Progress Report

Advanced Programming Assignment 2

Louis Baudinette: s3668025
Sim Bilkova: s3748189
Lachlan Boughton: s3769219
Peter Bui: s3786794

1. ADT Usage

The only ADTs that our group used and implemented was the required Linked List, arrays for storing miscellaneous information, and vectors for storing larger amounts of information. Firstly, our Linked List was a Doubly Linked List, where a node would have a reference to both the next and previous node. This design choice was implemented to ensure that our other functions could go back and forth between nodes so that a specific data piece may be found. While our methods primarily utilized the head node to add tiles to the linked list, the tail was used instead if we wanted to add to the end of the linked list. While having a standard Linked List with just the head may have been fully sufficient, it may have not been the most efficient at adding and deleting nodes compared to a Doubly Linked List.

For the implementation of arrays, they were simply used to record basic information, such as student information and the tiles' characteristics, since they were easier to manipulate into other data structures. Furthermore, there was no real circumstance to utilize arrays for larger scale implementation, since STL containers are much more efficient for larger scale memory storage, and STL containers are generally more straightforward to process when evaluating code.

Vectors were primarily used for storing players and the board state within our game implementation. Firstly, since vectors are easily resizeable by construct, the amount of players added by the user may change from game to game, which allows for easier code implementation and readability. If arrays were used, then we would have to create different amounts of memory every time manually through the code, which may be more inefficient compared to the self-processing nature of vectors. Secondly, using vectors for the board state allowed us to have a more easier time resizing the board after base implementation, whereas numerous issues may have arisen if we kept to a 2D array board or other similar memory construct.

2. Software Design & Efficiency

Our solution for implementing a C++ Qwirkle game was to focus on creating several classes that were responsible for specific parts of the game. From the simple classes such as Nodes and Tiles that were responsible for storing their information and characteristics, to the more complex Board and GameEngine classes that had a much bigger role in the game's state, each class had a defined role to execute and ensure the program ran correctly. Furthermore, our functions relied on other specific functions from other classes to complete properly, which allowed for higher levels of coupling between classes, which is not necessarily a bad feature in this implementation. For example, the Menu class was responsible for outputting most of the text output to the console, which meant that the other classes must rely on it to provide text output. The GameEngine is the class that has access to

all the classes so that the game works properly, so it doesn't execute the required functions by itself, but calls other functions to figure out how the game state is. So while coupling may seem high, this is all for the purpose of ensuring the game happens in once place, whilst separating the work between multiple classes.

User input was a feature that we had a slight focus upon, as we provided a help menu and visual cues for the player to utilize so that they play correctly. While we did provide all the help that we can for the players to play Qwirkle, it is also up to them to provide the proper input at the proper places. There is only so much a program can handle from the programmer's side, but the user must be able-minded to utilize the software in the correct manner.

Our program's runtime during testing seems to be running smoothly, which may indicate that the game isn't taking up much memory, which should be the case. Yet, the code implementation may not seem to suggest that, as some of our functions had to call other functions to complete an objective, which reduces the running speed of the program and may cause lag if this was a large-scale program. While some of the program seems efficient, such as using a Doubly Linked List or vectors for the board, calling upon other functions while running at the same time is definitely not the most efficient a program can be. Therefore, we should reduce coupling in order to increase efficiency, so that our functions are only allowed to do exactly one method. Yet, that might not have the best of code readability, so we rather implemented with calling other functions in order to complete one objective.

3. Test Case Effectiveness

Our test cases covered the core aspects of the game, which involved creating/loading/saving a game that involved 1 to 4 players, ensuring the menu options worked correctly, placing/replacing tiles and checking if they worked correctly, and finally the game ends correctly. These test cases ensure that we cover the majority of the base game's input options. Nonetheless, the test cases don't cover EOF char usage, but that is more reliant on the user's personal input rather than testing if it fully works or not. Therefore, the test cases do cover the base implementation and options associated with it, but not every single input possible, such as EOF char input.

4. Group Management

Our group mostly communicated through Discord to ask for help and ensure everybody knows what they were doing. Trello was the secondary means to show what functions and features needed to be implemented within the program, with it gradually being completed once we got closer to the due date. But primarily, Discord was the medium that

all members used to communicate. Github was also the primary means of exchanging code, since it is probably the simplest means of ensuring code quality is correct and git flow is easy to recognize. Therefore, Discord and Github were the primary applications that were used in the implementation of this game, with Trello as a secondary place for initial ideas and work separation.