed changed or weren’t clearly defined in the project guidelines. Three features of debate were present with the flags we give to the user. The flags may be placed in any cell at any time however our reveal cell algorithm ignores them if they are not on a mine. This results in a flag or flags returning the user if a flagged cell is revealed and not a mine. This feature is how the google minesweeper works but the Microsoft Windows minesweeper leaves the flag in place no matter what the cell is. The Windows version is clearly more difficult but, after much debate, we left the game alone. If it isn’t broken, don’t fix it. We didn’t set out to create a difficult minesweeper, just a minesweeper that works. The next ambiguous issue was that after placing a flag, the flag could still be clicked and the cell revealed. After speaking with instructor Kline about the behavior, we decided that our game didn’t need to hold the players’ hand. Thus, the “issue” went unchanged. Finally, something that wasn’t verbose in the project guidelines was the win conditions. In our version of minesweeper, the user must flag all mines to win the game. This is exactly in accordance project guideline and thus went unchanged. Under no circumstances does the user win unless all mines are flagged. However, in the original minesweeper the user would win if all cells are revealed that are not mines as well.

What we would have done differently:

Ask the group for further talking points?

As Jay-Z once famously rhymed, “Time waits for no man, can't turn back the hands. Once it's too late, gotta learn to live with regrets.” Powerful, relatable words for PySweeper indeed. The minesweeper project started innocently enough but soon became an amalgamous blob of code. This was covered in the challenges section, as were all of the things we would have done differently, but the lack of adjustments in planning must be reiterated. How much time could have been saved if we had simply taken a couple hours to meticulously plan out every class, definition, and feature? We will never know but I do know we would have had a lot less bugs and an easier to follow final product with greater ease. Sure, we ended up with the same result but that doesn’t give us our time back. This would have to tie directly into another regret, we should have broken up the coding into more than one giant session. The long session on Friday the 8th of February was very productive and quite rewarding but once the GUI and main pysweeper.py files started to go beyond the scope of our outline we should have stopped, gone back to the white board, and planned out the rest with the same tenacity we had on the day of the first meeting. This wasn’t simple as we were incredibly eager to continue coding and see our progress. We made so much of that progress! The cost of the progress followed us through the rest of the project.

To conclude, a project is only as good as the sum of its parts. Those parts are intertwined and inseparable, even if you’d like it to be otherwise. Such is the group involved with the project. Our group worked as a unit and honestly, we were all delightfully surprised that we were able to work quickly and finish the project early and with more features than envisioned. We spent a lot of time together seeing that vision become a reality. Along the way, we’ve learned more about each other, our strengths and desires as computer scientists. These imprints are all over the project. Learning about each other--our individual visions for the project and where we are strongest, would have made the project’s finish smoother. But this all fits in with planning. We could have taken the time to communicate in the planning phase and made our dream a reality much sooner. The big take away is that as engineers we will be expected to work efficiently, not just get the job done. Get the job done and get it done quickly. Going forward, we will take the needed steps to plan ahead and when things don’t go according to plan, come up with a new plan. Overall, the regrets are minimal for this group and upon demoing the final product, it is easy to see why.

Works Cited:

InputBox class code inspired by: <https://stackoverflow.com/questions/46390231/how-to-create-a-text-input-box-with-pygame/46390412>

General game logic influenced by Daniel Shiffman’s video: <https://www.youtube.com/watch?v=LFU5ZlrR21E&vl=en>