Julian Panici
CIS 611
Final Report

Introduction

For my half-semester project, I chose to recreate the game Immaculate Grid, a baseball tic-tac-toe style trivia game. However, instead of baseball, my game focuses on soccer, specifically the English Premier League. I called it FootyGrid, and the rules are easy to follow. The game begins by choosing 6 teams, one for each row and column on the 3x3 game board. The player is then able to click each button on the grid. After clicking, a window appears that allows the user to enter a player's name who they think has played for the selected button's row/column team combination. After submitting their search, another window appears with the results, and the user can then pick a player from the list to submit as their final answer. If correct, the player's image appears on the selected button. The user has 9 guesses to try and correctly answer each square on the grid's team combination and win the game. Due to this being a web-based game, there are many choices I made in terms of what software engineering artifacts to prioritize, and what I was able to leave out. In this report I will go through each software engineering phase, and discuss those choices while reflecting on the success or failure I had with them.

Definition

Requirements

For this project, I used a combination of formal requirements (shall, will, could), and informal requirements in the form of a document I wrote out describing the initialization and logic of the game I was creating. This combination proved to be very beneficial.

The formal requirements helped me to identify which mechanics in my game were crucial to it running correctly, and which ones were good additions but not necessarily needed for the game to run properly. This was great for mapping out where to start building my project, and for deciding what aspects I can skip and come back to at the end once I have a working prototype.

The informal requirements allowed me to write out step by step how my game should run from start to finish. I began with loading up the page and initialization of the game, then described the actions the user can take, and ended with the win conditions and resetting of the game. I found this to be as beneficial if not more than the formal requirements as it clearly mapped out the order in which different functions should run and when certain actions should be executed. It allowed me to declutter my head and gain some clarity as to what parts of my project I wanted to work on as I worked through it.

Use Case Diagram

I created a use case diagram after finalizing my requirements. This diagram allowed me to take the requirements I created and turned them into use cases which would help me when creating each function. I found the use case diagram to be one of the most useful artifacts in the entire project as it allowed me to clearly trace my requirements to actions that could be performed in the system, whether by the user or the program itself.

Development/Design

Agile Development Process

I utilized an agile software development process for the duration of this project. Through each step of my software process, I found myself constantly going back and updating my requirements, which in turn led to me changing my use cases. This cycle repeated while I created each part of my code, and even in testing I had to adapt and update my functions to make certain functionalities able to be tested. I found this process helped me greatly in this project, as I never felt locked into a certain version of my project, and I had the freedom to change and adapt all of my artifacts as I felt was necessary.

UML

I did not create a UML diagram, or any other design diagram for this project. All of my functions were created in javascript, which means they were all held in a file simply named script.js. This means I did not have any classes, completely nullifying the purpose and benefits a UML diagram would have provided me. Due to this being only a half-semester project, the scope and scale of my project was not very big. This enabled me to keep track of what I had created simply in my head, and it never got to be large enough that I became confused or overwhelmed with all of the information. The other upside to this project being created using javascript was that when I realized I needed to add more, I was simply able to create another function and add in any functionality or use that was necessary at the time. I will admit, I most likely could have benefited from creating some sort of work flow diagram to keep me on track, but there was never a point where I felt it was 100% needed to complete the project.

Verification

Testing

I used a combination of unit and system testing in order to fully test the functionality of my code. This project was my first attempt at writing and testing javascript and html code, and as a result I was very unfamiliar with what to do to create unit tests for most of my functions. As a result, I created only a few unit tests for the functions which I knew I could concretely test the functionality of, and then I created many system tests in order to both test the rest of my functions and make sure my tests covered every requirement. I likely would have benefited more from creating unit tests for each function instead of unit testing all my functionalities. However, creating system tests required me to think through every action the user could make, and figure out a way to appropriately test all of these actions. This allowed me to gain a deeper understanding of the code I had written, as well as provide myself with more security and confidence in my project.

Traceability

I was able to create traceability matrices from my requirements to use cases, as well as my requirements to system tests. This proved to be massively useful to me as I was able to see what I had and had not tested yet. This was not only a good progress indicator as I worked through each test, but it also allowed me to pinpoint errors quicker as I could quickly identify the requirement being violated, then trace that to the specific use case and tests being violated, which finally led me to the function which needed to be fixed.

Maintenance

Though I did not create any maintenance specific artifacts, all the different artifacts I created in the previous steps of this project can be used to aid in future maintenance of what I have created. If and when an issue arises, I can identify which requirement it violates, and then use my traceability matrices to figure out which use case, and in turn which function, needs to be fixed.