Project Documentation

**Description of the Game**

Densmordle is yet another variant of Wordle — however, this one is unlike any other. Instead of randomly selected five-letter words, you are guessing randomly selected classrooms from across the Boston University campus based on the photo of said classroom that you are prompted with. The user inputs their guess — which is then error checked to ensure that it is a valid guess. A valid guess consist of both a building that exists (i.e. ‘CAS’ or ‘PHO’) as well as a room number that exists (a combination of any three digits). Based on the guess that the user has input, each of the characters are displayed with different colors as the background: green meaning that the correct character is in the correct position as that of the actual classroom, yellow meaning that the character is correct but in a position other than that of the actual classroom, and red meaning that the character is not at all found in the actual classroom (similar to Wordle).

Densmordle has three different game modes for you to choose from (1) regular, (2) timed, and (3) limited. Regular is an easier gamemode of Densmordle as it allows an unlimited number of guesses. Timed is a bit more difficult as the objective of the game is to correctly guess as many classrooms as possible in 60 seconds. Limited is also a bit more difficult as it restricts you to six guesses to correctly identify the classroom — otherwise, you lose! This game is intended to be enjoyed by those who are familiar with the BU campus — students, staff, professors, and alumni.

**Decisions Made**

When we first decided on an idea for the project, we ideally wanted to make it a website — like Wordle is. Throughout the process of creating our program, we were conflicted as to whether or not to try to take what we had in C++ and recreate it using HTML, CSS and JavaScript in order to create a website. However, since none of us are very experienced in HTML, CSS or JavaScript, we ultimately decided to stick with our C++ program as it worked and it would have been a better use of our time to add more features to our game rather than redo what we had done in another language.

Additionally, we wanted to limit the number of classrooms in the game for the scope of this project. The Boston University campus is very large and has a lot of classrooms that many students may never even step foot in. We decided that classrooms that have a capacity of 80 people or greater would be added. As a result, users would be less likely to be prompted with an obscure classroom that is in a building that they may have never even heard of.

The decision to store the classroom information in a class was to allow for easy access to all the necessary information for each classroom. In terms of storing the classroom information, inthe game, the user is supposed to guess both the building name and room number of the classroom whose picture is presented. Therefore, making the building name and the room number members of a class which can be instantiated for each room allows the easier checking of user input and makes the code easier to navigate through. Considering the possibility of new classrooms being added to the list, we decided to read in the classroom information from a .txt file containing the building number, room number and link to a picture of the room. This also allowed our code to be much easier to debug and to read. An array containing all of the rooms can be created in the main function using the MakeRooms function.



Originally, much of the code was written in a main file. However, we decided to convert it into .h and .cpp files for modularity, allowing the code to be more readable and the game to be adapted to different modes or applications.

**Challenges**

One of our main challenges was our limited knowledge of languages outside of C++ which consequently meant that we were not able to explore the project to an extent our original design intended. While we had the idea and were able to implement it with our current skills, ideally we wanted to enhance the game interface with a website or an app rather than the console. However with a limited amount of time, we decided that the more feasible decision was to add more features that allowed Densmordle to have a more impressive game play overall.

A few other challenges faced through this project were the logic behind the error checking and the slight lack of readability of shared code. There were problems that came up in the error checking when matching the real answer with the guessed one and code running over the previously administered commands. At times it was hard to understand what was happening in another person's code because there weren’t comments or were abstruse variable names. However, this was remedied with some communication with each other about improving readability for the collaborators.

One of the first challenges we faced was figuring out how to open an image link in the browser. This was easily resolved in a meeting with Professor Densmore. Something confusing with this, however, was that while one of the text editors used for coding said that there was an issue with the line , it actually was still able to compile and run.

Another challenge that was faced in the Timed mode was figuring out how to call the timer to run synchronously, so that while the game is being played, the timer is still running, so it ends at the appropriate time. This required use of a lambda function and passing the boolean argument, time\_over, as a reference variable. See the Timed Mode section further down in this document.

One of the challenges faced with using the PlaySound function was needing to add “-lWinmm” to the end of the line used to compile the files. This was not explicitly obvious in most searches on the internet discussing the PlaySound function.

Another challenge was for error checking the input, since fscan was used, but in class we most often used cin, it required some other functions like getchar() instead of cin.ignore or cin.clear to make sure the input that caused an error was handled properly. This was especially challenging for the TimedMode, since after a minute, a message is shown, and affects the way fscan reads information.

**More Details**

**Recording and displaying the highest score**:

The highest score of the user is stored in a .txt file, and it is deleted and changed if the user has a new high score. The display\_highest\_score function is called with a reference to the current score, and displays the final score of the current game, as well as the highest score.

Required libraries:

* <fstream>

**Timed Mode**:

In the timed mode, the user tries to guess as many classrooms as possible within a time limit of 1 minute. Once the time ends, a message is displayed which states that the user is allowed to make a final guess. 

Required Libraries:

* <chrono>
* <future>
* <thread>