**Advance Programming Techniques Assignment 2 Report:**

In our Qwirkle game implementation we have done:

ADTs, LinkedLists, arrays and vectors:

* Created ADTs for the LinkedList, Node, Tile, Board, Game, Bag, Player.
* In our LinkedList we provided functions to implement in the implementing files. These functions are:
* size function to get the size of the LinkedList
* clear function to clear the LinkedList
* get function to get the tiles of the LinkedList
* Add front to add the tiles to the front of the list
* add back to add the tiles to the back of the list
* add at to add the tiles at any specific index of the list
* delete at to delete any tiles from a specific index of the list to provide for constant time instead of the traditional linear time which the LinkedList without this method would grab nodes through linear time.
* Contains method which allows to get the size of the list of tiles
* And delete data function deletes a specific tile from the Linked List
* In terms of implementing the bag of tiles we used a 2D array to iterate through both the colours and shapes of the tile and adding it to the back of the LinkedList and add the size of the bag size by 2 to make sure all pieces of Qwirkle is in the bag twice.
* In terms of implementing the board we implemented it by using a 2D array of pointers to tiles and iterating through the array by for loop for the rows and cols of the board. The efficiency of this is that it allows for quick access to the pointers of tiles into the board the downside to this is that it is iterating through the rows and cols which may be inefficient
* In terms of the look of the board we opted to use nested for loops to iterate through the rows and cols and input the empty spaces which would create the foundation for our pointers to replace the empty spaces thus simulating the gameplay of Qwirkle in the board. In terms of efficiency it is not very efficient as for loops are used to iterate through all rows and cols to print the empty spaces onto the board with lines and dashes to represent the board and ASCII characters to represent the rows and the integers to represent the cols.
* For our gameplay we used separate class to house the implementation of our gameplay functions in order to better facilitate the styling code of our program we provided functions in the ADT of game:
* The getBoard, getPlayer1 and getPlayer2, getTileBag function to be used in other functions in order to simulate gameplay
* replaceTile function to replace tiles in players hand
* add tile to board function to add scores to the current player during gameplay
* hasGameEnded function allows for us to check if game has ended and if it does return true
* find score function finds the score from the current player and depending on the move they make
* the add Tile to Board function holds a lot of the logic to the qwirkle game, as it decides where the tiles on the board should be placed, if they can be placed, and if they are placed, and what points to add to the players. Functions such as find Score and num Of Tiles help the Board function to complete its goal of adding tiles to the board
* In terms of our design of our software we opted to implement milestone 2 mostly through separate classes having specific set of instructions.
* We did implement one minor enhancement for Milestone 3 which was Better Invalid Input, as when the player attempts to do something Invalid, the program will tell them that want they got wrong, if it was that the tile isn’t in their hand, or a tile can’t be placed in that location.
* We opted to put all the gameplay functionality and menu functions in the qwirkle.cpp file with allowing the file to include the header files of all other headers
* In order for us to do this we created objects as references to each individual files when needed in order to provide efficiencies we decided to go with creating pointers and calling the LinkedList functions to implement the gameplay functionality specifically the player’s hand and the tile bag.
* With every object we created with the keyword “new” we deleted the pointers to them through deconstructors on specific implementing files and whilst also in the qwirkle.cpp file.
* Keeping a more simplistic approach to designing our software we used if and else statements on our menu function to look for the specified integer the user inputs to choose an option from the menu. By doing this approach we can easily and more quickly compare user inputs and to call a specific function as required.
* We also included some comments on above codes to give a more clearer understanding of what we intended to do with specific code or segment
* In our qwirkle.cpp for each individual function we provided its own variables and to pass them as parameters to objects we created as references to header files. In terms of efficiencies this can be said to be inefficient as we are constantly having to provide variables for different functions.
* For our test cases we covered as much grounds on what the user can input in our software as we can
* We have covered for test cases are:
* Testing for valid input tile placements and its output of providing the board layout and its correct tile and positioning on the board.
* A test for invalid tile input tile placements and for its output we provided a board and its incorrect tile and positioning on the board
* Another test for EOF input from user and its expected output of displaying on the console “Goodbye” and exiting the program
* A test case for valid player name and its expected output of displaying “Enter a name for player 2 (uppercase characters only) and the “> “ arrow symbol indicating a user prompt
* A test case for invalid player name when the player types a player name that is not uppercase or is the same name as another player we provided the output of this to printing “Player name is invalid” and will prompt for user input again until the input is valid
* A test case for valid replace tile and its expected output of tile in the bag has been replaced
* A test case for invalid replace tile and expected output of printing to the console “invalid tile to replace”
* A test case for getting a qwirkle score in the game, in which a game is loaded and the current player places a tile to get a qwrikle
* A test case for player typing in an invalid tile, the player tries to put a tile on the board that doesn’t exist
* A test case for the player trying to put tile on the board that isn’t in their hand
* We can say that we have tried to cover all scenarios where user can input into our program and have covered them with input and expected outputs
* In terms of our group coordination and project management we decided to work closely through github as a way of pushing our code to the same platform to collaborate on the assignment.
* We also used a communication platform in order to communicate among the group on the issues we were having and to ask questions of who can help fix the errors or bugs we came across.
* We coordinated fairly well by merging our gits regularly with each other and working on the game implementations.