MINESWEEPER AND SWIFTUI – MY PROCESS OF BECOMING A BETTER DEVELOPER

by

ANNA FINLAY

A THESIS

Presented to the Department of Computer Science
and the Robert D. Clark Honors College
in partial fulfillment of the requirements for the degree of
Bachelor of Science

May 2025

An Abstract of the Thesis of

Anna Finlay for the degree of Bachelor of Science
in the Department of Computer Science to be taken June 2025

Title: Minesweeper and SwiftUI – My Process of Becoming a Better Developer

Approved: _____*Juan Flores, Ph.D.*_____
Primary Thesis Advisor

This thesis is the compliment to a Minesweeper app that was developed in Swift, over the course of 5 months. It details my journey of learning a new programming language, designing and developing a fully functional app, and reflecting on the technical and personal growth experienced throughout the process. By documenting each stage of development, from initial concepts and design challenges to final implementation, this thesis explores how engaging in a creatively and technically demanding project can contribute meaningfully to one's development as a software engineer.

By analyzing both successes and setbacks, and defending the decisions made along the way, this work aims to answer the central question: *How did this project make me a better developer?*

Acknowledgements

I would like to express my deepest gratitude to my former professor, Dr. Flores, for your guidance and unwavering belief in me. You played a pivotal role in my success at the University of Oregon and are the reason I chose to major in Computer Science. Thank you for giving me the opportunity to work as a Learning Assistant in the Winter of 2022. That experience truly changed the trajectory of my academic journey.

I am also deeply thankful to Dr. Faye. Your encouragement and genuine interest in both my academic and personal growth have been instrumental in keeping me grounded throughout my education. Thank you for your role as my academic advisor, and now as my Thesis mentor.

Additionally, I am forever grateful to Dr. Childs, who supervised classroom worktime and provided the space I needed to succeed. Your support allowed me to fully immerse myself in this personal project, and gave me lots of support, motivation, and belief in myself.

Thank you!

Table of Contents

List of Figures

Introduction: Why Minesweeper?

A Brief History and Personal Interest

Minesweeper is a popular online game that has been played for decades, first implemented in 1990 by Microsoft[1]. Amongst other old school web-games like Snake, Tetris, and Solitaire, its simple design has stood the test of time, and it still has a competitive worldwide following today. Minesweeper relies upon the players' numerical deduction and pattern recognition skills and is often played for the goal of either clearing the board, or doing so in the least amount of time possible. World records on standardized settings for expert level boards have been continuously broken in recent years, with one of the most recent records being set by China's Ze-En Ju in 2022.[2] On the standard board of 16 by 30 tiles, with 99 bombs (over 20% bomb density) he managed to clear every safe tile in a mere 26.6 seconds. In fact, just last year he managed to beat his own record, with a time of 25.1 seconds[3], which undergoing verification by Minesweeper World Records. [2]

I myself have sunk a couple hundred hours into various websites that host the game. I first made my account on Minesweeper.Online[4] in 2021, but also sometimes play on other websites.[5] The mechanics, strategy, and mathematical component of the game has kept me hooked for many years. Thus began a long personal project to try to re-create this game on iOS.

This project has fulfilled two personal goals. First, it allowed me to learn Swift, a language that was entirely new to me. This not only strengthened my technical skills as a developer by learning a new language, but it also gave me the opportunity to manage large-scale projects and systems. Second, it gave me a way to explore the "no-guess" element of Minesweeper, which is the aspect of the game I find most complex and compelling.

The “No Guess” Setting

A typical game of Minesweeper begins with a grid of hidden tiles. When a player clicks on a tile, it "flips" to reveal either a number or a bomb. If a number is revealed, it indicates how many of the eight surrounding tiles (adjacent, via orthogonal and diagonal connection) contain bombs. If the player “flips” a bomb, the game is lost, and he must restart.

Hidden among the unflipped tiles is a fixed number of bombs, randomly placed and unknown to the player. The objective is to uncover all non-bomb tiles while optionally flagging those that do contain bombs.

“No Guess” is a setting of gameplay that has already been implemented on some Minesweeper platforms.[4][6] Notably, this also includes Simon Tatham’s (open source) implementation of “Mines” under an MIT license.[7] Under this setting, players should strictly need to rely upon logic in order to solve the puzzle. That means no pattern will ever yield a 50/50 guess and perfect gameplay will result in a 100% win rate. Sound easy? The reader is encouraged to check out the “Evil No-Guess” board on Minesweeper.Online[4]. This, of course means that mine configurations are limited. Clusters amongst the edge of the board, specifically in corners, are especially limited, and cannot be generated randomly.

My App Design

Implementation choices across platforms. Some “No Guess” settings begin with a “starting tile” already selected for the player,[4] [6] whereas some do not restrict the first move in this way.[7] Additionally, Nicholas Emmett’s website[5] doesn’t allow ‘chording,’ (a player clicks on an already flipped tile to clear it’s neighbors,) while most other websites do.

Another variation involves the behavior of the first tile clicked. Some apps[4] [6] [7] always guarantee a “pocket” opening, meaning the first clicked tile is a 0 (indicating no adjacent

bombs). This ensures that at least nine tiles (the clicked tile and its neighbors) are revealed immediately, giving the player a playable opening. In contrast, other platforms[5] may return a number such as 1 or 2 on the first click, which can lead to scenarios where no safe follow-up move is available.

For my app, I chose to implement a “No Guess” approach as the default setting. The app does not restrict the first move to a specific tile, supports chording, and guarantees a pocket opening.

The most interesting part of this project was implementing the “No Guess” mechanics. I wanted to approach this in the most inspiring way possible, which led me to target the “No Guess” problem by building a solver for the game itself. The process involves generating a random board and testing whether the solver can complete it. If the solver fails, the system discards that board and generates a new one. This approach allowed me to avoid restricting the user to a predetermined start tile, since board generation occurs dynamically on click. Additionally, it would permit natural expansion of the game into a simple “help” button. If a player gets stuck, the computer is already equipped with a solver which will be able to point out areas in which deductions can still be made.

Limitations

Naturally, this implementation comes with several limitations, which are discussed below. The first major challenge is bomb density. If the density of bombs on the board is too high, it becomes increasingly more difficult to generate a valid “No Guess” board. For example, if 90% of the tiles are bombs, the likelihood of creating a solvable board without requiring any guessing becomes extremely low. Therefore, the solver was written with the stipulation that user restrictions must be put in place in order to prevent timeout issues.

Another limitation is the lack of efficiency in the board generation process. Since the solver attempts random boards and checks each one for solvability, it is not the most optimized method. This inefficiency becomes even more problematic because generation happens on-click, meaning the user must wait for the system to produce a solvable board in real time. In situations with high bomb density or limited computing resources, this delay could noticeably affect user experience.

Despite these limitations, this is still the approach I chose to take. Not only does it support a lot of the functionality that I was hoping to implement, but it also is the most fun and thought provoking method, and kind of why I decided on this project in the first place.

Chapter 2: Design process

What Makes This Novel?

While the idea of creating a Minesweeper app is far from novel, given the game's long history and the many existing implementations, I believe my project offers a few unique contributions.

Most notably, this app includes an integrated solver. Although many standalone Minesweeper solvers exist, it's rare to see them built directly into a playable Minesweeper platform. Even more uncommon is the way I've implemented the "No Guess" feature. In this approach, mine generation occurs only after the first user click. This not only allows the user complete freedom in selecting a starting tile, but also enhances fairness by avoiding early guesswork. While such a method may raise concerns about potential lag, at the board sizes and mine densities I've chosen, performance remains smooth and unaffected.

To my knowledge, there are very few—if any—similar implementations written in Swift. For these reasons, I believe this project makes a meaningful contribution to the broader landscape of Minesweeper development.

Preliminary Design

The implementation of the app started in early January, with my work environment set up by Jaunary 17th of 2025. The first month was an exercise in learning the new software and language. Goals for the end of the second week were items such as "make a responsive button," and "make the button a variety of colors." By the end of week 3, I had produced some Software Requirements Specifications (SRS), as well as the Software Design Specifications (SDS).[8] These two documents produced the backbone and structure of what I would set out to build over

the next couple months. The SRS is a written out series of requirements that the project should meet, or aim to meet. Found on on this list are technical specifications, as well as requirements for user experience and gameplay mechanics. Here are some examples of the requirements generated.

> "1. There should be a board, made up of tiles. Each tile will either be a bomb, or a safe space, but should not be visible to the player before uncovering…
>
> 10. Standard gameplay mode should… have a ratio of mines per tile that is less than or equal to hard (20.63%) yet more challenging than easy (12.35%)…
>
> 21. Bombs should randomly distribute around the board, with no obvious or predictable patterns to their population, and indeed be the number of bombs intended every time.
>
> 22. You should not be able to flag an already explored tile."

Although not every element on the list was implemented exactly as originally planned, each one was ultimately addressed—either in the manner outlined in the document or through an alternative, novel approach.

Accompanying this list of requirements is the SDS, which is in diagram format. Although originally hand written, it has been digitized. Both of these documents were produced on the 20th of January and are included to demonstrate the ways that I changed as a developer, and checkpoint my process in the journey.

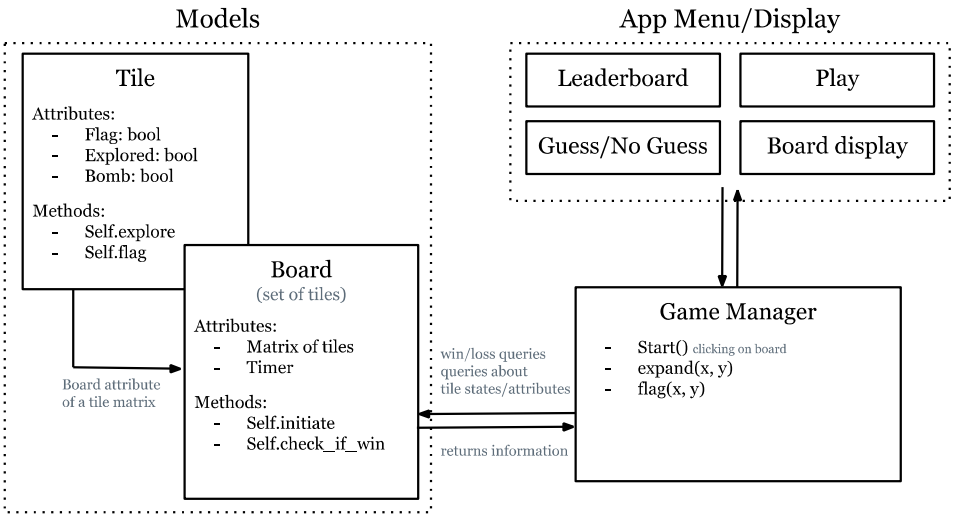


Figure 1: "**Preliminary System Design Specifications"**

This diagram was originally created on January 22, 2025, and digitized on May 15, 2025. It illustrates the initial planning and layout for how the backend would interact with the frontend, outlining the necessary models and methods of communication between components. Key elements include core structures such as *Tile*, *Board*, and *GameManager*, as well as UI components like the app menu buttons—*Play*, *Guess/No Guess*, and others.

By the end of January, I already had already checked off half of the first requirement, having implemented a grid of button tiles. By the first week of February, I had implemented a very rudimentary version of the game. The most interesting part about this was the explore() function, which had been implemented recursively.

```
165     func explore(row: Int, col: Int) {
166
167         if buttons[row][col].bool == true {
168             buttons[row][col].color = Color.red;
169             buttons[row][col].text = "b";
170             gameCease = true
171         } else {
172             buttons[row][col].color = Color.green
173             if (buttons[row][col].int == 0){
174                 buttons[row][col].text = "0"
175                 for adj_r in row-1..<row+2 {
176                     for adj_c in col-1..<col+2 {
177                         if adj_r >= 0 && adj_c >= 0 {
178                             if (adj_r < num_rows && adj_c < num_cols){
179                                 if(buttons[adj_r][adj_c].color == Colo
180                                     explore(row:adj_r, col:adj_c)
181                                 }
182                             }
183                         }
                    }
```

Figure 2: "**Recursive Implementation"**

In the basic implementation of Minesweeper, the *explore* function needed to be called recursively on all tiles with zero neighboring bombs.

In the basic implementation of Minesweeper, the *explore* function is typically implemented recursively to efficiently reveal all connected tiles with zero neighboring bombs. When a player clicks on a tile with no adjacent bombs, the game should automatically reveal all surrounding tiles that also have zero bomb neighbors, continuing this process outward until the edges of the "safe" area are reached.

Building the Solver

The solver that I built can be broken down into various distinct stages. The algorithm that I will be talking about is similar to one described by Games Computers Play[9] with a couple subtle differences. The simplest level of this solver can be thought of as a dynamic programming[10] problem, and then the rest is constraint propagation on sub-groups of tiles. It is

also based heavily on my experience with writing a Sudoku Solver. The algorithm is written as follows.

The solver begins by performing an initial *exploration* from a chosen starting tile. This recursive step uncovers all adjacent tiles with zero neighboring bombs, expanding outward until the edge of the safe region is reached. Once this area has been revealed, the program enters its first solving phase. Here, it looks for tiles that could only possibly be bombs. It will iterate through revealed numbers (e.g., a tile marked with a “2” or "3") and counts how many unopened, unflagged tiles surround it. If that number matches the tile’s value, the solver can safely mark all of those adjacent tiles as bombs by placing flags. (In sudoku solving, this is similar to what are called “naked singles.”)[11]

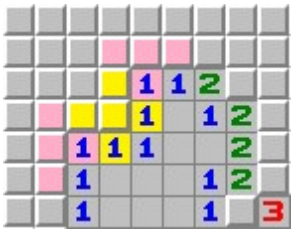
After flagging, the solver re-scans the board to check whether any numbered tiles are now fully satisfied in terms of adjacent flags. If they are, any remaining unclicked neighbors must be safe and can be explored recursively. This loop of flagging and exploring continues until no further progress is made. At that point, the program escalates to the second phase, *find_grouping_constraints*.

In *find_grouping_constraints*, the solver identifies all numbered tiles with a small number of adjacent unopened tiles and groups them into sets. It then compares these sets to each other—if one set is fully contained within another, the constraints differ by a known amount, thus it can deduce the nature of the remaining tiles (either bombs or safe spaces). This form of subset-based reasoning helps the solver break through more complex situations where surface-level logic no longer applies.

This logic helps with placing minimums and maximums on the numbers of bombs that can be found per region. For example, *find_grouping_constraints* can solve some of these familiar patterns produced in Minesweeper, such as the figures below.

Figures 3: "**1-1-X"**

Figure sourced from minesweepergame.com[2].



Figures 4: "**1-1-Corner"**

Figure sourced from minesweepergame.com[2].

Figures 5: "**2-2-X"**

Figure sourced from minesweepergame.com[2].

Once the solver has completed this more complex deduction, it will switch back to the first series of clearing empty cells, flagging naked singles, and so on. This will continue until the program either stagnates, having made no progress in 1 full succession of both types of solving algorithm, or terminates due to a solved board.

Final Design

After much time spent developing, the final system design ended up looking something more like this.

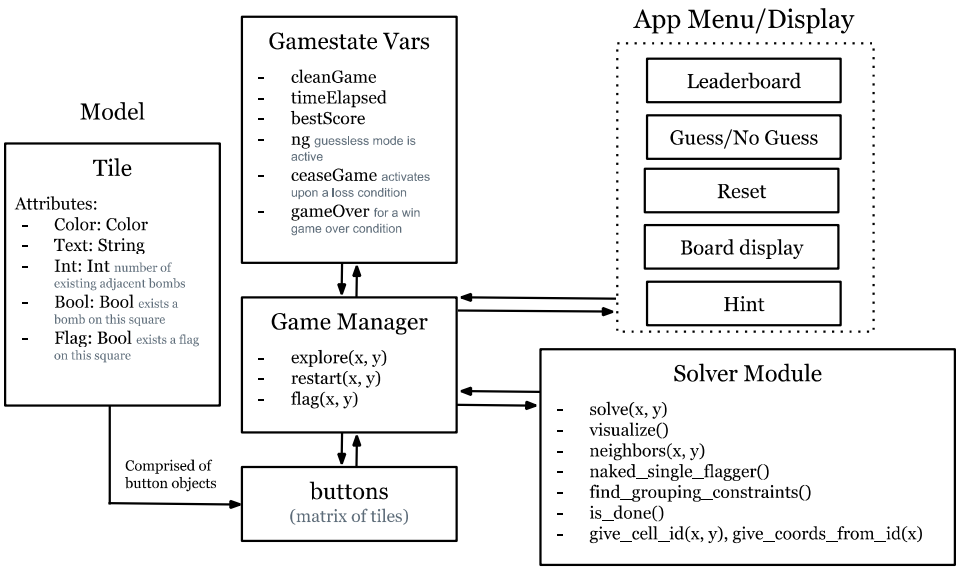


Figure 6: "**Final System Design Specifications"**

This diagram was originally created on May 20th, 2025. Contrasting against the preliminary software design, you can see that the solving software was broken out into it's own module, and many more global variables were introduced. Additionally, the board object was scrapped, in favor of a buttons array (comprised of tiles,) which was mutated through game management functions.

Some changes occurred, but overall, there is a very similar structure to what was originally sought out. The primary changes were obviously due to the additions of a solver module and bonus global variables that weren't thought of initially. There was also the transition from a Board object to a matrix of tiles called *buttons*, which were mutated through game management functions.

Additionally, there was a *Hint* button added. After having already implemented the solving logic, this button simply calls the already written solving module, (accessed through

Game Management,) and the game highlights which tiles have a mathematical deduction that can be made with a lightbulb.

The hint button increments the player time by 10 seconds, strictly if there was a hint that indeed given. (Some players may choose not to play on “No Guess” mode.) All of these buttons had to be added to the final UI.

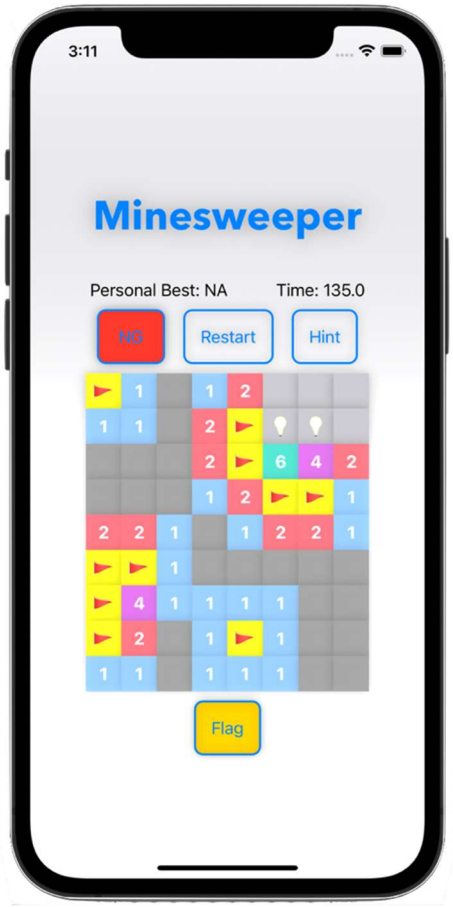


Figure 7: “**Final UI”**

Screenshot originally captured on May 21st of 2025. This image shows the final working product of the game, highlighting the key features implemented, visual choices (including colors selected,) and final layout of buttons.

This final user interface stays true to the classic Minesweeper colors. This is a mapping from blue to one, two to red, three to green, and so forth. However, the colors for the numbers and the buttons beneath them are inverted. (Traditionally, the numbers are color coordinated by value

while the buttons are all gray.) This makes my product distinct and fun, and has created an enjoyable play experience.

Chapter 3: Reflection

Setbacks

The journey of developing my Minesweeper solver was filled with challenges, some expected and others unforeseen. One of the major setbacks I encountered was switching from Python to Swift for the logic of the solver. (Python's flexibility and fewer type restrictions allowed for faster development.) However, switching back to Swift after perfecting the logic proved to be a more difficult an experience than I anticipated. Swift's stricter type system made it challenging to directly translate my Python code, requiring me to somewhat rework large portions of the project. This transition doubled the amount of work and significantly slowed progress, forcing me to learn the nuances of a new language while also attempting to keep the project on track.

Another significant hurdle came very early in the process when I struggled with setting up my development workspace. I found that I couldn't proceed effectively without a solid foundation, and this led to a lot of frustration. However, this setback reminded me of the importance of seeking help. I leaned on the expertise of my peers and teachers and was able to overcome these initial obstacles and gain a deeper understanding of the technical aspects of my work.

Swift's error messages also posed another considerable challenge. Debugging was difficult when errors would cause crashes or strange behaviors without providing enough feedback to pinpoint the issue. Before learning about the output console that is built into Xcode, it felt as though I was coding in the dark, unable to understand the root causes of the problems I was encountering. It took me quite a while to find my footing in the new software, and especially to be able to run many iterations of the Swift program repeatedly, without re-starting the App

and UI altogether every time. My inexperience in navigating these challenges, led to long sessions without clear solutions. Looking back, I now realize the importance of setting up an efficient development environment early on in a project. This will be one of my priorities in future endeavors.

One of the most frustrating technical challenges was dynamically scoping the button array to adjust to different sizes based on user input. This issue persisted for weeks, with the program crashing unexpectedly due to a bug. Despite my best efforts, I couldn't resolve the problem in time, which forced me to rethink the project's design. I ultimately reworked the requirements to remove the dynamic button sizing from the user's control. In the end, this change not only resolved the issue but also solved some of the technical requirements by eliminating the need to worry about bomb density in the matrix. Pre-set dimension settings can regulate this potential for the program timing-out.

Finally, the sheer volume of the project itself was a source of significant stress. This was the largest-scale project I had ever undertaken, and the pressure of managing its complexity often felt overwhelming. There were moments when the scale of the task, both in terms of its technical demands and the time constraints, seemed insurmountable. These moments of struggle only made the eventual completion of the project all the more rewarding.

Takeaways

While this project presented numerous setbacks and challenges, the lessons I learned throughout the process were invaluable. I learned perseverance, especially when facing technical difficulties. For example, I preserved through the difficulty of switching from Python to Swift, and also through the self-doubt of taking on a project of this scale. I learned to pivot when something wasn't working, and I was reminded of the value of asking for help. I'm deeply

grateful for the guidance I received from my peers and teachers, which helped me move past difficult roadblocks and continue my work.

The biggest takeaway from this experience, however, was the importance of resilience. This project tested my problem-solving abilities, patience, and mental fortitude more than any other academic task I've faced. It taught me that the process of working through setbacks, whether it's reworking code or adjusting my approach to the problem altogether, was just as important as the final result. The satisfaction of overcoming these hurdles and seeing the project come to fruition made the journey worthwhile. This project has not only improved my technical skills but has also strengthened my ability to manage large and complex tasks. (Both in terms of the software, but also emotionally.) While the experience was often daunting, I have ultimately been left with a rewarding collection of memories and an immense amount of pride.

Conclusion

Overall, this project sought to answer the question, how will the process of developing this app make me a better developer? That question has been answered in a couple of ways. Obviously, I have acquired new hard skills in the language of Swift, iOS development through the IDE of Xcode, including the syntax, quirks, and some of the drawbacks of such a development environment. These technical skills will hopefully aid me in my future, effectively expanding my toolkit as an engineer. Secondarily, my findings revealed personal character weaknesses, as well as weaknesses in strategy and wisdom. I have learned a great deal about deadline management, personal motivation strategies, and what the achievement of my goals can looks like. This was the second way in which my research question was answered. I am better for this project because I learned how to overcome my own doubts and lack of discipline. Finally, I have learned a lot about what Minesweeper is as a game, and what the math is surrounding it.

Learning about the solving algorithms was one thing, but implementing them was much more challenging than their conceptualization. Breaking down such algorithms into concrete pieces, actualizing them, and debugging them, has given me a very thorough understanding and appreciation for the magic of the game of Minesweeper.

There are certainly areas for improvement upon my game. This can include additions to the solving algorithm, like taking bomb count as a factor with which to solve the board. (Deducing if the given number of bombs on the board restricts the potential remaining configuration possibilities.) This is something that is already included on other online platforms, but was not an area of investigation for me thus far. Another item that I would consider adding to the app would be things that were considered at the beginning of the project but did not come to fruition. That includes elements like a button board which is dynamically sized, and can re-generate at different dimensions based on customizable settings. Again, this was cut from the feature set due to lack of alignment with the goals and areas of interest of investigation. Regardless, its inclusion would certainly create a more robust user experience, and bring a more competitive edge to the App.

The app, if intended to be played on competitively, should also invest more resources into the testing of the timing infrastructure. It is possible that the timer can be broken, bugged, or otherwise tampered with, which would discredit the times produced by users. That being said, there are many other platforms which are already accredited, and used as the ‘standard’ for minesweeper times, so it is unlikely that new technologies and platforms are going to be of use in that regard.

Overall, there were a couple potential errors of the process itself. This includes periods of weeks in which the project was not worked on at all. This made me forget where I was, and in a

sense, develop a fear of returning. This was exacerbated by a lack of comments throughout my code, and I sometimes found myself hunting through the code, trying to re-remember how it worked. This was mostly due to the fact that the app was developed over a full 5 months. In the future, I will put more emphasis on appropriate variable naming, and code commenting.

This project was limited by many human factors, but ultimately, was an exercise in growth. My sense of self, and understanding of what I can achieve has expanded. Ultimately, the pride I take to have contributed something to the minesweeper community will forever hold a very special place in my heart.

Bibliography

[1] R. Cobbett, “The most successful game ever: A history of minesweeper,” TechRadar, https://www.techradar.com/news/gaming/the-most-successful-game-ever-a-history-of-minesweeper-596504 (accessed May 14, 2025).

[2] Minesweeper World Records, https://minesweepergame.com/world-records.php (accessed Feb. 10, 2025).

[3] kiraa96, “Minesweeper Expert World Record in 25.10 by Ze-en Ju (JZE),” YouTube, https://www.youtube.com/watch?v=idP6pai3gD8 (accessed May 14, 2025).

[4] “Minesweeper.Online,” Minesweeper.Online, https://minesweeper.online/ (accessed November 1st, 2021).

[5] N. Emmett, “Play free online minesweeper,” Minesweeper Online, https://minesweeperonline.com/ (accessed October 15th, 2024).

[6] “Logic minesweeper (No Guess),” LogiGames, https://www.logigames.com/minesweeper/logic (accessed May 14, 2025).

[6] S. Tatham, “Mines,” Simon Tatham’s Portable Puzzle Collection, https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/mines.html (accessed May 14, 2025).

[7] “Edit in DokuwikiEdit in AsciidocEdit in Markdown 🌱 PlantUML at a glance,” PlantUML.com, https://plantuml.com/ (accessed May 15, 2025).

[8] I. Sommerville, *Software Engineering*, 10th ed. Pearson Education Limited, 2015.

[9] Games Computers Play, “Python script beats Minesweeper in seconds (30+% success on expert),” YouTube, https://www.youtube.com/watch?app=desktop&v=uvRQUWxOHqo (accessed May 15, 2025).

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: The MIT Press, 2022.

[11] “Naked single - sudoku solving technique,” Sudoku9X9, https://www.sudoku9x9.com/techniques/nakedsingle/ (accessed May 15, 2025).