Computer Science Department

Capstone Project Proposal

Devils Grip

[December 14th, 2020]

Steven Moody

CSC 520

CSC 521Project Advisor

Dr. Thomas Schmidt

Contents

Student Objectives

The computer Science capstone is intended to be a showcase of a students learned knowledge and the application of such knowledge in a way that is both easily represented as well as complete in nature. I believe my proposal, application, and presentation of my capstone project, Devils Grip, will satisfy both requirements. My project proposal consists of applying as well as furthering my knowledge in both the technical aspect as well as the ability to convey my proposal in an easily understandable way to those outside of the field (in terms of a product presentation).

In terms of technical knowledge objective, my proposal as a whole contains a solutions process containing the technical steps I intend to take in order to complete my capstone project, as well as a tools list containing every piece of software I intend to use to do so. In doing this project I will be exposing myself to data rock/ data core to develop a local database for uninterrupted access to a high score leaderboard. I will also be learning about and utilizing the Unity game engine to help further my knowledge in the use of game engines. I will be programming in C# as well as Swift which will assist in diversifying my knowledge across various programming languages. Upon completion of my capstone there will be two presentable game mediums, one that will be playable on the PC and then one that will be playable on an IOS device. Lastly, as a whole I will be furthering my knowledge in the Software Development Life Cycle in having to accomplish each phase of that on my own.

In terms of nontechnical objectives, I feel that there are a couple areas that I will be growing in doing this capstone project. For starters, in finding the problem that allowed me to justify the development of my game I have had to speak to members of my family to seek how to go about the development of Devils Grip. Due to the fact that they are not in our field, I had to learn how to speak in a non-technical aspect to help develop the requirements needed for this capstone project. To compound onto that, at each phase of the project I plan to have external testing done when possible and have to again rely on individuals who are not technically inclined and will not be conversing with me using technical terms. These two aspects will be furthering my ability to showcase software development and converse about as well as present products with/to clients once I am part the workforce.

In conclusion, I feel that the objective of developing Devils Grip are both technical and nontechnical. Both of which I feel are the goal of the Capstone proposal and completion/presentation of the Capstone project. From furthering my knowledge in the Software development Life Cycle to being able to talk about the product itself to a potential client, I feel that the project I am choosing to pursue will further these things very well.

Problem Specification

As we all know, COVID and the quarantine that ensued presented many of us with more time in both isolation as well as a lot of time away from our everyday lifestyles. I know that myself as well as my family decided to spend 1-2 nights a week playing board games or some sort of card game to help pass the time. At first, coming up with games to play was rather simple as we quickly went through our assortment of board games and transitioned to a variety of card games. However, this quickly became almost as boring and repetitive as quarantine itself. We then started to use Google to help us decide what games to play and started competing in games traditionally meant to be a single-person game to see who could complete it faster. That is when we stumbled upon the game known as Devil's Grip.

Devil's Grip is a solitaire based game in which a single person attempts to sort two shuffled decks of cards (with the aces removed) into 3 rows and 8 columns with each pile of cards consisting of a certain arrangement of same-suited cards in a sequential order. When completed successfully, the top row contains 8 piles of cards that each contain a 2, 5, 8, and a jack. The piles in the middle row should contain a 3, 6, 9, and queen. Lastly, each pile of cards within the bottom row should contain a 4, 7, 10, and a king. Breaking this down further, essentially there should be 24 individual piles of cards that each contain 4 cards of specific values in a specific order.

A player begins this game by shuffling both decks together and then taking the top 24 cards of the newly formed deck of cards and placing them in the arrangement described above (1 card in each position forming a 3 x 8 grid). Once these cards have been placed, the player can swap cards one at a time into new positions as necessary. For example, if there is a 3 of diamonds in the bottom row and a 4 of hearts in the middle row, the player could swap these cards so that each number is in its correct location on the grid. Similarly, the player can also begin stacking cards as necessary as well. Using the previous example, lets say that after the player moved these cards, they noticed a 6 of diamonds was in the top row. The player could take this 6 and place it on top of the 3 of diamonds they just moved previously. The now empty location where the 6 was, is replaced by a card from the top of the shuffled deck. Once this is completed the player then begins to flip over cards from the deck in sets of 3, only being able to use them in reverse order in which they were removed from the deck and fielding them as necessary, using the outlines rules as a guide.

Traditionally, this game ends when a player either successfully completes this arrangement with each row containing 8 piles of sorted, same-suit cards based on row or there are no moves left. If a player successfully sorts the cards, they receive a score of 0 which is the best possible score. However, if they still have cards left in the deck, the number of cards left equals their score. The lower a score the better.

Now that we have an understanding of the game, let me explain why I decided to make an application capable of running it. When my family was playing these card games, very often the surfaces that we were playing on would become cluttered and it would almost always become too cluttered to track the card arrangements. The surface area required to play this game was big and if playing on a floor we would also have to account for pets and such moving the piles and creating confusion. Often, we would use the app store from apple to download games and play them digitally to avoid this, however when we stumbled across Devil's Grip we could not find an app that contained the game nor a website. That is when I got to thinking, why not create it myself? This app would allow for players to effortlessly play this game anywhere at any time as well as track their overall scores within the app itself so that if they were competing against someone else it would be easily tracked and shown. While this may seem simple in nature, I found it to be rather interesting that no one had bothered to include it within their app and during this time, the ability to pass time with effortless games seems to be very important. Upon research I realized that I can actually create two playable versions of this game during this creation process, one that will be playable on a PC and one that will be playable on an IOS device.

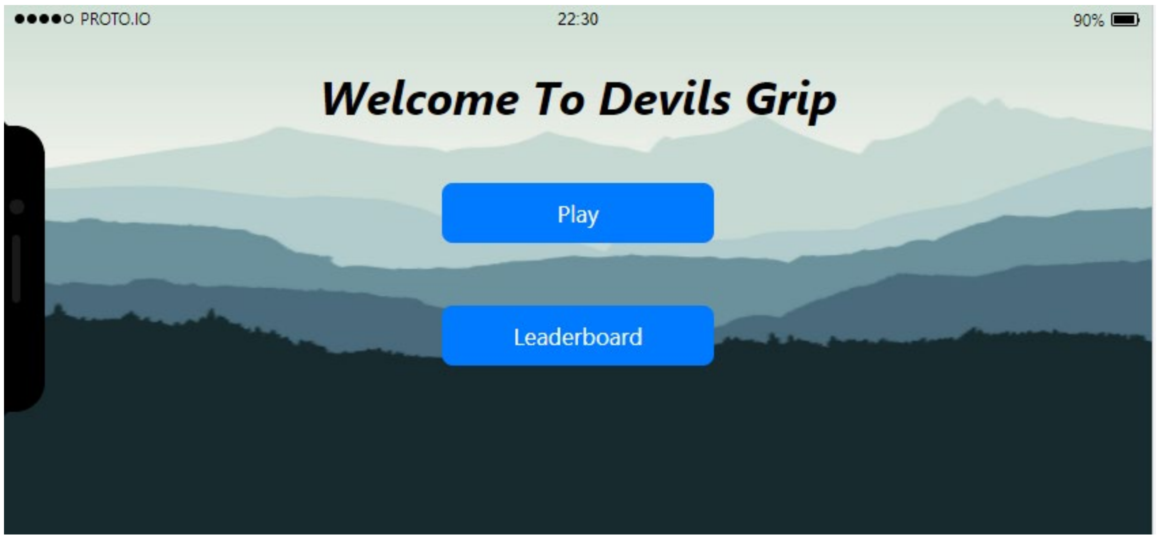
Requirement Collection and Analysis

- Stakeholder Identification
    - Myself(Steven Moody); being made for family use (Mother, father, Sister, and girlfriend are clients); open for download by anyone with IOS device OR playable on a PC.
- Requirement Lists for Requirement Specifications
    - A fully playable game on the PC that includes all of the PLAYABLE features that will be seen on the app created for IOS devices (see below for those features)
    - Fully functional IOS application that includes the following features:
        - YouTube link to how to play game: https://www.youtube.com/watch?v=0fIAIxwQVLg
    - A functional PLAYABLE game
        - Upon accomplishing the above use cases, Client(s) will select the "Play" button at which time they will be brought to a new screen/view where cards will be shuffled/dealt and the player then can begin playing "Devils Grip" following the constraints (rules) of the game.
    - A functional IOS application
        - Client(s) will download game to IOS (Apple) capable device, upon completion will start the application and ensure they are brought to the home screen
    - A functional home menu with "leaderboard" and "game start" buttons

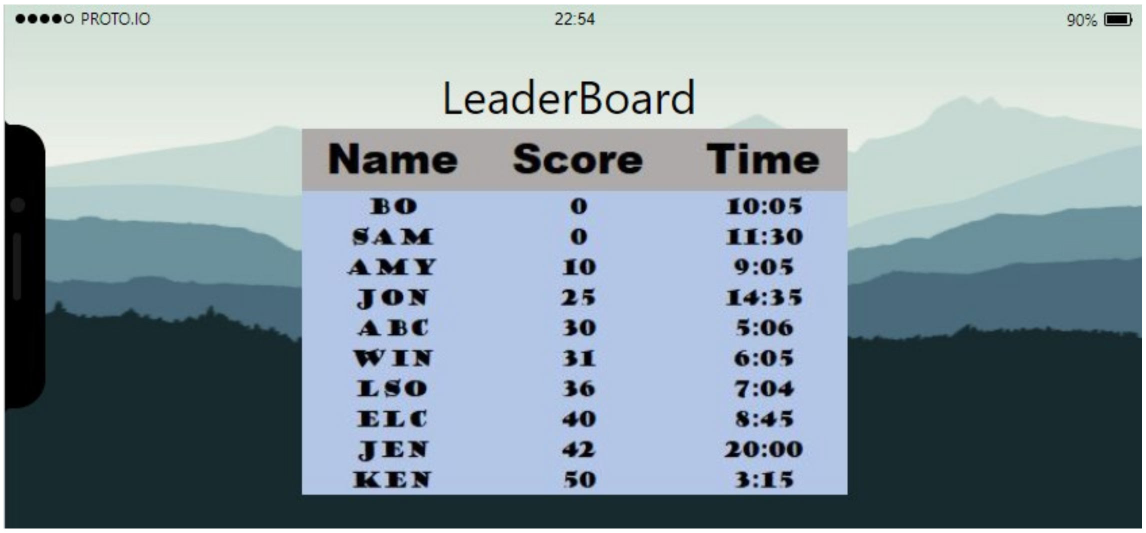
        - Clients(s) will start the IOS application and upon loading of the home screen will be presented with three SELECTABLE buttons. The first will be "Play", the second will be "Options", and the third will be "Leaderboard"
    - The ability to input name associated with each high score
        - Upon completion of the above Use Cases, once a game has been successfully finished or no further moves remain, Client(s) will be given the option to enter a 3 character high score name to be placed on the "Leaderboards" if applicable.
    - The ability to start over once a game has completed/no more moves remain
        - Upon completion of the above Use Cases, Client(s) will be given the option to either begin a new game or return to the home screen menu
    - The ability to choose difficulty
        - At the home screen menu, Client(s) will select the "Options" button and when brought to a new screen will be allowed to choose the difficulty in which they want to play the game at
    - A background that is neutral and appealing within the menu
        - Upon loading into the application, Client(s) will be greeted by a background that is appropriate based a background of their choosing
- Major goals/tests associated with the above requirements specifications include:
        - Client will download a functional IOS application of "Devils Grip" and will test that each feature listed above performs as expected.

Prototype Screens

These screens are exclusively to show what the app version of my game will look like. The PC version of my game will look similarly however, will likely be excluding certain features.



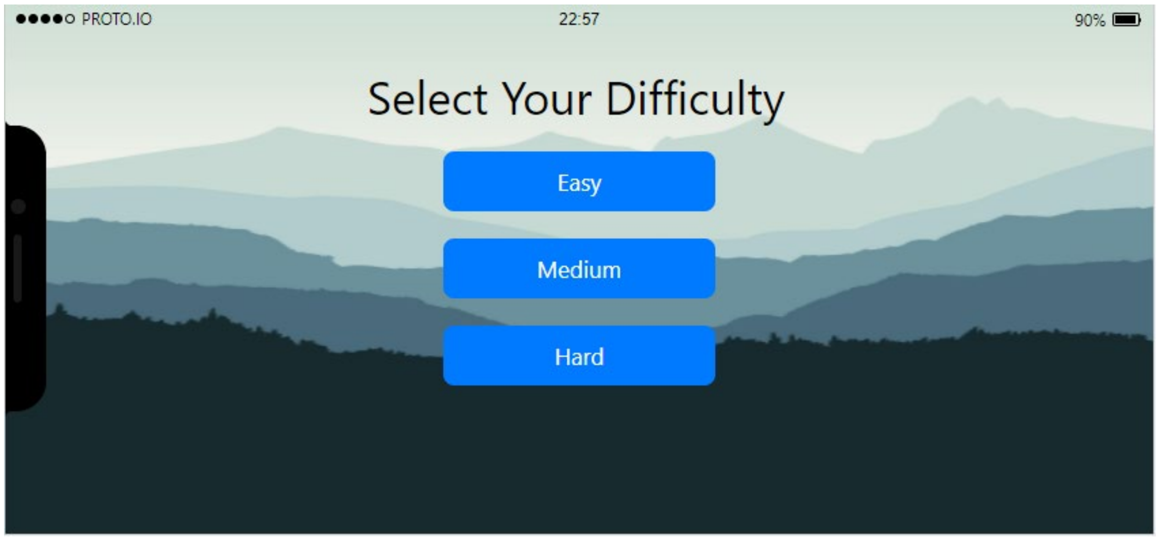
Upon loading into the application, this is what the home screen will look like. Each of these buttons will be selectable and will lead to follow-up screens.



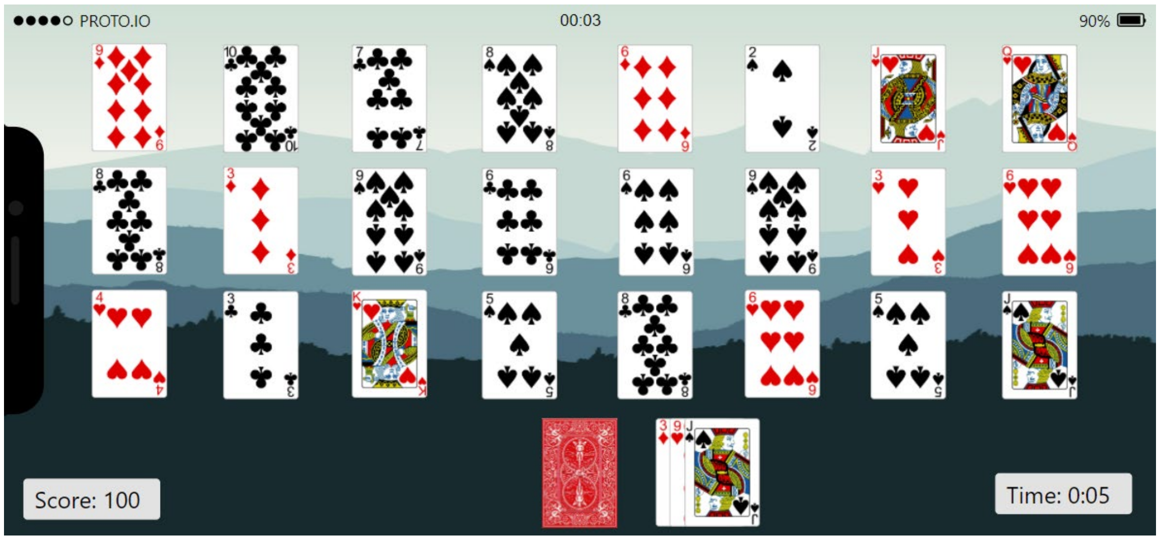
LeaderBoard

Name	Score	Time
BO	0	10:05
SAM	0	11:30
AMY	10	9:05
JON	25	14:35
ABC	30	5:06
WIN	31	6:05
LSO	36	7:04
ELC	40	8:45
JEN	42	20:00
KEN	50	3:15

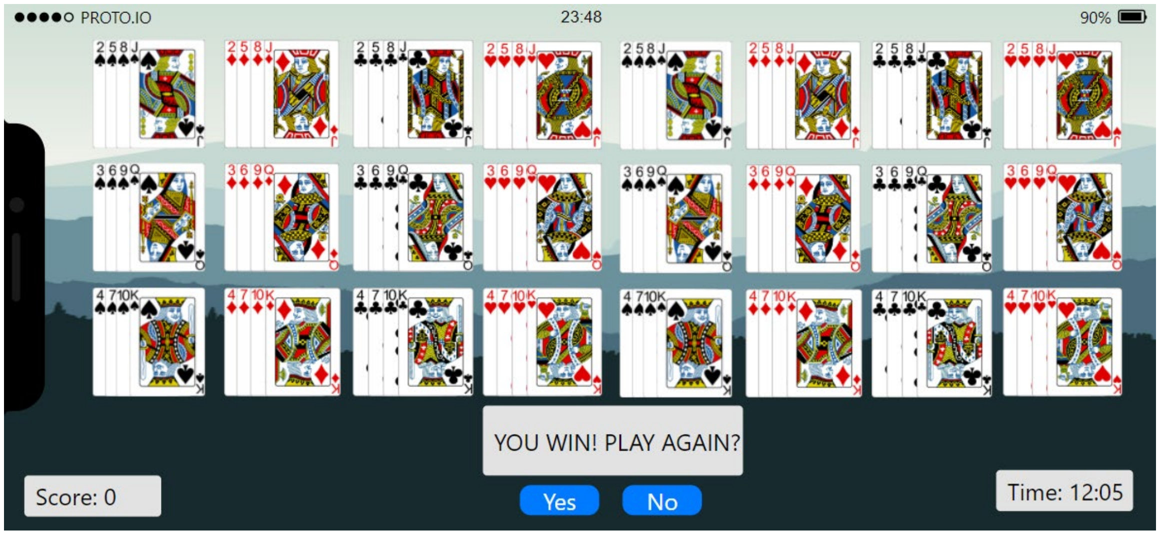
This is the leaderboard screen that will house the TOP 10 scores, ranked by score first (the lower the better), followed by time (again, lower is better).



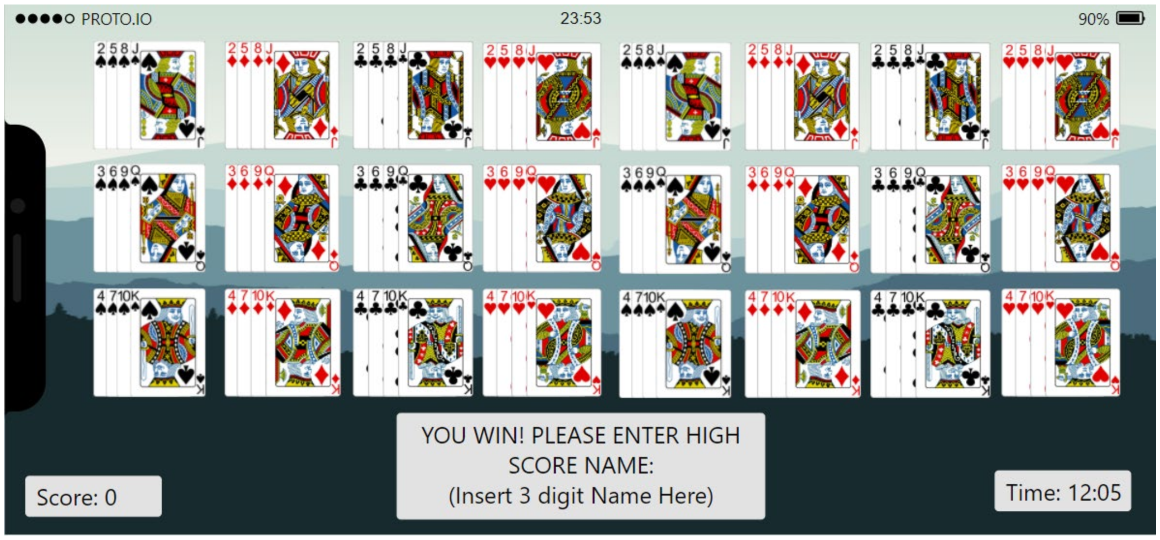
Once you select "play" from the home menu, this is the follow-up screen. From here you will select your difficulty and this will trigger a switch statement within the code depending on which you select that will randomize the cards (based on your selection).



This is a prototype of the card layout at the beginning of the game. Here players will be able to move cards as allowed by the game, and move through the deck at the bottom until no move are left or the board has reached its final stage.



This will be the completion screen if you DO NOT have a score high enough to add to the leaderboard.



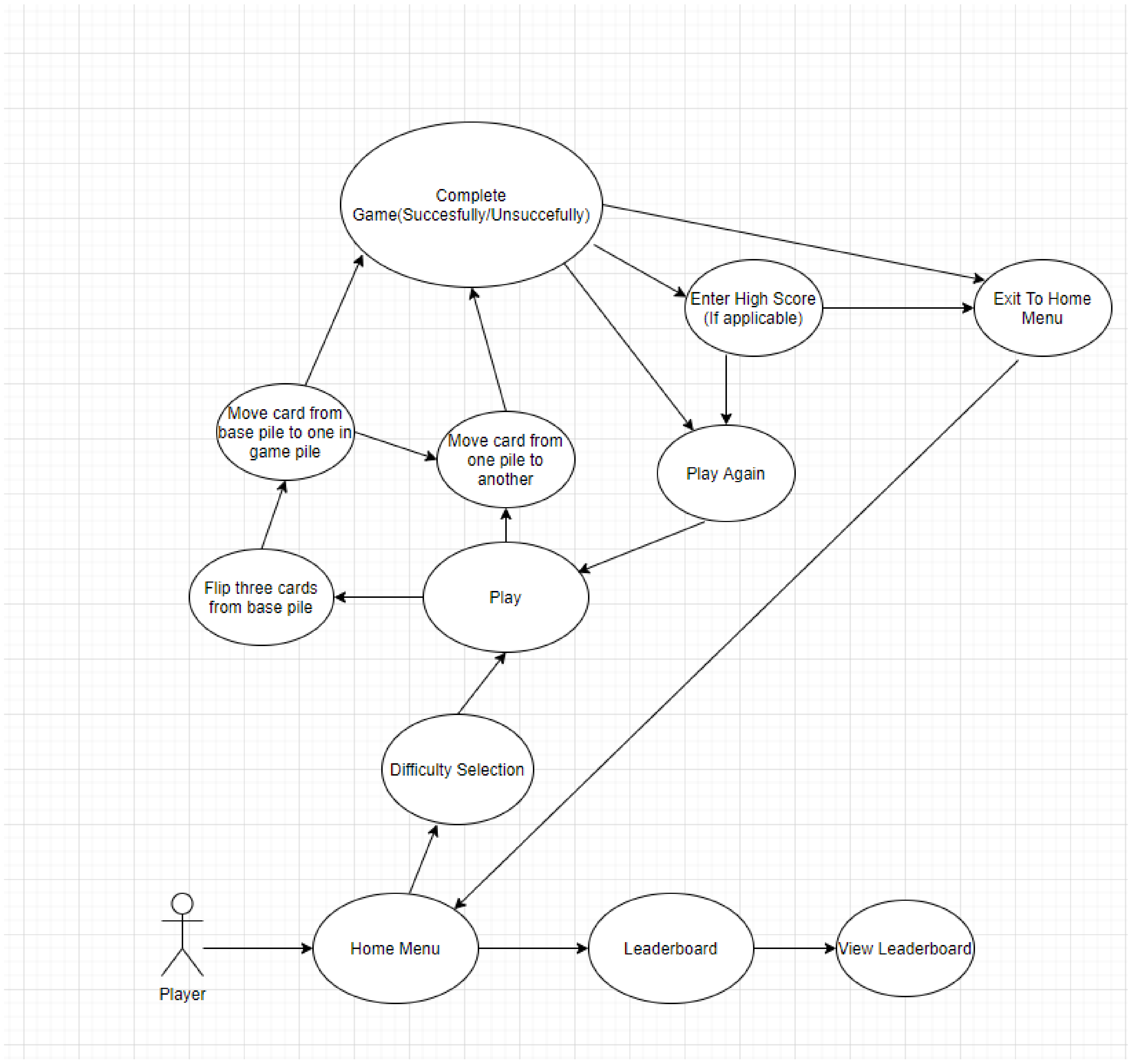
YOU WIN! PLEASE ENTER HIGH SCORE NAME:
(Insert 3 digit Name Here)

Score: 0

Time: 12:05

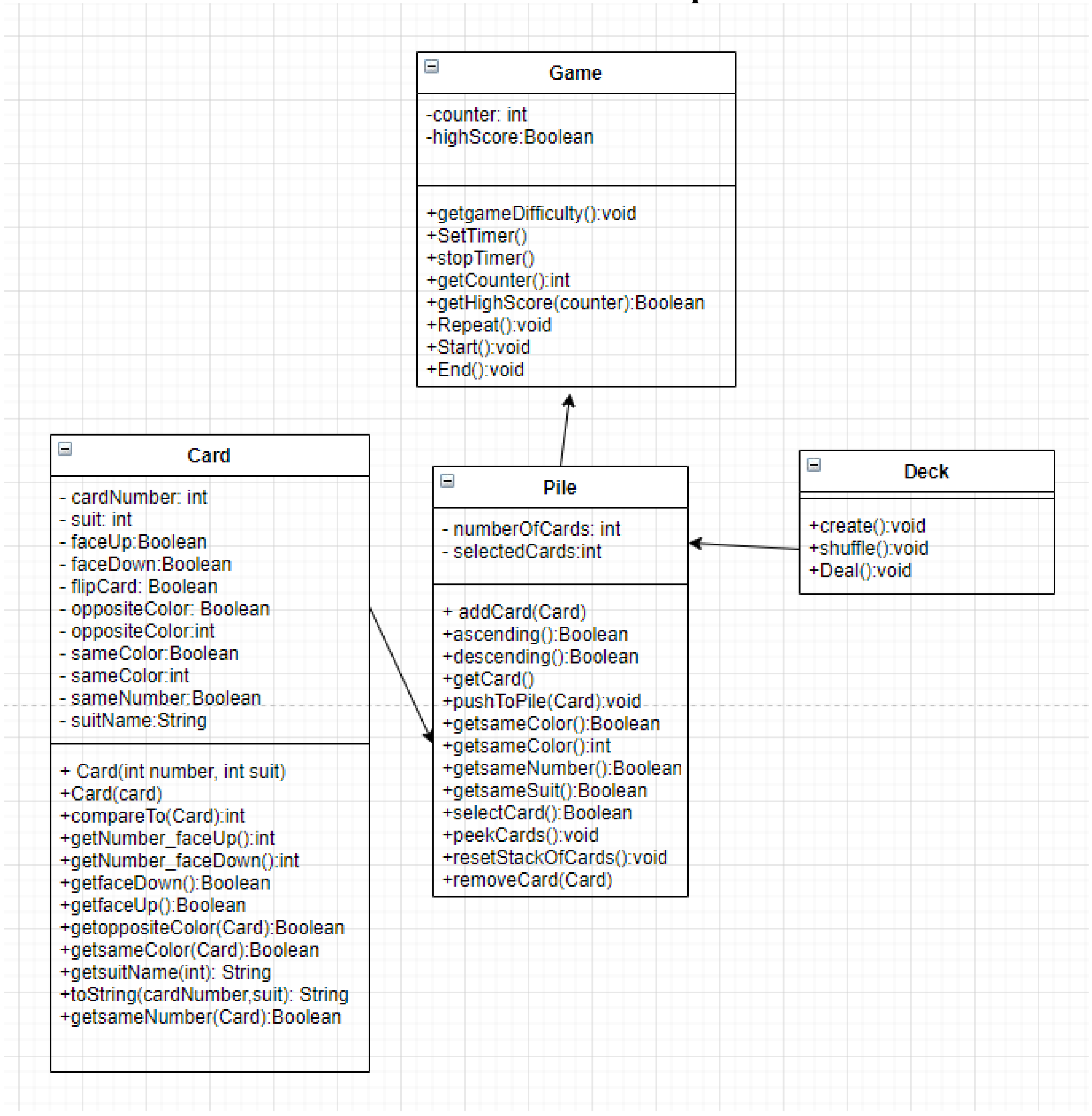
This is the completion screen if you DO have a score high enough to add your name to the leaderboard.

Sequence Diagram for "Devils Grip"



Complete Game(Succesfully/Unsuccefully)
Enter High Score (If applicable)
Exit To Home Menu
Move card from base pile to one in game pile
Move card from one pile to another
Play Again
Flip three cards from base pile
Play
Difficulty Selection
Player
Home Menu
Leaderboard
View Leaderboard

UML For Devils Grip



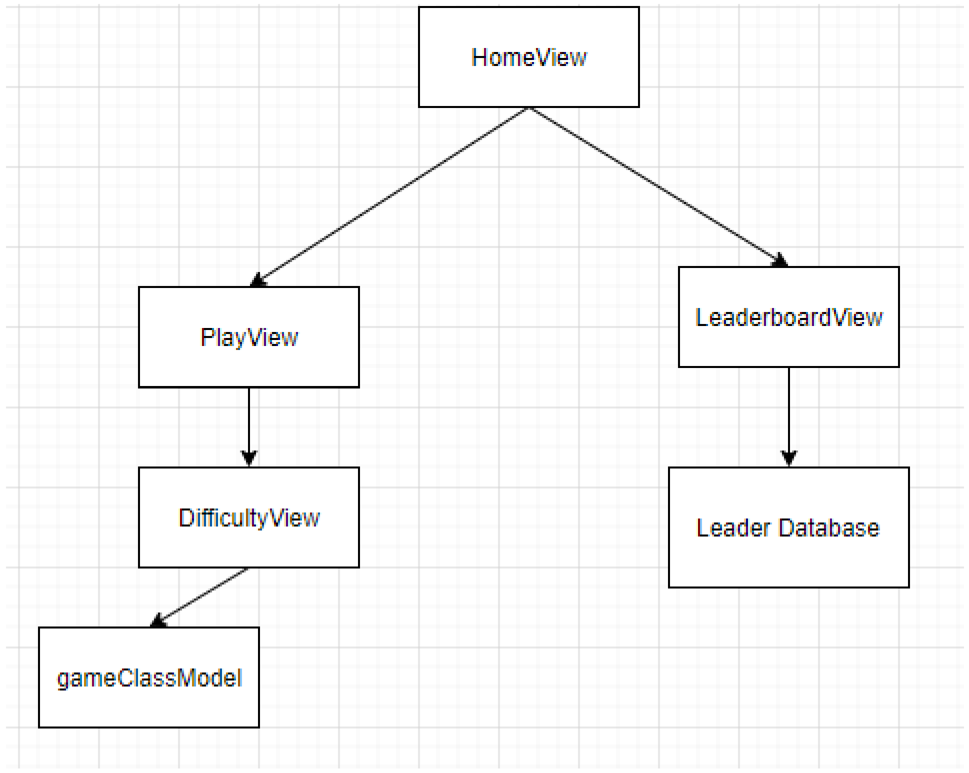
Description

This UML is specifically for the in-game aspect of my IOS application. What we have here is the Game class will be the driver code for the rest of the classes. It will hold the functions of starting and stopping the in-game timer, the counter that will decrease based on percent completed as well as a function to add your score to the leaderboard database. One thing to note here is that any score can be added to the leaderboard, however when a user goes to view the leaderboard they will only be able to see the top 10 scores, ranked first by score and then if the score is equal second by time to complete.

Following the Game class, we have the second most important class which is the Pile class. This class contains all the functions necessary to control each pile seen on the screen to include the deck pile itself. There are many functions here but to note is that a user can add/remove one or more cards from a given pile as well as flip over cards in each pile as necessary or allowed. There are also checks in place to ensure a user cannot add invalid cards to any pile nor remove any card that is below another without first moving the card above it. There are also a few Boolean function calls to ensure that only same suit cards are added to specific piles as the game rules specify.

Following this we have our Card and Deck classes. The deck class is rather straight forward and has minimal functions just to create the deck itself, shuffle(randomize based on difficulty specified by the Game class), and deal the cards to each pile. As for the card class, this class contains all the core variables and functions necessary to detail a card. Each instance of a card is given a value whether it is face up or face down, this value with remain the same throughout the duration of play and there will only be one instance of each suit with 4 instances of any number. The numbers will range from 1(ace) through 13(King). Each suit will have both a number associated with it as well as a string of the suit name that will also be assigned to each card.

Swift/Unity UI Model



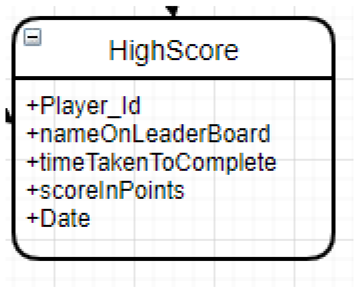
Description

It is possible that Unity UI can cover this, based on my research even though it is being developed on Unity it will still need to be imported to XCode and may need to be modified slightly via swift to enable the UI for IOS users. However, either way this will be the structure of the UI. I am not entirely sure yet exactly what code will be necessary for each view, but above is the current state of how I believe the views will flow. From the HomeView as with the sequence diagram, we will have the option of changing to three different views; the playView, the LeaderboardView.

The playView will transition immediately to a difficulty selection view which will contain an option for 3 different difficulties, respectively. From here the view will then change to the gameClassModel which will be the view for the remainder of the time you are in game and will return to the homeView upon completion/exit from the gameView.

The LeaderboardView will take you to a view of the leaderboard database, which will display in a standard list format with the playerName entry on the far left followed by their overall score, the time it took to complete the game, and lastly the date in which the score was entered. To note here, only the top 10 entries overall will be shown.

Core Data/Rocket Data



Description

For my game, having a way to store high scores is very important, however my app will likely not have networking functionality nor will it require multiple tables within the database to store these scores. Essentially these high scores will be stored locally within the IO application itself. My proposed solution is to use the “Core Data” storage framework developed by Apple to locally store the high scores within the app itself. While core data is technically not a database itself, it can be used with SQLite to function as an offline database that is local to the IOS application itself. This does mean that when the app is deleted, the leaderboard would also be deleted however this seems to be the best option considering that it would be an all-in-one package with my IOS app. I will still be using SQL script to create and query the database that is created through data core however I can also use an NSPredicate object with each query to allow for more complex returns if necessary.

If for some reason this proves to be a bad choice I have a backup of using Rocket Data which may be necessary to use the game through unity on a PC. Rocket Data functions exactly the same as Data Core with two exceptions, it does not block the main thread and it is not tied specifically to Apple.

Benchmark Specifications

As with any program or product, benchmarking is an integral part of ensuring timelines are met and adjusted accordingly if needed to allow for increases in unforeseen workload. My capstone project allows for many points in which benchmarking can occur as well as be monitored to ensure I am on a timely schedule as well as my proposed deliverables are accomplished. I believe that my benchmarking should be broken down into the following sections: Unity scripting and backend core game code completion, Integration to Mac and XCode, UI completion, proper database implementation, IOS app creation, as well as de-bugging at each major phase.

Benchmark One

First, my project can actually be playable on two fronts, the end state is to have a fully downloadable and playable app through the Apple app store however once I complete the unity scripting and backend code implementation completed the game will technically be playable on any Windows PC. I think this would make a great first benchmark as this will ensure I have the core game completed and also ties in well with my proposed timeline(more on that next).

Benchmark Two

Second, once the game is completed through Unity I will have to export/import it to XCode and this in and of itself should be a smaller second benchmark as there may be some code changes or other problems that arise that could need to be worked on before moving to the next step.

Benchmark Three

Third, once my game is residing on the Apple side of things and is ready to be transitioned to an IOS device, I will need to use Swift UI to create the views that the User sees at each stage. This will likely be very time consuming as I will have to tie each portion of code to various views/models and I believe that the completion of this would make sense to be a third benchmarking point.

Benchmark Four

Lastly, I think that a final benchmarking point should be the emulation of the game itself through the XCode emulator before submitting it for review by Apple for placement on the IOS store. I think this will be vital as while I will be de-bugging at each stage when possible this will be the last opportunity to quickly implement any fixes needed, as future fixes will require a turn around time from Apple before being released.

Tool List

I will be using various tools throughout this project. Each tool will have a vital role to play in my apps completion and will ensure efficiency as well as accuracy. The tools I will be using are listed below as well as the reasons behind each tool.

Git: I believe that everyone will be using Git as a version control and I am not different. If something happens I want to be able to re-trace my steps and even roll back to a previous version if I have to.

GitHub: While Git offers great local version control I will be utilizing a repository so that I can have a more easily viewable repository for all of my code and documentation, this will also be shared with my advisor for CSC 521.

Unity: Unity is the game engine in which my entire game relies upon to work. I chose Unity purely out of the desire to learn more from this project. I know that traditionally Unity is used for 3D games or more advanced 2D side-scrolling games and that I could have used something written in code, but I feel that this is a good introduction to utilizing a game engine for me and I think I will have great success in doing so.

Visual Studio: During the first portion of my project, I will be using Visual Studio to write my backend code that will be linked with Unity to create all of the functionality of the images being manipulated.

XCode: During the second portion of my project, I will need to transition to Mac in order to build the IOS app and thus will naturally be using XCode to accomplish this.

Swift UI: Swift UI will be used to develop the User Interface that every end-user will see. I chose Swift UI as I personally do not have very much experience with UI in general and based on research this seems to be the best option to use with Swift.

Core Data: As stated above, my app will not have networking functionality, thus I need to find an offline database solution and Core Data seems to be a viable option as its implementation can allow for my database to be stored locally within my app.

Rocket Data: Rocket Data is a backup to Core Data and again as mentioned above functions very similarly in comparison to Core Data, however may be necessary to use for my PC implementation of my project.

XCode Emulator: During my UI implementation I will likely be relying on the XCode Emulator frequently to test each view for functionality and accuracy.

IOS Device: My completed project will be available for download through the Apple app store thus, I will need an IOS device to prove that this in fact works.

Windows PC: My initial stages will not be possible without the inclusion of a Windows PC.

Mac: The second half of my project will not be possible without the inclusion of a Mac.

Proto.io: This is being used to help visualize my capstone visually.

Time Schedule

For any app development a proposed time schedule is as vital and goes hand in hand with benchmarking, without one or the other a product may never come to fruition. Throughout the 14 weeks I will be creating this product, various tasks will be completed and I have proposed the specific schedule listed below.

Benchmark One

Weeks 1-4: here I will be in the beginning stages of my product and will be utilizing both Unity and Visual Studio in unison to create the graphics to my game itself as well as create/implement the backend code for the games functionality. This stage will be most vital as no other future stage can progress without the completion of this one.

Benchmark Two

Weeks 4-8: This stage will consist of me transitioning the finished Unity game from a windows PC to Mac and migrating it with XCode. I will also be using this time to create the UI that will be seen upon completion of the app. While migrating may take some time, the majority of this time will be spent on developing the UI necessary for users to seamlessly navigate my app.

Benchmark Three

Weeks 8-11: During this stage I will be developing test cases for both the PC version of my game as well as the IOS version. I have allocated 3 weeks to this due to the fact that developing a complete suite of test cases to thoroughly test both aspects will likely be challenging and very time consuming.

Benchmark Four

Weeks 11-13: At this stage, I should have a completed game that is ready for export to IOS. What I will spend this time doing is cleaning up any last-minute bugs that are found during alpha/best testing and then submitting the finished app to Apple for addition to the app store.

Weeks 13-14: This final stage will be spent completing any necessary paperwork needed for the completed package submission of my capstone as well as the final presentation itself.

Proposed Grading Scheme

The grading scheme for this project took some careful time and consideration as I did not want to get overly ambitious in any one area and not be able to deliver in others. With that said I believe that the grading scheme should be broken down as follows:

PC Executable: I believe that this should be worth 25% of the overall grade. This will include anything that has to do with being able to fully run my game on a windows PC. My reason for that is that the entirety of my project hinges on my ability to complete this portion and no other can be completed without it.

Mac Integration/UI for IOS: The area will include everything that has to do with the Mac portion of my project with the exception of the UI itself. I believe this should be worth 25% of my overall grade as well due to that.

IOS App/emulated: This is going to be based around my ability to actually play the game on IOS after everything is complete and how efficiently and accurately my work was. For this reason as well as the IOS app being the primary “selling” point of my capstone, I feel this should be worth 20%.

Test Cases: Test cases are going to be vital to the completeness of my code as well as assist me in ensuring functionality of both my PC game as well as the IOS extraction of it, thus I believe this should be worth 10%.

Report: My final report will include everything that I have done in detail as well as any diagrams and visuals necessary and as a result should be worth 10%.

Final Presentation: This will be a culmination of my 4 years as a Computer Science student at Salem State University, a showcase of both the dedication of the faculty as well as my ability to showcase what I have learned in presentation form, I believe this should be worth 10%.

Deliverables

The list of deliverables seems to be the same for every student completing a capstone, which include: the original proposal and presentation file(s) from CSC 520 as well as any amendments to this proposal which will be approved by Professor Schmidt, all system Architecture diagrams necessary as well as detailed information about them, my source code with comments, the documentation of project functionality which will include screenshots, videos, and any other material necessary to prove completion as well as a sample output, any executable/ the project itself(which everyone with an IOS device will have a portion of already), all presentation documents, a personal project journal written by myself detailing my journey through completing this project, a project post mortem that discusses what I took away from this project and what I could have done better, a list of what(if any) areas of the project were not completed, the presentation of the final project, and lastly a user's manual detailing how to use my project.