Abrar Al-Sagheer s3707180
Elizabeth Tawaf s3723812
Erick Blucher s3723054

## //OBJECTIVE

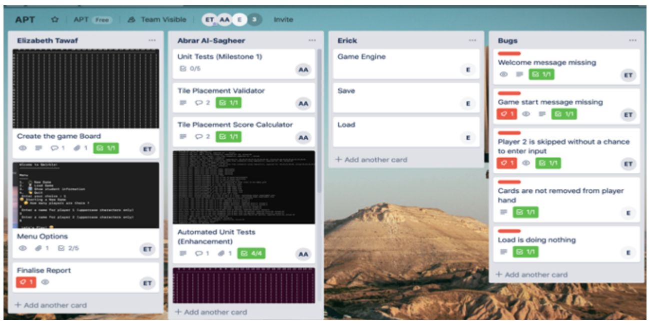
As a team of 3 students, this assignment was a great opportunity to learn and understand the C++ Language. As a team we decided that each one of us will work on their own class to reduce conflict; Weekly meetings were done after class sessions in order to discuss problems and each team member's role.

Our team split the components of the game so each one of us define our interfaces and implement our components. This allowed others in the team to use the methods of other classes through the interfaces while the implementation progressed. We found static class implementations were suitable for Validation and Score Calculation since they wouldn't need their own object to perform these tasks. it also made Automated unit tests easier. Below is a summary of the tasks each member performed.

Elizabeth (Resizable Grid , Menu, unit test, 3,4 players)||Abrar(validation, Colour Output, score calculations, Automated Tests) ||Erick (GameEngine, LinkedList unit test, save, load game)

## // Trello

A Trello board was created and used for time management and efficiently (A link to our Trello https://trello.com/invite/b/8AutwshA/20a4a697e28a08571577830e3abecfee/apt )
Trello was used as a To-do list where each member has their task and when it is done, they mark the task as done



Elizabeth Tawaf
- Create the game Board — 1 1 1/1 ET
- Menu Options — 1 2/5 ET
- Finalise Report — 1 ET

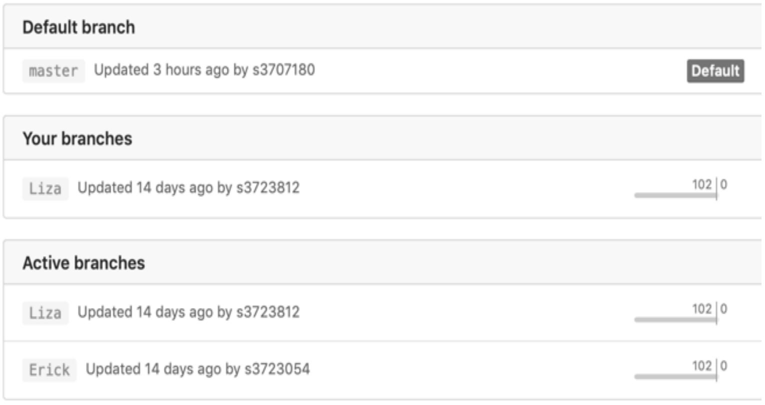
Abrar Al-Sagheer
- Unit Tests (Milestone 1) — 0/5 AA
- Tile Placement Validator — 2 1/1 AA
- Tile Placement Score Calculator — 2 1/1 AA
- Automated Unit Tests (Enhancement) — 1 1 4/4 AA

Erick
- Game Engine — E
- Save — E
- Load — E

Bugs
- Welcome message missing — 1/1 ET
- Game start message missing — 1 1/1 ET
- Player 2 is skipped without a chance to enter input — 1 1/1 ET
- Cards are not removed from player hand — 1/1 E
- Load is doing nothing — 1/1 E

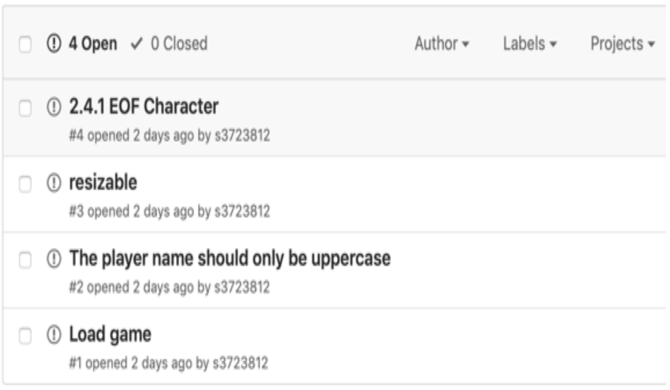
## // GitHub

A GitHub repository was created, at the start each member had their own branch, and as the game started to have a solid structure, all branches were merged in the master branch.


Default branch
master Updated 3 hours ago by s3707180 Default

Your branches
Liza Updated 14 days ago by s3723812 102 | 0

Active branches
Liza Updated 14 days ago by s3723812 102 | 0
Erick Updated 14 days ago by s3723054 102 | 0



4 Open ✓ 0 Closed

2.4.1 EOF Character
#4 opened 2 days ago by s3723812

resizable
#3 opened 2 days ago by s3723812

The player name should only be uppercase
#2 opened 2 days ago by s3723812

Load game
#1 opened 2 days ago by s3723812

## // Enhancements

- Colour tiles (minor)
- Emojis (Minor)
- Resizable Grid (minor)
- Better Invalid Input (minor)
- 3-4 players (Major)
- Automated Tests
  (Major "according to Amir")

## // Grid(Resizable, Enhancement)

As a group we decided to make the grid as a vector of vectors as we decided that we want the grid to be resizable, ( expands as the user enters the size ), the game starts with as 6x6 grid, and as the user enters a position that is on the edge of the current size, the grid adds another row and collumn to the right and bottom edge( until it reaches the maximum size which is 26x26).

First the method setTile(), which creates the vertical side of the grid ( alphabetical side )
Creating the tile on the grid was through getTile(); and finally, after we get both the tile and the alphabetical side of the grid , a toString() was used to append the appropriate game board.

## // GameEngine (medium performing logic on the user input)

The gameEngine class is the class which works with the cleansed input from the gameMenu(). It uses this input to apply logic and manipulate the given values to modify the data so that it matches what it should be after an action is performed. It is responsible for keeping the board and LinkedList, tracking the players, collating this information in a form that can be used by the gameMenu and do the actions by the gameMenu's request. When a player place's a tile, it determines whether it is a valid move, with the additional help of the Validator static class. This is done by using a series of if statements to check if the move can be made.

If at any point it finds that the move is invalid, an errors flag is modified and the return value is set to false. When a person replaces a tile it ensures that the tile is removed from the player's hand and that a new tile is removed from the tilebag. The gameEngine is also responsible for keeping the tile objects and storing their pointers, as we believed instantiating them on the heap all in one place would be the easiest way to store them and destroy them when necessary, as well as preventing making duplicates on the heap of these tiles as only 1 of each unique tiles is needed to be instantiated, even if there are multiple tiles in the tileBag that references this tile.

We also check to see if a player has no valid moves, to prevent a no-win situation, where the player can't do a move and the tilebag is empty. This isn't explicitly part of the assignment or game rules, but it felt necessary nonetheless.

## // Menu ( user input/output/validation)

The run() is the starting point of the game, where it calls the right methods in order to save, load, show the student information or quit the game. The newGame() and loadGame() both work to start off the initial settings of the games while the play game is the main interface between the player and the game as it takes the user input and passes it to other methods to make moves.
The player can decide the number of players when they start a new game(major enhancement) and the game can be loaded from a previous saved file.

The main menu validates the user's input and it handles the EOF (^D) character which ends the game if given by the user. We also decided to have all the valid commands available under the board just to make the game more user friendly. Load and save the game are called from the gameEngine which checks for the user's input and if the file exists. Any large chunks of information required from the gameEngine is collated into the one string by the game engine to be output by the gameMenu.

## // Validation

The validation is performed by the Validator static class. It is called prior to adding a tile to the board. This is to ensure the new placement follows the game rules. A simple bool is returned indicating valid/invalid placement. The validation method first tries simple tests like checking if the tile is inside the board or out of its bounds. After that, it tries to check if the various game rules

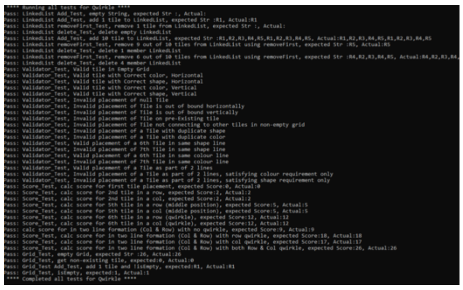
apply by investigating the neighbouring tiles first vertically, then horizontally. This makes it efficient as it limits the number of tiles investigated. The number of tiles in both the vertical and horizontal lines are calculated assuming its valid. Using this count, and by looking at the individuals tiles held by each position we can determine if the placement is valid. There are special cases like the first tile in the game that does not need to follow some of these validation rules.

## // Score Calculation

The score calculation method assumes the tile placement is already validated and just focuses on calculating the score. This improves efficiency as the actual tile's data become irrelevant to the calculation, just the presence of a tile. It is part of the Validator class and works in a similar fashion to the validation in the sense that it counts the tiles vertically and horizontally then depending on the numbers it calculates the score. Due to the similarity of the code with the validation method, this was placed with the validation class.

## // LinkedList

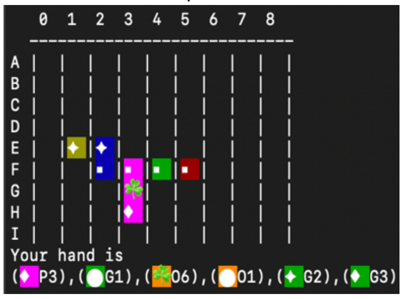
The LinkedList class is what is used to store the nodes, containing a pointer to the tiles that they contain. We had the opportunity to change this into a template, with the nodes possessing no predefined data type, however, due to time constraints it was unfeasible to implement that, for the little gain it would do. We kept the LinkedList limited to the methods that we would need, there are various tasks we could have had it perform, however, none of these would have had any bearing or impact on our program, so we elected to keep them out of the linked list class to improve the readability and make the object more concise. We have a removeFirst() method which returns the data contained in the first node. We elected to have it return this since it would be very useful, saving us from needing to check



```
**** Running all tests for Qwirkle ****
Pass: LinkedList Add_Test, empty String, expected Str :, Actual:
Pass: LinkedList Add_Test, add 1 tile to LinkedList, expected Str :R1, Actual:R1
Pass: LinkedList removeFirst_Test, remove 1 tile from LinkedList, expected Str :, Actual:
Pass: LinkedList delete_Test, delete empty LinkedList
Pass: LinkedList Add_Test, add 10 tile to LinkedList, expected Str :R1,R2,R3,R4,R5,R1,R2,R3,R4,R5, Actual:R1,R2,R3,R4,R5,R1,R2,R3,R4,R5
Pass: LinkedList removeFirst_Test, remove 9 out of 10 tiles from LinkedList using removeFirst, expected Str :R5, Actual:R5
Pass: LinkedList delete_Test, delete 1 member LinkedList
Pass: LinkedList removeFirst_Test, remove 6 out of 10 tiles from LinkedList using removeFirst, expected Str :R4,R2,R3,R4,R5, Actual:R4,R2,R3,R4,
Pass: LinkedList delete_Test, delete 4 member LinkedList
Pass: Validator_Test, Valid tile in Empty Grid
Pass: Validator_Test, Valid tile with Correct color, Horizontal
Pass: Validator_Test, Valid tile with Correct shape, Horizontal
Pass: Validator_Test, Valid tile with Correct color, Vertical
Pass: Validator_Test, Valid tile with Correct shape, Vertical
Pass: Validator_Test, Invalid placement of null Tile
Pass: Validator_Test, Invalid placement of Tile is out of bound horizontally
Pass: Validator_Test, Invalid placement of Tile is out of bound vertically
Pass: Validator_Test, Invalid placement of Tile on pre-Existing tile
Pass: Validator_Test, Invalid placement of Tile not connecting to other tiles in non-empty grid
Pass: Validator_Test, Invalid placement of a Tile with duplicate shape
Pass: Validator_Test, Invalid placement of a Tile with duplicate color
Pass: Validator_Test, Valid placement of a 6th Tile in same shape line
Pass: Validator_Test, Invalid placement of 7th Tile in same shape line
Pass: Validator_Test, Valid placement of a 6th Tile in same colour line
Pass: Validator_Test, Invalid placement of 7th Tile in same colour line
Pass: Validator_Test, Valid placement of a Tile as part of 2 lines
Pass: Validator_Test, Invalid placement of a Tile as part of 2 lines, satisfying colour requirement only
Pass: Validator_Test, Invalid placement of a Tile as part of 2 lines, satisfying shape requirement only
Pass: Score_Test, calc score for first tile placement, expected Score:0, Actual:0
Pass: Score_Test, calc score for 2nd tile in a row, expected Score:2, Actual:2
Pass: Score_Test, calc score for 2nd tile in a col, expected Score:2, Actual:2
Pass: Score_Test, calc score for 5th tile in a row (middle position), expected Score:5, Actual:5
Pass: Score_Test, calc score for 5th tile in a col (middle position), expected Score:5, Actual:5
Pass: Score_Test, calc score for 6th tile in a row (qwirkle), expected Score:12, Actual:12
Pass: Score_Test, calc score for 6th tile in a col (qwirkle), expected Score:12, Actual:12
Pass: calc score for in two line formation (Col & Row) with no qwirkle, expected Score:9, Actual:9
Pass: Score_Test, calc score for in two line formation (Col & Row) with row qwirkle, expected Score:18, Actual:18
Pass: Score_Test, calc score for in two line formation (Col & Row) with col qwirkle, expected Score:17, Actual:17
Pass: Score_Test, calc score for in two line formation (Col & Row) with both Row & Col qwirkle, expected Score:26, Actual:26
Pass: Grid_Test, empty Grid, expected Str :26, Actual:26
Pass: Grid_Test, get non-existing tile, expected:0, Actual:0
Pass: Grid_Test Add_Test, add 1 tile and !isEmpty, expected:R1, Actual:R1
Pass: Grid_Test, isEmpty, expected:1, Actual:1
**** Completed all tests for Qwirkle ****
```

## // Automated Tests (<u>Enhancement</u>)

Automated tests are implemented to check the main application's functionality. These tests helped us regression test continuously as we developed the game. We added a flag to enable and disable this in OperationMode.h header. If this flag is checked in the main function, at the runtime the tests are performed instead of running the game normally. When the tests are enabled a number of tests are run where each test class is responsible for a specific component like LinkedList, Grid, Validation etc. Each test creates the given component and then performs a task on it, comparing the resulting data to that of which is expected. If the result matches, it prints pass and otherwise, it prints fail with details for failures. This is modelled after the Java JUnit tests framework. There are various unit test frameworks available for C++ like "Google Test" but our implementation seems adequate for the purpose of the assignment without adding too much complexity.



```
  0   1   2   3   4   5   6   7   8
 ------------------------------------
A |   |   |   |   |   |   |   |   |   |
B |   |   |   |   |   |   |   |   |   |
C |   |   |   |   |   |   |   |   |   |
D |   |   |   |   |   |   |   |   |   |
E |   | ✦ | ✦ |   |   |   |   |   |   |
F |   |   | ■ | ■ | ■ | ■ |   |   |   |
G |   |   |   | ✤ |   |   |   |   |   |
H |   |   |   | ♦ |   |   |   |   |   |
I |   |   |   |   |   |   |   |   |   |
Your hand is
(♦P3),(●G1),(✤O6),(●O1),(✦G2),(♦G3)
```

## // Colour Output and Emojis (<u>Enhancement</u>)

We implemented the colouring of the tiles as one of optional enhancements. We used ANSI escape sequences to achieve this. The results are not visible in terminals not supporting this like Windows command line. We added a switch to enable and disable this in OperationMode.h header.

## // Better Invalid Input (<u>Enhancement</u>)

A custom error message is designed to help the user know where the error is. For example if the user tries to trade a tile that does not exist, they get this message.

```
Tile given is not in hand, make sure you enter a valid tile
```

And if the user tries to place a tile that does not match the colour or the shape of the sequence, they would get this message.

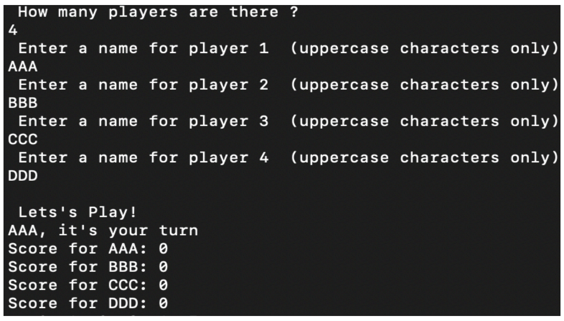
```
Validation Error: Shape and Color in this line are not matching the new tile
```

## // Unit tests

Unit tests were implemented to cover a wide variety of scenarios and ensure that many of the functions that can be performed have results that match what they are meant to. They work by checking the expected output against the actual output for differences to help find where things go wrong. As such, when we created these tests we had to use a specific load file to test with rather then using the new game method. The output that would occur from the new game method would have too much random data, as the tile bag is required to be random, and as such won't be able to match the commands and output correctly without loading in a game.

## // 3-4 Player modes (Enhancement)

The game supports up to 4 players, thus when saving the game, we have to loop through each player to print out their data. When loading in, the game checks whether the line entered matches that of the top of the board, and if it is found to be it knows that the next entry is no longer a player and all players are entered, and as such can load in the board and start the game.



```
 How many players are there ?
4
 Enter a name for player 1  (uppercase characters only)
AAA
 Enter a name for player 2  (uppercase characters only)
BBB
 Enter a name for player 3  (uppercase characters only)
CCC
 Enter a name for player 4  (uppercase characters only)
DDD

 Lets's Play!
AAA, it's your turn
Score for AAA: 0
Score for BBB: 0
Score for CCC: 0
Score for DDD: 0
```

## // ASSUMPTIONS

- We assumed that the Grid does not get bigger than 26x26.
- The grid only needs to be resizable from the south and east sides( according to Amir).
- The first tile place on the Grid is not added to the score as it is a free move.
- The user's name should only include uppercase (no spaces).
- The player can place a tile anywhere on the grid only if the grid is empty.
- The grid will always be a square.
- Number of players range from 2 to 4
- The players could continue playing the game after saving the game.
- In order to successfully load the game, we assumed that the length of each line of the board is the same. If it isn't, then the data stored in the save isn't considered a valid board as it doesn't match the conventions of the save file.

## //Efficiency

As a group we decided that a vector is required to build the grid, vectors utilize continuous storage positions to save elements. They can be managed, saved and grow dynamically in an efficient way. Vectors grow dynamically thus this featured helped us to do the resizable Grid

We elected to instantiate all the tiles in the one spot, being the game engine. This was decided since it would allow us to be more efficient with the space that would need to be allocated in the heap; only 1 tile is created for each unique tile and everything that needs to access these is given a pointer to said tile. This means, that while there are 2 copies of each tile present in the game, there aren't 2 spots in the heap allocated for these same tiles. It also allows for future expansion, adjusting the game so there are much more tiles will be a simple matter of changing a constant, and no extra space will be allocated for these tiles, just for the nodes that contain their pointers.

## //Issues faced

Loading the game was one of the most challenging part of the assignment as after certain lines another line was expected, and an end of file check was necessary to prevent the program from reading in non-existent lines if the save files were corrupt or modified. We also needed to take the data in the form it was given and change it so that we could match it with t he actual data it represents rather than just the string it shows. This meant for lines such as hands and tilebag, we needed to know how to break up the string so that each of the individual tiles were found in turn, and checked to make sure they matched with an existing tile
Other small bugs were usually posted on Trello and GitHub.

## // Conclusion

It was a great opportunity to learn and explore how different team members handle similar problems differently. We Have decided that as a team, we will grow the game and add more features to the solid base we have built.